# Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints: The Case of 3 Recursive Rules

Laurent FRIBOURG
Marcos VELOSO PEIXOTO

Laboratoire d'Informatique, URA 1327 du CNRS
Département de Mathématiques et d'Informatique
Ecole Normale Supérieure

LIENS - 94 - 9

July 1994

# Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints: The case of 3 Recursive Rules

Laurent Fribourg Marcos Veloso Peixoto

45, rue d'Ulm, 75005 Paris - France
L.I.E.N.S (URA 1327 CNRS)
email: fribourg@dmi.ens.fr, veloso@dmi.ens.fr

## Abstract

Datalog Programs with arithmetic constraints have received considerable attention during the last few years. They have been studied as special forms of Constraint Logic Programming languages, as models for temporal databases, and also as tools for Theorem Proving.
We define in this paper a class of Datalog programs over integers for which the bottom-up evaluation always produces a linear arithmetic formula. A program within this class is composed of three recursive rules. Each rule contains one arithmetic atom in its body and increments its arithmetic arguments.
We adopt a geometric approach to construct the output of the bottom-up evaluation process: each application of the immediate consequence operator T is represented either as an horizontal movement, or as a vertical movement, or as a transversal movement.
In this report, we are more interested in showing that the output of this programs can be expressed by a (finite) arithmetic formula, rather than expressing it explicitly.

## Résumé

Les bases de données avec contraintes arithmétiques ont reçu une attention consi-dérable depuis ces dernières années. Elles ont été étudiées à la fois comme un genre nouveau de langage de Programmation avec Contraintes, comme des modèles pour les bases de données temporelles et comme outils de Démonstration Automatique.
Nous définissons dans ce papier une classe de programmes Datalog sur des entiers dans laquelle le processus d'évaluation ascendante engendre toujours une formule arithmétique linéaire. Un programme appartenant à cette classe contient trois règles récursives, qui possèdent, chacune, un seul atome arithmétique dans le corps, et qui incrémentent ses arguments arithmétique.
Nous adoptons une approche géométrique pour construire le résultat du processus d'évaluation ascendante: chaque application de l'opérateur de conséquence immédiate T est représentée soit par un déplacement horizontal, soit par un déplacement vertical, soit par un déplacement transversal.
Dans ce rapport, nous nous intéressons à montrer que le résultat de cette évaluation ascendante peut être exprimé par une formule arithmétique finie, mais nous ne chercherons pas à caractériser explicitement cette formule.

## 1 Motivations

There are two basic approaches for reasoning with logic programs. The first approach is "top-down": it tries to solve a given query by backward chaining reasoning. The second approach is "bottom-up": it infers atomic consequences of the program by forward chaining reasoning. The first approach is rather used in Logic Programming (SLD-resolution) [21], while the second one is rather used in Deductive Databases for querying Datalog programs (i.e., logic programs with constants and variables, but no function symbol) [4].
We describe in this paper a bottom-up evaluation process for Datalog programs (or logic programs) whose variables are interpreted as integers. Our initial motivation was to use such a process in order to prove formulas combining lists and linear arithmetic [10]. Let us illustrate this point with an example which corresponds to an adaptation of the Boyer-Moore Majority Algorithm (see [24]). This algorithm determines the element of a list that occurs more than half times the size of the list, if such an element exists.

Let $p(L, v, w, x, y, z)$ be an atom where $L$ is a list of (numeric) characters, $y$ is the length of $L$, $x$ is the number of occurrences of a *possible* majority item $v$ and $z$ is the number of occurrences of an item $w$ in $L$. The predicate $p$ can be recursively defined by the logic program:

$$\begin{array}{lll} p([\,], v, w, x, y, z) & :- & x = 0,\ y = 0,\ z = 0 \\ p([u|L], v, w, x+1, y+1, z) & :- & u = v, u \neq w,\ p(L, v, w, x, y, z) \\ p([u|L], v, w, x, y+1, z+1) & :- & u \neq v, u = w, 2x > y,\ p(L, v, w, x, y, z) \\ p([u|L], v, w, x, y+1, z) & :- & u \neq v, u \neq w, 2x > y,\ p(L, v, w, x, y, z) \end{array}$$

Suppose now that we want to prove the formula:

$$\forall\ L, v, w, x, y, z\ \ p(L, v, w, x, y, z) \implies (v \neq w \implies z \leq y\ div\ 2)$$

It is difficult to prove this formula directly (e.g., by induction), but it becomes easy if we are provided with a lemma saying that $p(L, v, w, x, y, z)$ implies $v \neq w \Rightarrow 2x \geq y \wedge\ y \geq z + x \wedge x \geq 0$. A priori such a lemma is not known, but a bottom-up evaluation process can automatically generate such a lemma. The bottom-up evaluation process is not performed directly on the $p$ program, but instead over a simplified form (that is merely obtained by deletion of the list argument $L$ and its argument $u$ and by "pushing" the constraints on $v \neq w$):

$$\begin{array}{lll} p'(v, w, y, z) & :- & x = 0,\ y = 0,\ z = 0 \\ p'(v, w, x+1, y+1, z) & :- & p'(v, w, x, y, z) \\ p'(v, w, x, y+1, z+1) & :- & 2x > y,\ p'(v, w, x, y, z) \\ p'(v, w, x, y+1, z) & :- & 2x > y,\ p'(v, w, x, y, z) \end{array}$$

A refined form of bottom-up evaluation over $p'$ that we will explain in this paper produces the relation $2x \geq y \wedge\ \ y \geq z + x \wedge x \geq 0$ as a generic output. The lemma $p(L, v, w, x, y, z) \implies (v \neq w \implies z \leq y\ div\ 2)$ then follows directly from the fact that, by construction, $p(L, v, w, x, y, z))\ wedgev \neq w$ implies $p'(v, w, x, y, z)$.
Apart from the automatic generation of lemmas, an interesting application of bottom-up evaluation of Datalog programs over integers is the proof of termination of logic programs [5, 13]. From a more general point of view, bottom-up evaluation procedures for Datalog

programs over integers constitute an interesting subject of research *per se* and have been extensively studied in recent years. One reason for this interest is the emergence of Constraint Logic Programming and Constraint Query Languages (e.g. [15, 16, 18, 19, 22, 23, 27, 29]). Another reason lies in the use of integers for representing temporal data and the use of bottom-up evaluation for computing temporal queries [2, 3, 8].
The rest of this report is organized as follows:
In sections 2 we compare our results with related works, in section 3 and 4 we present some basic definitions. In section 5 we explain our method for programs with two recursive rules and in section 6 for programs with three recursive rules. Section 7 contains some optimizations of our method. In section 8 we presents some applications of our method and section 9 contains the conclusion of this report.

## 2 Comparison with related work

Bottom-up evaluation procedures for Datalog programs with integers have been developed by researchers of the Deductive Database community [3, 8, 27] on one hand, and by researchers interested by the proof of termination of Prolog programs [5, 13] on the other hand.
Revesz has elaborated a procedure that always terminates for a class of Datalog program with integers, but this class does not allow for the incrementation of the recursion arguments [27]. Chomicki and Imielinski have considered a class of programs that is useful for modeling time in Temporal Databases: time is represented by a numeric argument ("temporal" argument) which is decremented at each recursive call. The class of Chomicki and Imielinski allows for only one temporal argument [8]. Baudinet, Niézette and Wolper consider a class of Datalog programs that allows for several temporal arguments, but their procedure does not terminate in general; besides, the tuples of the extensive database of their programs (i.e., the tuples satisfying the non recursive rules) correspond to linear arithmetic formulas of a restricted form ("linear repeating points") [3].
As in Baudinet, Niézette and Wolper [3], our programs allow for incrementation over several recursion arguments. In contrast to [3], our programs can take any linear arithmetic formula as for an extensive database. We have besides identified a subclass of programs for which our bottom-up evaluation procedure is guaranteed to terminate (class of programs with three recursive rules and one inequality constraint per rule).
The idea of automatically generating linear inequalities among variables of a program was exploited in an imperative framework by [7], using linear programming techniques. Researchers interested by the mechanization of termination proofs have designed bottom-up evaluation procedures for Datalog programs with integers: given a Prolog procedure, the aim is to infer inequalities among the sizes of the arguments of the auxiliary predicates called by this procedure, and to use these inequalities for proving the procedure termination [5, 13]. The method of [5] is able to infer disjunctions of sets of inequalities, but each inequality is necessarily of the form $arg(i) + c > arg(j)$, thus involving at most two arguments. The method of [13] can infer sets of more general inequalities (e.g., $arg(i) + arg(j) > arg(k) + c$), but cannot infer disjunctions of such formulas. This is also the case in the work of [7] and in methods that use top-down strategies instead of bottom-up [26, 30]. In contrast, our method has no such limitation, and is able to infer the arithmetic formula that is characteristic of the program.
The main limitation of our method comes from the syntactical restrictions attached to the class of programs that we consider: the recursion scheme is necessarily direct and linear (no

mutual recursion, no multiple recursive calls), and within each recursive rule, the recursion arguments are necessarily increased by a constant. We focus in this report on the case where the programs have (at most) 3 recursive rules. We describe a method, based on bottom-up evaluation, which allows to transform the recursive programs into linear arithmetic formulas. These arithmetic formulas will not be given explicitly, but only represented under a geometric form. The explicit arithmetic formulations and subsequent possible simplifications (using, e.g., techniques of variable elimination [20, 28]) are beyond the scope of the present report.

## 3 Preliminaries

Our goal is to prove that the output of the bottom-up evaluation of programs $\pi$ defined by a non-recursive base rule $R_0$ and three linear recursive rules can be expressed as a finite arithmetic formula. More precisely, we want to show that an atom $p(u, v, w)$ belongs to the output of the bottom-up evaluation of $\pi$ iff $u, v, w$ satisfy a finite linear arithmetic formula.

We consider that our programs are of the form:

$$\begin{array}{lll l}
p(x, y, z) & :- & \xi(x, y, z). & \text{rule } R_0 \\
p(x + a_1, y + b_1, z + c_1) & :- & \Phi_1(x, y, z),\ p(x, y, z) & \text{rule } R_1 \\
p(x + a_2, y + b_2, z + c_2) & :- & \Phi_2(x, y, z),\ p(x, y, z) & \text{rule } R_2 \\
p(x + a_3, y + b_3, z + c_3) & :- & \Phi_3(x, y, z),\ p(x, y, z) & \text{rule } R_3
\end{array}$$

where $a_1,\ a_2,\ a_3,\ b_1,\ b_2,\ b_3,\ c_1,\ c_2,\ c_3\ \in \mathbb{Z}$ and $\xi(x, y, z)$ is a linear arithmetic formula, and $\Phi_k(x, y, z)$ denotes respectively the constraints $d_k x + e_k y + f_k z + g_k \odot_k 0$, with $k \in \{1, 2, 3\}$, $\odot_k \in \{<, >, \leq, \geq\}$ and $d_k, e_k, f_k, g_k\ \in \mathbb{Z}$.

In this report, for the sake of algebraic simplicity, we will let $\Phi_1(x, y, z)$, $\Phi_2(x, y, z)$ and $\Phi_3(x, y, z)$ be respectively $x \geq 0$, $y \geq 0$ and $z \geq 0$. The results presented in this paper remain valid for constraints of the form $d_k x + e_k y + f_k z + g_k \odot_k 0$.

For the sake of notation simplicity, we have also assumed that the predicate $p$ has arity 3. (Our results remain valid for any other arity.)

We use arithmetic formulas to extend the notion of Herbrand atom. We define a *generalized Herbrand atom* as a pair composed of an atom and an arithmetic formula. Note that an Herbrand atom of the form $p(6, 5, 7)$ can be considered as a generalized Herbrand atom composed of $p(x, y, z)$ together with $x = 6 \wedge y = 5 \wedge z = 7$. The immediate consequence operator for the program $\pi$ (see [1, 9]) is adapted in order to be applied to generalized Herbrand atoms (*cf* [14]). Given a set $I$ of generalized Herbrand atoms, let, for $1 \leq k \leq 3$:

$$T_k(I) \ = \ \{\ p(x + a_k, y + b_k, z + c_k)/\ \Phi_k(x, y, z)\ \text{ holds and } p(x, y, z)\ \in\ I\}$$

The operator $T_k$ can be seen as the immediate consequence operator associated with the rule $R_k$ $(1 \leq k \leq 3)$. The immediate consequence operator $T$ associated with $\pi$ can be defined by:

$$T(I) \ = \ T_1(I) \ \bigcup \ T_2(I) \bigcup \ T_3(I)$$

Let now $T^s$ be defined inductively as:

$$\begin{array}{lll} T^1(I) & = & T(I) \\ T^{s+1}(I) & = & T(T^s(I)), \quad \text{for } s \geq 1 \end{array}$$

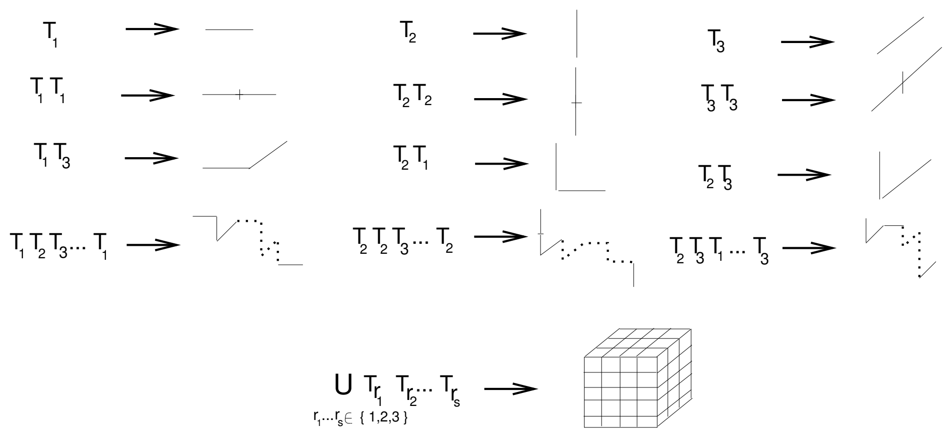
The output $S$ of the bottom-up evaluation of $\pi$ is $\bigcup_{s>0} T^s(I_\xi)$, where $I_\xi = \{\ p(x,y,z)\ /\ \xi(x,y,z) \text{ holds }\}$ . This output is the least fixpoint of the operator $T$. It is also the least generalized Herbrand model of $\pi$ [14, 9].

We will use a geometrical representation for computing the output of the bottom-up evaluation of $\pi$. This output can also be expressed as:

$$\bigcup_{r_1,r_2,\ldots,r_s \in \{1,2,3\}} T_{r_1}T_{r_2}...T_{r_s}(I_\xi)$$

We will associate a geometrical *path* connecting points of $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$ to each operator composition $T_{r_1}T_{r_2}...T_{r_s}$: each number 1 in the sequence $r_1r_2...r_s$ will be represented by a horizontal movement, each number 2 by a vertical movement and each number 3 by a transversal movement. This is schematized in figure 1.

For short, we will also refer to horizontal movements as $x$-moves, vertical movements as $y$-moves and transversal movements as $z$-moves.



$T_1 \rightarrow$
$T_2 \rightarrow$
$T_3 \rightarrow$
$T_1T_1 \rightarrow$
$T_2T_2 \rightarrow$
$T_3T_3 \rightarrow$
$T_1T_3 \rightarrow$
$T_2T_1 \rightarrow$
$T_2T_3 \rightarrow$
$T_1T_2T_3 \ldots T_1 \rightarrow$
$T_2T_2T_3 \ldots T_2 \rightarrow$
$T_2T_3T_1 \ldots T_3 \rightarrow$
$\bigcup_{r_1 \ldots r_s \in \{1,2,3\}} T_{r_1} T_{r_2} \ldots T_{r_s} \rightarrow$

Figure 1

Within a path, the origin (0,0,0) represents actually the tuple $< x, y, z >$ and each point $M$ of coordinates $(\alpha, \beta, \gamma)$ represents the tuple $< x + \alpha a_1 + \beta a_2 + \gamma a_3, y + \alpha b_1 + \beta b_2 + \gamma b_3, z + \alpha c_1 + \beta c_2 + \gamma c_3 >$ generated after $\alpha$ applications of rule $R_1$, $\beta$ applications of rule $R_2$ and $\gamma$ applications of $R_3$.

Formally, a *path* $\mathcal{P}$, denoted $\ll (0,0,0), ..., (\sigma, \theta, \upsilon) \gg$, is a finite sequence of points of $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$, starting at point $(0,0,0)$ and ending at point $(\sigma, \theta, \upsilon)$, such that if a point $(\alpha, \beta, \gamma)$ distinct from $(\sigma, \theta, \upsilon)$ is in $\mathcal{P}$, then either $(\alpha+1, \beta, \gamma)$ or $(\alpha, \beta+1, \gamma)$ or $(\alpha, \beta, \gamma+1)$ is in $\mathcal{P}$.

For each horizontal segment $(\alpha,\beta,\gamma)$-$(\alpha+1,\beta,\gamma)$ of a given path $\mathcal{P}$, there is an associated constraint $x+\alpha a_1+\beta a_2+\gamma a_3 \geq 0$ (denoted by $\phi_1(\alpha,\beta,\gamma)$), which represents the formula that must be satisfied for applying rule $R_1$ at point $(\alpha,\beta,\gamma)$. Likewise, for each vertical segment $(\alpha,\beta,\gamma)$-$(\alpha,\beta+1,\gamma)$, there is an associated constraint $y+\alpha b_1+\beta b_2+\gamma b_3 \geq 0$ (denoted by $\phi_2(\alpha,\beta,\gamma)$), which represents the formula that must be satisfied for applying rule $R_2$ and for each transversal segment $(\alpha,\beta,\gamma)$-$(\alpha,\beta,\gamma+1)$, there is a constraint $z+\alpha c_1+\beta c_2+\gamma c_3 \geq 0$ (denoted by $\phi_3(\alpha,\beta,\gamma)$), which represents the formula that must be satisfied for applying rule $R_3$.

Let $M=(\alpha,\beta,\gamma)$ and $M'=(\alpha',\beta',\gamma')$ be two points. We say that the point $M$ is located before $M'$, and write $M < M'$, iff $(\alpha,\beta,\gamma) \neq (\alpha',\beta',\gamma')$ and $\alpha \leq \alpha'$, $\beta \leq \beta'$, $\gamma \leq \gamma'$. The constraint $\phi_i(\alpha,\beta,\gamma)$ of a point $M=(\alpha,\beta,\gamma)$ will be also written sometimes as $\phi_i(M)$.

We say that a constraint of the point $M$, say $\phi(M)$, is *stronger* than a constraint of the point $M'$, say $\phi(M')$, iff $\phi(M) \Rightarrow \phi(M')$. We say also that $\phi(M')$ *subsumes* $\phi(M)$, which is denoted, at the risk of a notational abuse, by $\phi(M) \leq \phi(M')$

Similarly, at the risk of a notational abuse, the formula:

| | | |
|---|---|---|
| $\phi(M) \geq \phi(M')$ | denotes that | $\phi(M') \Rightarrow \phi(M)$ |
| $\phi(M) = \phi(M')$ | denotes that | $\phi(M') \Leftrightarrow \phi(M)$ |
| $\phi(M) > \phi(M')$ | denotes that | $\phi(M') \Rightarrow \phi(M) \wedge \phi(M) \not\Rightarrow \phi(M')$ |
| $\phi(M) < \phi(M')$ | denotes that | $\phi(M) \Rightarrow \phi(M') \wedge \phi(M') \not\Rightarrow \phi(M)$ |
| $\phi(M) \neq \phi(M')$ | denotes that | $\phi(M) > \phi(M') \vee \phi(M) < \phi(M')$ |

We say that a constraint $\phi$ is *strictly increasing* iff $\forall M,\ \forall M' > M\ \phi(M) < \phi(M')$ and that $\phi$ is *strictly decreasing* iff $\forall M,\ \forall M' > M\ \phi(M) > \phi(M')$. We say that $\phi$ in *strictly monotonic* iff $\phi$ is either *strictly increasing* or *strictly decreasing.*

The *global constraint* $C_{\mathcal{P}}$ associated with a path $\mathcal{P}$ from the origin $(0,0,0)$ to some point $(\sigma,\theta,\upsilon)$ is the conjunction of the constraints of all the segments of $\mathcal{P}$. It contains $\sigma+\theta+\upsilon$ atoms. Formally, it is defined as:

$$C_{\mathcal{P}} \stackrel{\text{def}}{=} C_{\mathcal{P}}(\sigma,\theta,\upsilon)$$

where:

$$\begin{array}{llll}
C_{\mathcal{P}}(0,0,0) & \stackrel{\text{def}}{=} & \text{true.} & \\
C_{\mathcal{P}}(\alpha+1,\beta,\gamma) & \stackrel{\text{def}}{=} & C_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \phi_1(\alpha,\beta,\gamma) & \text{if } (\alpha+1,\beta,\gamma) \in \mathcal{P}. \\
C_{\mathcal{P}}(\alpha,\beta+1,\gamma) & \stackrel{\text{def}}{=} & C_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \phi_2(\alpha,\beta,\gamma), & \text{if } (\alpha,\beta+1,\gamma) \in \mathcal{P}. \\
C_{\mathcal{P}}(\alpha,\beta,\gamma+1) & \stackrel{\text{def}}{=} & C_{\mathcal{P}}(\alpha,\beta,\gamma) \wedge \phi_3(\alpha,\beta,\gamma), & \text{if } (\alpha,\beta,\gamma+1) \in \mathcal{P}.
\end{array}$$

We denote the set of all the paths $\mathcal{P}$ from the origin $(0,0,0)$ to the point $(\sigma,\theta,\upsilon)$ by $\Delta(\sigma,\theta,\upsilon)$. Note that this set of paths corresponds geometrically to a cube (see figure 2).

The disjunction of the global constraints of all the paths from $(0,0,0)$ to $(\sigma,\theta,\upsilon)$ is denoted by $\Gamma(\sigma,\theta,\upsilon)$, i.e.:

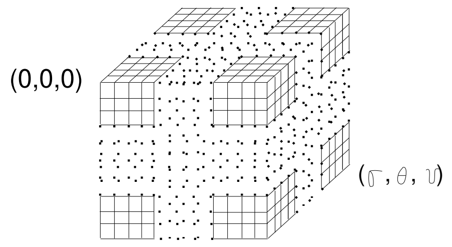


Figure 2:

$$\Gamma(\sigma,\theta,\upsilon) \ \overset{\text{def}}{=} \ \bigvee_{\mathcal{P}\in\Delta(\sigma,\theta,\upsilon)} C_{\mathcal{P}}$$

The output $S$ of the bottom-up evaluation of a program $\pi$ is given by:

$$S = \cup_{\sigma,\theta,\upsilon\in\mathbb{N}}\{\ p(x+\sigma a_1+\theta a_2+\upsilon a_3,\, y+\sigma b_1+\theta b_2+\upsilon b_3,\, z+\sigma c_1+\theta c_2+\upsilon c_3)\ /\ \xi(x,y,z)\ \wedge\ \Gamma(\sigma,\theta,\upsilon)\}$$

Our method for expressing $S$ as an arithmetic formula proceeds by simplifying $\Gamma(\sigma,\theta,\upsilon)$ in two steps:

**$\bigvee$-simplification:** we determine a subset $\Delta'(\sigma,\theta,\upsilon)$ of the cube $\Delta(\sigma,\theta,\upsilon)$, such that all the paths in $\Delta'(\sigma,\theta,\upsilon)$ have a characteristic form and subsume the other paths of $\Delta(\sigma,\theta,\upsilon)$ in the following sence:

$$\bigvee_{\mathcal{P}\in\Delta(\sigma,\theta,\upsilon)} C_{\mathcal{P}} \ = \ \bigvee_{\mathcal{P}\in\Delta'(\sigma,\theta,\upsilon)} C_{\mathcal{P}}$$

**$\bigwedge$-simplification:** we simplify $\bigvee_{\mathcal{P}\in\Delta'(\sigma,\theta,\upsilon)} C_{\mathcal{P}}$ to an arithmetic formula by reducing each conjunction of constraints $C_{\mathcal{P}}$, for all $\mathcal{P}\in\Delta'(\sigma,\theta,\upsilon)$.

We focus in this paper on the first step of the method. The second step of $\bigwedge$-simplification to an arithmetic formula does not pose any serious difficulty because of some monotonicity properties satisfied by the paths of $\Delta'(\sigma,\theta,\upsilon)$.

## 4 Periods and Patterns

The approach presented in this report basically relies on the notion of *periodic constraint* and on the associated notion of *pattern*. Roughly speaking, a constraint $\phi$ is periodic iff there exists a vector $\vec{\tau}$ such that for all point $M$: $\phi(M)=\phi(M+\vec{\tau})$. The pattern associated with $\phi$ is the set of all the paths linking a given point $M$ to the point $M+\vec{\tau}$. Without loss of understanding, we will also denote the pattern by the vector $\vec{\tau}$. In the following subsections, we formally define these notions in the case of programs with two and three recursive rules.

For programs with two recursive rules, let us consider a point $M$ of coordinates $(\alpha,\beta)$. The vector $\vec{\tau}$ will stand for $(\tau_1,\tau_2)$ and the expression $M+\vec{\tau}$ will stand for $(\alpha+\tau_1,\beta+\tau_2)$.

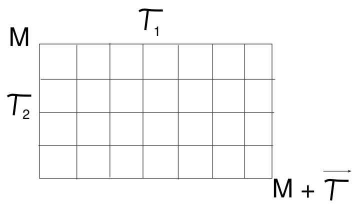
For programs with three recursive rules, let us consider a point $M$ of coordinates $(\alpha,\beta,\gamma)$. The vector $\vec{\tau}$ will stand for $(\tau_1,\tau_2,\tau_3)$ and the expression $M+\vec{\tau}$ will stand for $(\alpha+\tau_1,\beta+$

$\tau_2, \gamma + \tau_3)$.

Given a pattern $\vec{\tau}$ and a vector $\vec{\nu}$ ($\vec{0} \leq \vec{\nu} \leq \vec{\tau}$), the set of all the paths linking a given point $M$ to $M + \vec{\nu}$ will be called a *subpattern* of $\vec{\tau}$.

### 4.1 Two Recursive Rules

**Definition 4.1 (Periodicity)** *A constraint $\phi$ is* periodic *iff there exists a nonnull vector $\vec{\mu}$ of positive integers of the form $(\mu_1, \mu_2)$ such that:*
$\forall\alpha, \forall\beta \quad \phi(\alpha, \beta) = \phi(\alpha + \mu_1, \beta + \mu_2)$.
*The* period, *say $\vec{\tau}$, of $\phi$ is the smallest vector satisfying the above equation. The* pattern *associated with $\vec{\tau}$ (and also denoted $\vec{\tau}$) is the set of all the paths linking a given point $M$ to $M + \vec{\tau}$. See figure 3.*



$\phi(M) = \phi(M + \vec{\tau})$

*Figure 3: Pattern for $\phi$*

Given a period $\vec{\tau}$ for a constraint $\phi$, we will say that a constraint $\psi$ is *periodically increasing* (resp. *periodically decreasing*) iff $\psi(M) \leq \psi(M + \vec{\tau})$ (resp. $\psi(M) \geq \psi(M + \vec{\tau})$).

### 4.2 Three Recursive Rules

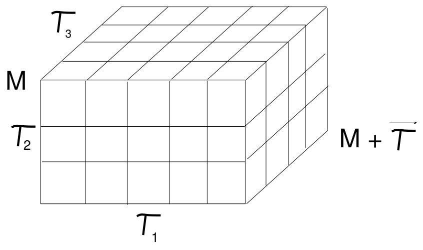
**Definition 4.2 (Co-periodicity)** *Two constraints $\phi$ and $\psi$ are* co-periodic *iff there exists a nonnull vector $\vec{\mu}$ of positive integers of the form $(\mu_1, \mu_2, \mu_3)$ such that:*

- $\forall\alpha, \beta, \gamma \quad \phi(\alpha, \beta, \gamma) = \phi(\alpha + \mu_1, \beta + \mu_2, \gamma + \mu_3)$.
  *and*
- $\forall\alpha, \beta, \gamma \quad \psi(\alpha, \beta, \gamma) = \psi(\alpha + \mu_1, \beta + \mu_2, \gamma + \mu_3)$.

*The* co-period, *say $\vec{\tau}$, of $\phi$ and $\psi$ is the smallest vector satisfying the above equations. The* co-pattern *associated with $\vec{\tau}$ (and also denoted $\vec{\tau}$) is the set of all the paths linking a given point $M$ to $M + \vec{\tau}$. See figure 4.*

Given a co-period $\vec{\tau}$ for two constraints $\phi$ and $\psi$, we will say that a constraint $\zeta$ is *co-periodically increasing* (resp. *co-periodically decreasing*) iff $\zeta(M) \leq \zeta(M + \vec{\tau})$ (resp. $\zeta(M) \geq \zeta(M + \vec{\tau})$).

**Definition 4.3 (Bi-periodicity)** *A constraint $\phi$ is* bi-periodic *iff there exist two nonnull vectors $\vec{\mu}$ and $\vec{\nu}$ of positive integers of the form $(\mu_1, \mu_2, \mu_3)$ and $(\nu_1, \nu_2, \nu_3)$ such that:*



$\phi(M) = \phi(M + \vec{\tau}) \qquad \psi(M) = \psi(M + \vec{\tau})$

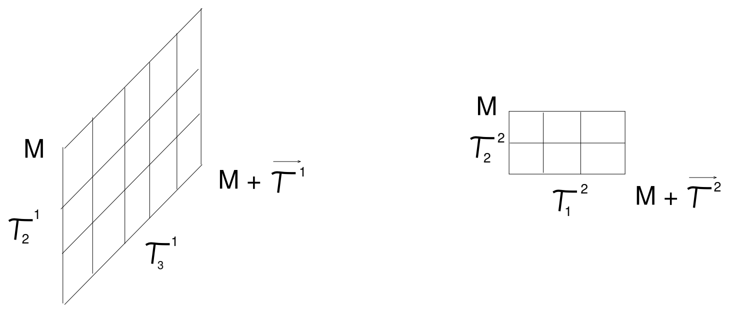
*Figure 4: Co-pattern for $\phi$ and $\psi$*

- $\exists i, j \in \{1, 2, 3\} \quad i \neq j \wedge \mu_i = \nu_j = 0$

  *and*

- $\forall M \quad \phi(M) = \phi(M + \vec{\mu}) = \phi(M + \vec{\nu})$.

*Given two indices $i, j$ satisfying the first relation, the* bi-period, *say $\{\vec{\tau^1}, \vec{\tau^2}\}$, of $\phi$ is the pair of smallest vectors satisfying the second double equation. For $l = 1, 2$, the* pattern *associated with $\vec{\tau^l}$ (also denoted $\vec{\tau^l}$) is the set of all the paths linking a given point $M$ to $M + \vec{\tau^l}$. See figure 5.*



$\phi(M) = \phi(M + \vec{\tau}^1) \qquad \phi(M) = \phi(M + \vec{\tau}^2)$

*Figure 5: Patterns for $\phi$*

Given a bi-period $\{\vec{\tau^1}, \vec{\tau^2}\}$ for a constraint $\phi$, we will say that a constraint $\psi$ is *bi-periodically strictly increasing* (resp. *bi-periodically strictly decreasing*) iff $\psi(M) < \psi(M + \vec{\tau^1}) \wedge \psi(M) < \psi(M + \vec{\tau^2})$ (resp. $\psi(M) > \psi(M + \vec{\tau^1}) \wedge \psi(M) > \psi(M + \vec{\tau^2})$).

## 5 Programs with Two Recursive Rules

In this section, we consider programs $\pi$ with two recursive rules and characterize the form of the paths of the subset $\Delta'(\sigma, \theta)$[1].

[1]The third coordinate $v$ is irrelevant in the case of two recursive rules

The method presented here differs substantially from the one in [11]. In [11], we used the notion of constraint implication (see section 7), which leads to a classification of programs of two recursive rules into 16 classes of programs. For 14 of these classes, the formulas generated to characterize the output of the bottom-up evaluation of $\pi$ of our method were quite simple. For the two other classes, a more sophisticated approach was required. The method we adopt here is similar to the one used in [11] for these two remaining classes. This method can be applied to any program of form $\pi$ and will be generalized to programs having three recursive rules in section 6. It is based on the concept of *periodic* constraint.

In the next two subsections, we exemplify the determination of $\Delta'(\sigma, \theta)$ for two programs $\pi_1$ and $\pi_2$. These programs can be seen as typical examples of programs belonging to the class of:

- Programs with one periodic constraint.
- Programs with two strictly monotonic constraints.

In the two subsequent subsections, we generalize this result for any program $\pi$ with 2 recursive rules.

## 5.1 Example of a Program with One Periodic Constraint

Consider the program $\pi_1$:

| | | | |
|---|---|---|---|
| $p(x, y, z)$ | :− | $\xi(x, y, z)$. | rule $R_0$ |
| $p(x-3, y-2, z-2)$ | :− | $x \geq 0,\ p(x, y, z)$ | rule $R_1$ |
| $p(x+5, y+3, z-1)$ | :− | $y \geq 0,\ p(x, y, z)$ | rule $R_2$ |

We have:

**Proposition 5.1 (Periodicity of $\phi_2$)**
$\forall\ \alpha,\ \forall \beta\ \phi_2(\alpha, \beta) = \phi_2(\alpha+3, \beta+2)$.

Proposition 5.1 means that there exists a pattern (3,2) for $\phi_2$ (see figure 6).

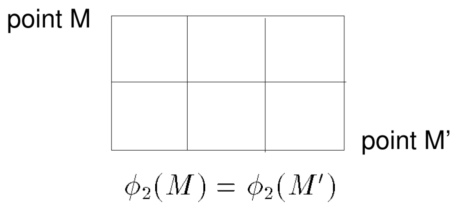
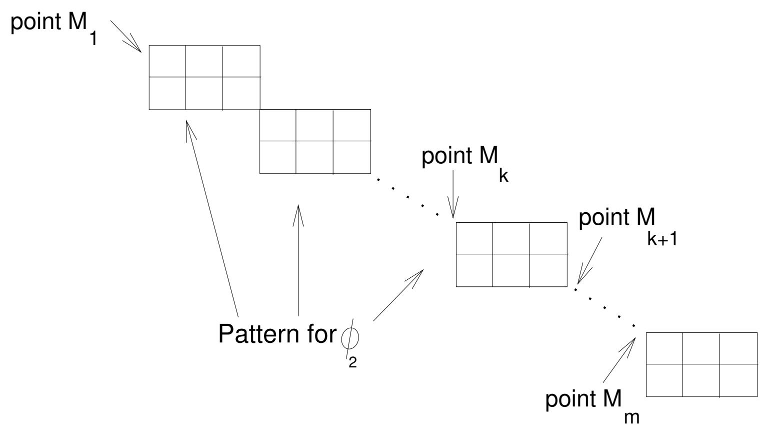


Figure 6: Pattern for $\phi_2$

The constraint $\phi_1$ is periodically increasing for the pattern (3,2):

**Proposition 5.1bis (Periodical increase of $\phi_1$)**
$\forall\ \alpha,\ \forall \beta\ \phi_1(\alpha, \beta) \leq \phi_1(\alpha+3, \beta+2)$.

It follows from propositions 5.1 and 5.1bis that one can compose the pattern (3,2) in an increasing way (w.r.t. $\phi_1$), as shown in figure 7.



$$\phi_2(M_1) \ = \ \ldots \ \phi_2(M_k) \ = \ \phi_2(M_{k+1}) \ = \ \ldots \ \phi_2(M_m)$$

$$\phi_1(M_1) \ \leq \ \ldots \ \phi_1(M_k) \ \leq \ \phi_1(M_{k+1}) \ \leq \ \ldots \ \phi_1(M_m)$$

Figure 7: Composing patterns for $\phi_2$

**Theorem 5.2** *For program $\pi_1$, the subset $\Delta'(\sigma, \theta)$ of characteristic paths is depicted in figure 8.*

## 5.2 Example of a Program with Two Strictly Monotonic Constraints

Consider the program $\pi_2$:

| | | | |
|---|---|---|---|
| $p(x, y, z)$ | :– | $\xi(x, y, z)$. | rule $R_0$ |
| $p(x+2, y-5, z-2)$ | :– | $x \geq 0,\ p(x, y, z)$ | rule $R_1$ |
| $p(x+3, y-1, z-1)$ | :– | $y \geq 0,\ p(x, y, z)$ | rule $R_2$ |

We have:

**Lemma 5.3 (Strictly monotonic constraints)** $\forall\ M,\ \forall\ \tau > \vec{0}$:

– $\phi_1(M) < \phi_1(M + \tau)$

– $\phi_2(M) > \phi_2(M + \tau)$

It follows that:

**Theorem 5.4** *For program $\pi_2$, the subset $\Delta'(\sigma, \theta)$ of characteristic paths is depicted in figure 9.*

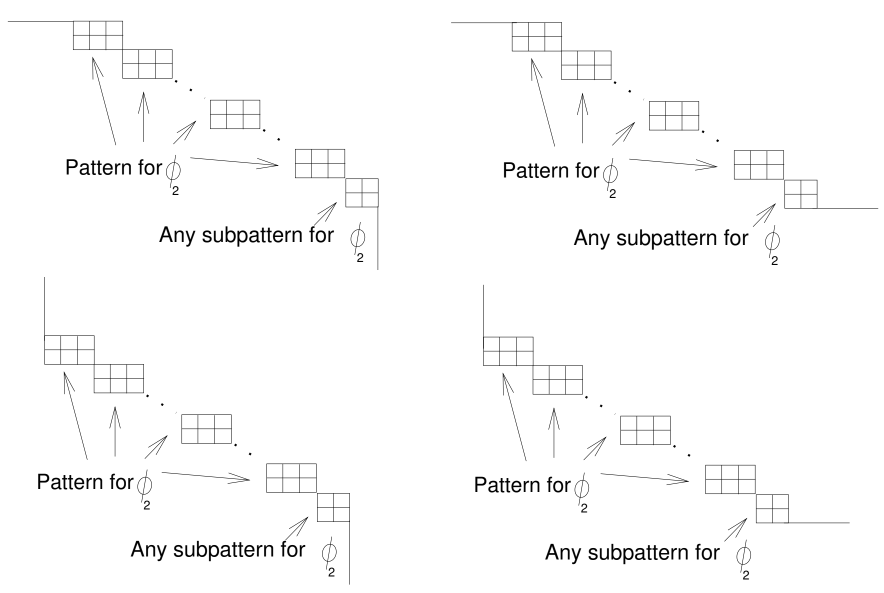

Pattern for $\phi_2$
Any subpattern for $\phi_2$
Pattern for $\phi_2$
Any subpattern for $\phi_2$
Pattern for $\phi_2$
Any subpattern for $\phi_2$
Pattern for $\phi_2$
Any subpattern for $\phi_2$

Figure 8:

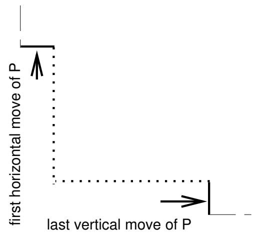


*Figure 9:*

## 5.3 Definition of the Classes

We now define the two different classes which appear in the case of programs with two recursive rules.

Consider a program $\pi$:

$$\begin{array}{lcll} p(x,y,z) & :- & \xi(x,y,z). & \text{rule } R_0 \\ p(x+a_1,y+b_1,z+c_1) & :- & x \geq 0,\ p(x,y,z) & \text{rule } R_1 \\ p(x+a_2,y+b_2,z+c_2) & :- & y \geq 0,\ p(x,y,z) & \text{rule } R_2 \end{array}$$

where $a_1,\ a_2,\ a_3,\ b_1,\ b_2,\ b_3,\ c_1,\ c_2,\ c_3\ \in \mathbb{Z}$ and $\xi(x,y,z)$ is an arithmetic formula.

For programs $\pi$, given a point $M$, the following situations can occur:

I. $-$ $\exists i \in \{1,2\},\ \exists\ \vec{\tau} > \vec{0},\ \forall\ M, \quad \phi_i(M) = \phi_i(M+\vec{\tau})$.

– For $j \neq i$:
Either $\forall M \phi_j(M) \geq \phi_i(M + \vec{\tau})$,
or $\forall M \phi_j(M) \leq \phi_i(M + \vec{\tau})$.

II. – $\forall i \in \{1, 2\}$:
Either $(\forall M,\ \forall\ \vec{\tau} > \vec{0}\ \phi_i(M) < \phi_i(M + \tau))$
or $(\forall M,\ \forall\ \vec{\tau} > \vec{0}\ \phi_i(M) > \phi_i(M + \tau)$.

Situation (I) means that $\phi_i$ is periodic of pattern $\vec{\tau}$ and that $\phi_j$ is either periodically increasing or periodically decreasing. Situation (II) means that $\phi_1$ and $\phi_2$ are strictly monotonic constraints. This provides a classification of programs $\pi$ into two classes of programs.

## 5.4 Algebraic Characterization of the classes

Consider a program of the form $\pi$. Consider now the equations below:

$$\begin{aligned} a_1\alpha + a_2\beta &= 0 \qquad\qquad \text{equation } (\star) \\ b_1\alpha + b_2\beta &= 0 \qquad\qquad \text{equation } (\star\star) \end{aligned}$$

Equation $(\star)$ states that the value of $x$ (and consequently the constraint $\phi_1$ of rule $R_1$) has not changed after $\alpha$ applications of $R_1$ and $\beta$ applications of $R_2$. Equation $(\star\star)$ states that the value of $y$ (and consequently the constraint $\phi_2$ of rule $R_2$) has not changed after $\alpha$ applications of $R_1$ and $\beta$ applications of $R_2$.

The program $\pi$ has a periodic constraint $\phi_1$ iff equation $(\star)$ admits a positive solution on $\alpha$ and $\beta$, i.e., iff $a_1.a_2 \leq 0$. A *pattern* $\vec{\tau^1}$ *for* $\phi_1$ is determined by a couple $(\tau_1^1, \tau_2^1)$, which is the smallest couple of positive integers solution of equation $(\star)$.

The program $\pi$ has a periodic constraint $\phi_2$ iff equation $(\star\star)$ admits a positive solution on $\alpha$ and $\beta$, i.e., iff $b_1.b_2 \leq 0$. A *pattern* $\vec{\tau^2}$ *for* $\phi_2$ is determined by a couple $(\tau_1^2, \tau_2^2)$, which is the smallest couple of positive integers solution of equation $(\star\star)$.

The program $\pi$ has two strictly monotonic constraints $\phi_1$ and $\phi_2$ iff neither $(\star)$ nor $(\star\star)$ admits a positive solution, i.e, iff $a_1.a_2 > 0$ and $b_1.b_2 > 0$.

This gives an algebraic characterization of the classes of programs with one periodic constraint or with two strictly monotonic constraints.

## 5.5 General Results

In this subsection, we generalize theorem 5.2 for any program with one periodic constraint and theorem 5.4 for programs with two strictly monotonic constraints.

**Theorem 5.5 (Programs with one periodic constraint $\phi_i$ )** *Given a program $\pi$ of class I, the subset $\Delta'(\sigma, \theta)$ of characteristic paths is depicted in figure 10.*

There are two different subclasses depending on whether $\phi_j$ $(j \neq i)$ is periodically increasing or periodically decreasing. This monotonicity property of $\phi_j$ is useful for the second step of $\bigwedge$-simplification of $\Gamma(\sigma, \theta)$ (see section 3).

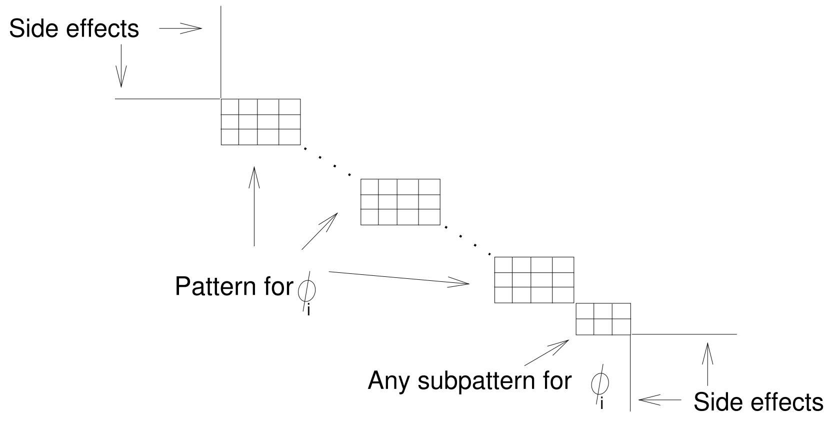


Figure 10:

**Theorem 5.6 (Programs with two strictly monotonic constraints)** *Given a program $\pi$ of class II, the subset $\Delta'(\sigma, \theta)$ of characteristic paths is depicted in figure 11.*

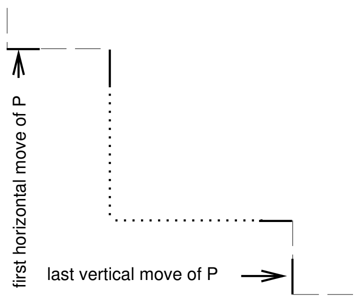
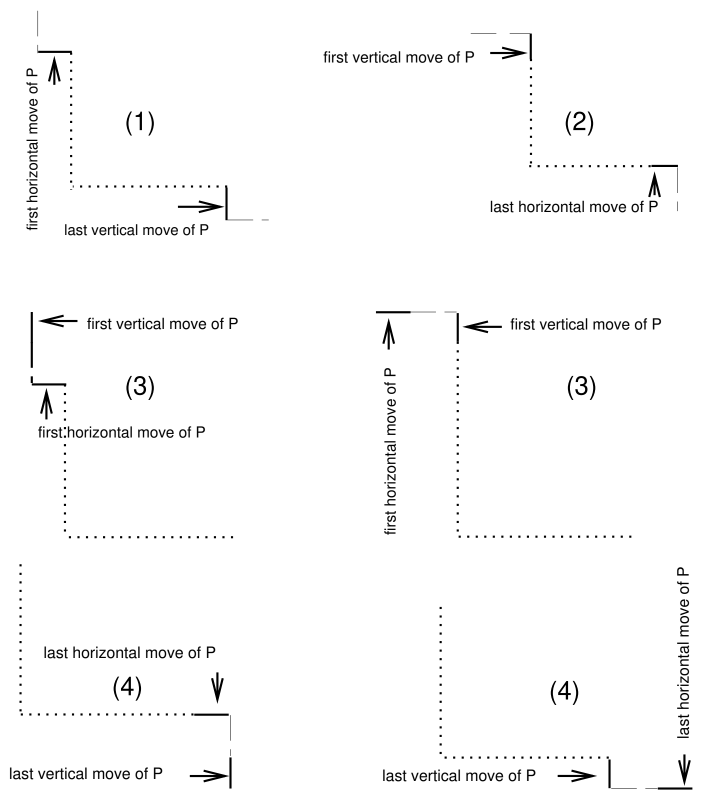


*Figure 11:*

For class II, we can simplify the form of the paths of figure 11 according to the signs of $a_i$ and $b_i$, as shown in figure 12.

We can algebraically characterize the classes of programs with two recursive rules as shown in table 5.5.

| | | | |
|---|---|---|---|
| $a_1 \geq 0,\ a_2 \leq 0$ | Class I | pattern for $\phi_1$ | $\phi_2$ periodically monotonic |
| $a_1 \leq 0,\ a_2 \geq 0$ | Class I | pattern for $\phi_1$ | $\phi_2$ periodically monotonic |
| $b_1 \geq 0,\ b_2 \leq 0$ | Class I | pattern for $\phi_2$ | $\phi_1$ periodically monotonic |
| $b_1 \leq 0,\ b_2 \geq 0$ | Class I | pattern for $\phi_2$ | $\phi_1$ periodically monotonic |
| $a_1 > 0,\ a_2 > 0,\ b_1 > 0, b_2 > 0$ | Class II | $\phi_1$ strictly increasing | $\phi_2$ strictly increasing |
| $a_1 < 0,\ a_2 < 0,\ b_1 < 0,\ b_2 < 0$ | Class II | $\phi_1$ strictly decreasing | $\phi_2$ strictly decreasing |
| $a_1 > 0,\ a_2 > 0,\ b_1 < 0,\ b_2 < 0$ | Class II | $\phi_1$ strictly increasing | $\phi_2$ strictly decreasing |
| $a_1 < 0,\ a_2 < 0,\ b_1 > 0, b_2 > 0$ | Class II | $\phi_1$ strictly decreasing | $\phi_2$ strictly increasing |



(1) $a_i < 0$ and $b_i > 0$ (2) $a_i > 0$ and $b_i < 0$
(3) $a_i > 0$ and $b_i > 0$ (4) $a_i < 0$ and $b_i < 0$

Figure 12:

## 6 Programs with Three Recursive Rules

In this section, we extend the results of the previous section for programs with three recursive rules. We follow the same aim as before, i.e. , to determine a subset $\Delta'(\sigma, \theta, v)$ of characteristic paths that subsume all the paths of the cube $\Delta(\sigma, \theta, v)$.

For the case of three recursive rules, we have 3 classes of programs. In the next three subsections we exemplify the determination of $\Delta'(\sigma, \theta, v)$ for three programs $\pi_3$, $\pi_4$ and $\pi_5$. These programs can be seen as typical examples of programs belonging to the classes of:

I. Programs with two co-periodic constraints.

II. Programs with one bi-periodic constraint.

III. Programs with three strictly monotonic constraints.

As in section 5, we generalize our results for any program of form $\pi$ in the subsequent subsections.

## 6.1 Example of a Program with Two Co-periodic Constraints

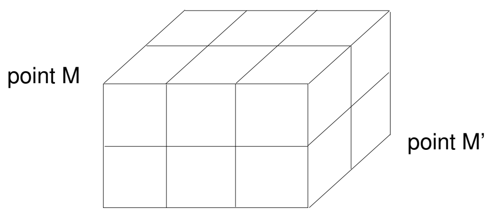
Consider the program $\pi_3$:

$$\begin{array}{lcll}
p(x,y,z) & :- & \xi(x,y,z). & \text{rule } R_0 \\
p(x+2,y-4,z-2) & :- & x \geq 0,\ p(x,y,z) & \text{rule } R_1 \\
p(x-5,y+9,z-1) & :- & y \geq 0,\ p(x,y,z) & \text{rule } R_2 \\
p(x+2,y-3,z+1) & :- & z \geq 0,\ p(x,y,z) & \text{rule } R_3
\end{array}$$

We have:

**Proposition 6.1 (Co-periodicity of $\phi_1$ and $\phi_2$)**
$\forall\alpha, \forall\beta, \forall\gamma \;\; \phi_1(\alpha,\beta,\gamma) = \phi_1(\alpha+3,\beta+2,\gamma+2) \;\wedge \phi_2(\alpha,\beta,\gamma) = \phi_2(\alpha+3,\beta+2,\gamma+2)$.

Proposition 6.1 means that there exists a same pattern (3,2,2) for the constraints $\phi_1$ and $\phi_2$. (see figure 13).



$\phi_1(M) = \phi_1(M')$ $\qquad$ $\phi_2(M) = \phi_2(M')$

Figure 13: Co-pattern for $\phi_1$ and $\phi_2$

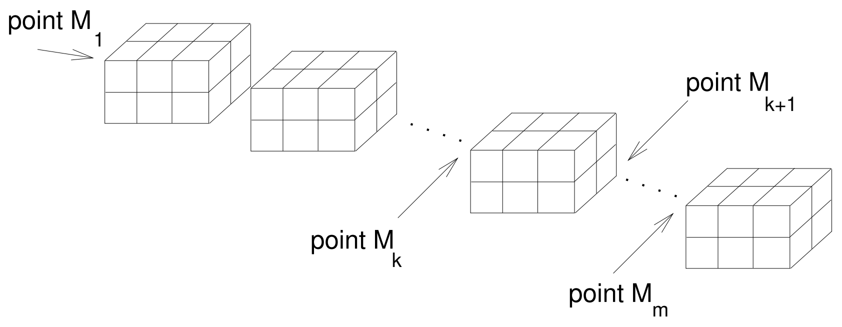
The constraint $\phi_3$ is co-periodically increasing:

**Proposition 6.1bis (Co-periodical increase of $\phi_3$)**
$\forall\alpha, \forall\beta, \forall\gamma \;\; \phi_3(\alpha,\beta,\gamma) \leq \phi_3(\alpha+3,\beta+2,\gamma+2)$

It follows from propositions 6.1 and 6.1bis that one can compose the pattern (3,2,2) in an increasing way (w.r.t. $\phi_3$), as shown in figure 14.

**Theorem 6.2** *For program $\pi_3$, the subset $\Delta'(\sigma,\theta,\upsilon)$ of characteristic paths is depicted in figure 15.*

There are nine possible combinations of planar moves before and after the composition of the co-pattern.



$$\phi_1(M_1) \;=\; \dots\; \phi_1(M_k) \;=\; \phi_1(M_{k+1}) \;=\; \dots\; \phi_1(M_m)$$
$$\phi_2(M_1) \;=\; \dots\; \phi_2(M_k) \;=\; \phi_2(M_{k+1}) \;=\; \dots\; \phi_2(M_m)$$
$$\phi_3(M_1) \;\leq\; \dots\; \phi_3(M_k) \;\leq\; \phi_3(M_{k+1}) \;\leq\; \dots\; \phi_3(M_m)$$

Figure 14: Composing co-patterns for $\phi_1$ and $\phi_2$

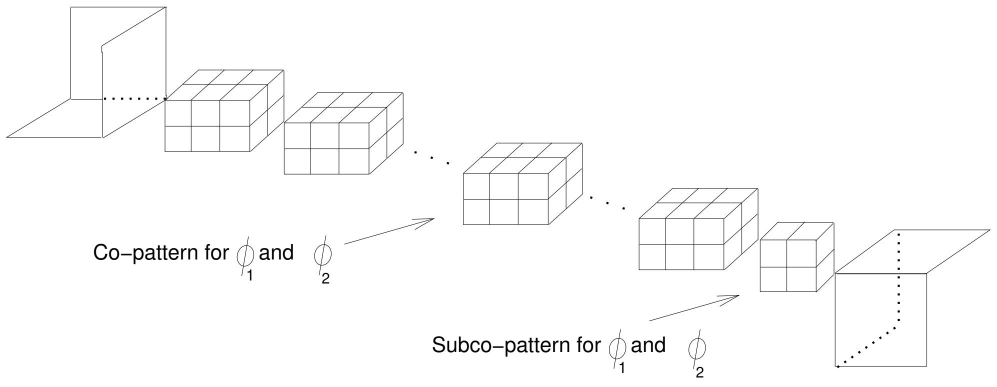

Co−pattern for $\phi_1$ and $\phi_2$
Subco−pattern for $\phi_1$ and $\phi_2$

Figure 15:

## 6.2 Example of a Program with One Bi-Periodic Constraint

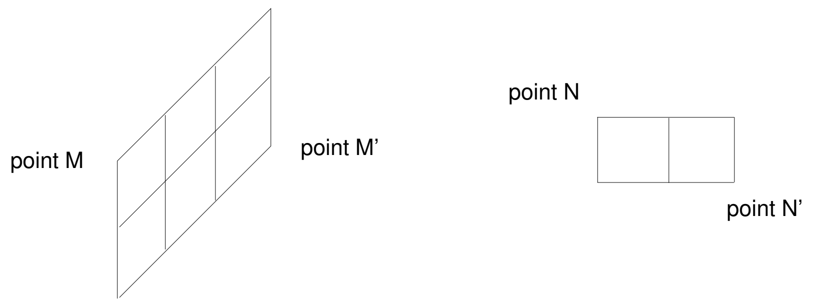
Consider the program $\pi_4$:

| | | | |
|---|---|---|---|
| $p(x,y,z)$ | :– | $\xi(x,y,z)$. | rule $R_0$ |
| $p(x+1,y-3,z-4)$ | :– | $x \geq 0,\ p(x,y,z)$ | rule $R_1$ |
| $p(x-1,y+6,z-3)$ | :– | $y \geq 0,\ p(x,y,z)$ | rule $R_2$ |
| $p(x+2,y-4,z+1)$ | :– | $z \geq 0,\ p(x,y,z)$ | rule $R_3$ |

We have:

**Proposition 6.3 (Bi-periodicity of $\phi_2$)**
$\forall\alpha, \forall\beta, \forall\gamma \quad \phi_2(\alpha,\beta,\gamma) = \phi_2(\alpha,\beta+2,\gamma+3) \ = \phi_2(\alpha+2,\beta+1,\gamma)$.

Proposition 6.3 means that there exists two different patterns (0,2,3) and (2,1,0) for constraint $\phi_2$, which are located in different planes (see figure 16).



$\phi_2(M) = \phi_2(M')$ and $\phi_2(N) = \phi_2(N')$

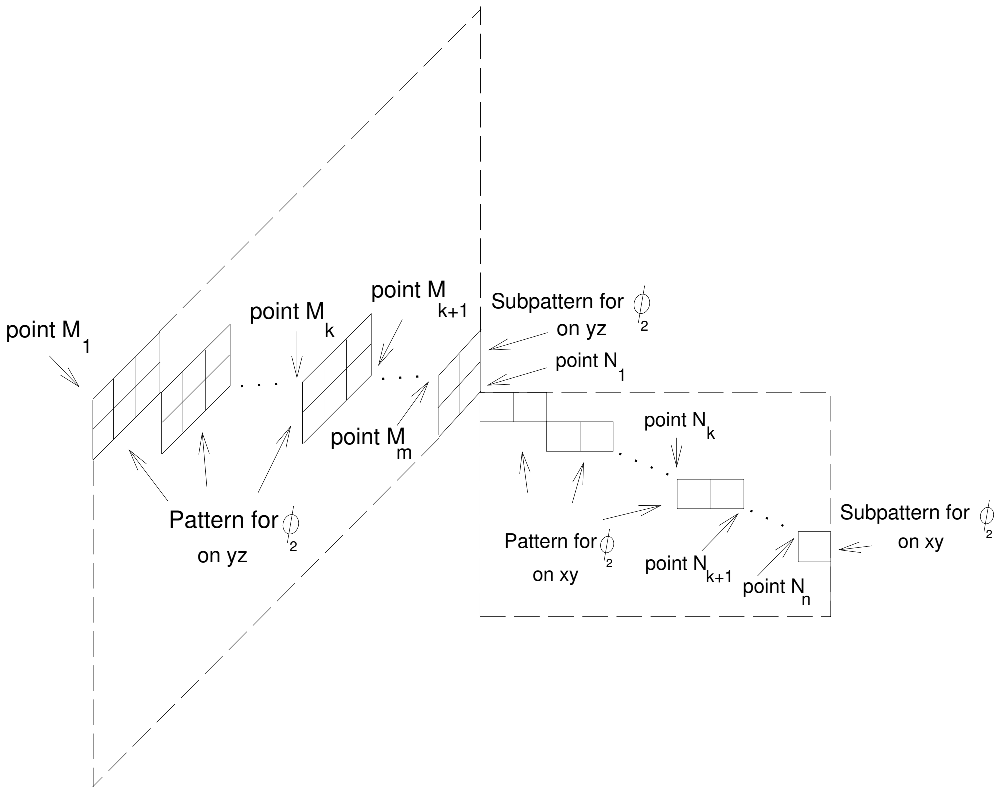
Figure 16: Bi-periodicity of $\phi_2$

The constraint $\phi_1$ (resp. $\phi_3$) is bi-periodically strictly increasing (resp. decreasing):

**Proposition 6.3bis ( Bi-periodical strict increase of $\phi_1$ and decrease of $\phi_3$)**

- $\forall\alpha,\ \forall\beta,\ \forall\gamma\ \ (\ \phi_1(\alpha,\beta,\gamma) < \phi_1(\alpha+2,\beta+1,\gamma)\ \wedge\ \phi_1(\alpha,\beta,\gamma) < \phi_1(\alpha,\beta+2,\gamma+3)\ )$
  *and*
- $\forall\alpha,\ \forall\beta,\ \forall\gamma\ \ (\ \phi_3(\alpha,\beta,\gamma) > \phi_3(\alpha+2,\beta+1,\gamma)\ \wedge\ \phi_3(\alpha,\beta,\gamma) > \phi_3(\alpha,\beta+2,\gamma+3)\ )$.

It follows from propositions 6.3 and 6.3bis that one can compose first the pattern (0,2,3), and then pattern (2,1,0) in a globally monotonic way (w.r.t $\phi_1$ and $\phi_3$), as shown in figure 17.


point $M_1$
point $M_k$
point $M_{k+1}$
Subpattern for $\phi_2$ on yz
point $N_1$
point $M_m$
point $N_k$
Pattern for $\phi_2$ on yz
Pattern for $\phi_2$ on xy
point $N_{k+1}$
point $N_n$
Subpattern for $\phi_2$ on xy

$$\phi_2(M_1) = \ldots \phi_2(M_k) = \phi_2(M_{k+1}) = \ldots \phi_2(M_m) \leq \phi_2(N_1) = \ldots \phi_2(N_k) = \phi_2(N_{k+1}) = \ldots \phi_2(N_n)$$

$$\phi_1(M_1) < \ldots \phi_1(M_k) < \ldots \phi_1(M_{k+1}) < \phi_1(M_m) \leq \phi_1(N_1) < \ldots \phi_1(N_k) < \ldots \phi_1(N_{k+1}) < \phi_1(N_n)$$

$$\phi_3(M_1) > \ldots \phi_3(M_k) > \phi_3(M_{k+1}) > \ldots \phi_3(M_m) \geq \phi_3(N_1) > \ldots \phi_3(N_k) > \phi_3(N_{k+1}) > \ldots \phi_3(N_n)$$

Figure 17: Bi-periodicity of $\phi_2$

**Theorem 6.4** *For program $\pi_4$, the subset $\Delta'(\sigma, \theta, \upsilon)$ of characteristic paths is depicted in figures 18 and 19 .*

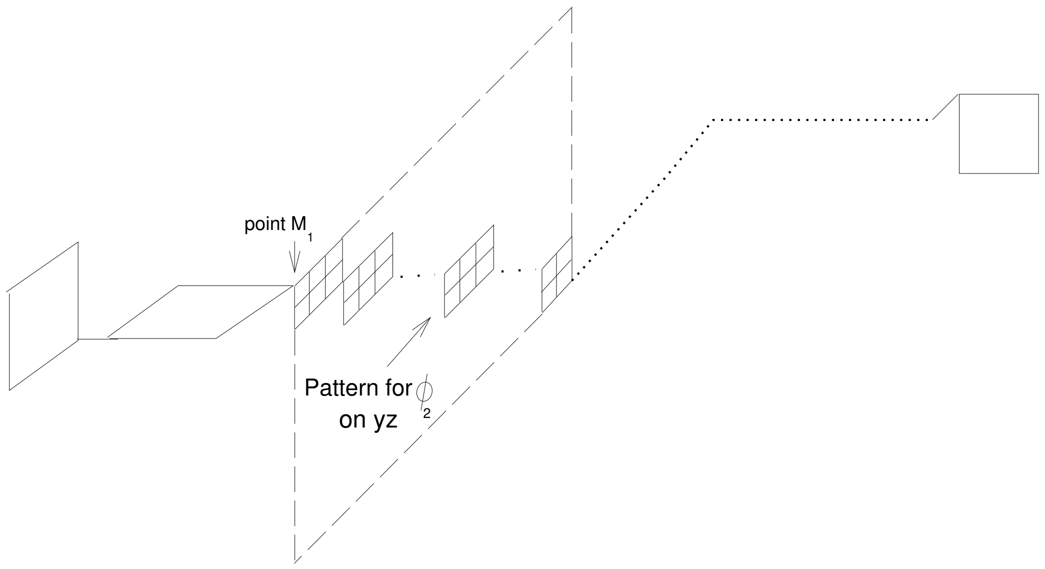


*Figure 18: Relative shortage of y-moves w.r.t. z-moves*

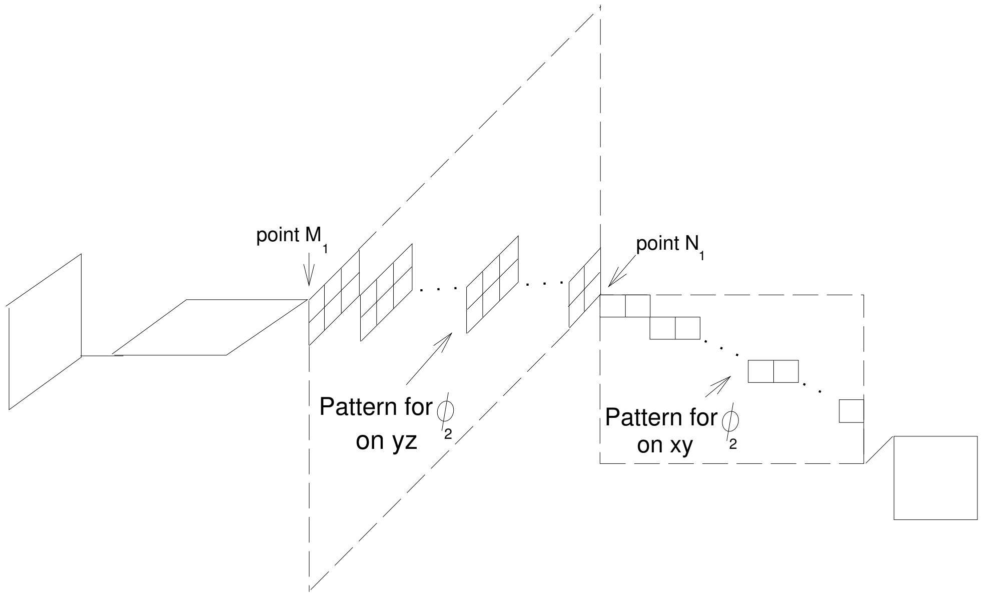


*Figure 19: Relative shortage of z-moves w.r.t. y-moves.*

## 6.3 Example of a Program with Three Strictly Monotonic Constraints

Consider the program $\pi_5$:

$$
\begin{array}{llll}
p(x,y,z) & :- & \xi(x,y,z). & \text{rule } R_0 \\
p(x+2,y-5,z-2) & :- & x \geq 0,\ p(x,y,z) & \text{rule } R_1 \\
p(x+3,y-1,z-1) & :- & y \geq 0,\ p(x,y,z) & \text{rule } R_2 \\
p(x+5,y-1,z-3) & :- & z \geq 0,\ p(x,y,z) & \text{rule } R_3
\end{array}
$$

We have:

**Proposition 6.5 (Strictly monotonic constraints)** *$\forall\ M$, $\forall\ M' > M$:*

- $\phi_1(M) < \phi_1(M')$
- $\phi_2(M) > \phi_2(M')$
- $\phi_3(M) > \phi_3(M')$

It follows that:

**Theorem 6.6** *For program $\pi_5$, the subset $\Delta'(\sigma, \theta, \upsilon)$ of characteristic paths is depicted in figure 20.*

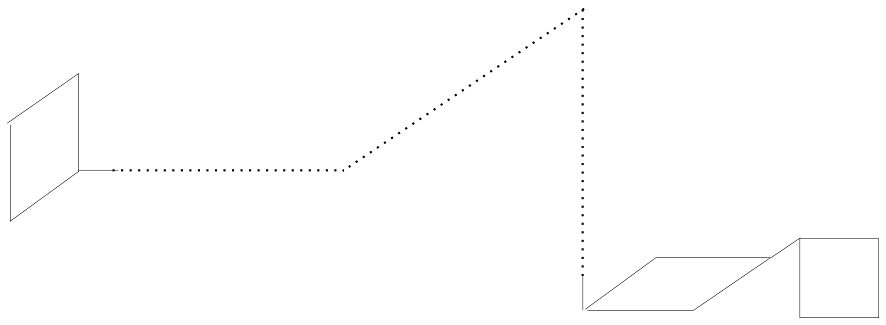


*Figure 20:*

## 6.4 Definition of the Classes

We now define the three different classes which appear in the case of programs with three recursive rules.

Consider a program $\pi$:

$$
\begin{array}{llll}
p(x,y,z) & :- & \xi(x,y,z). & \text{rule } R_0 \\
p(x+a_1,y+b_1,z+c_1) & :- & x \geq 0,\ p(x,y,z) & \text{rule } R_1 \\
p(x+a_2,y+b_2,z+c_2) & :- & y \geq 0,\ p(x,y,z) & \text{rule } R_2 \\
p(x+a_3,y+b_3,z+c_3) & :- & z \geq 0,\ p(x,y,z) & \text{rule } R_3
\end{array}
$$

where $a_1,\ a_2,\ a_3,\ b_1,\ b_2,\ b_3,\ c_1,\ c_2,\ c_3\ \in \mathbb{Z}$ and $\xi(x,y,z)$ is an arithmetic formula.

For programs of form $\pi$, given a point $M$, the following situations can occur:

I. $\exists i,j \in \{1,2,3\},\ \exists \vec{\tau} > \vec{0}$ :

- $\forall\ M \quad \phi_i(M) = \phi_i(M+\vec{\tau}) \ \wedge\ \phi_j(M) = \phi_j(M+\vec{\tau})$.
- For $k \neq i$ and $k \neq j$:
  Either $\forall M \quad \phi_k(M) \leq \phi_k(M+\vec{\tau})$,
  or $\forall M \quad \phi_k(M) \geq \phi_k(M+\vec{\tau})$.

II. $\exists i \in \{1,2,3\},\ \exists \vec{\tau} > \vec{0},\ \exists \vec{\nu} > \vec{0}$ with $\tau_l = 0,\ \nu_j = 0$ and $l \neq j$, such that:

- $\forall M \quad \phi_i(M) = \phi_i(M+\vec{\tau}) = \phi_i(M+\vec{\nu})$.
- $\forall\ k\ \in \{1,2,3\},\ k \neq i$:
  Either $\forall M \quad (\ \phi_k(M) < \phi_k(M+\vec{\tau}) \ \wedge\ \phi_k(M) < \phi_k(M+\vec{\nu})\ )$,
  or $\forall M \quad (\ \phi_k(M) > \phi_k(M+\vec{\tau}) \ \wedge\ \phi_k(M) > \phi_k(M+\vec{\nu})\ )$.

III. $\forall i \in \{1,2,3\}$:
  Either $\forall M,\ \forall\ \vec{\tau} > \vec{0} \quad \phi_i(M) < \phi_i(M+\vec{\tau})$,
  or $\forall M,\ \forall\ \vec{\tau} > \vec{0} \quad \phi_i(M) > \phi_i(M+\vec{\tau})$.

Situation (I) means that $\phi_i$ and $\phi_j$ are co-periodic of pattern $\vec{\tau}$. The constraint $\phi_k$ is either co-periodically increasing or co-periodically decreasing. Programs in this situation are called co-periodic programs.

Situation (II) means that $\phi_i$ is bi-periodic of patterns $\{\vec{\tau}, \vec{\nu}\}$. The others constraints are either bi-periodically strictly increasing or bi-periodically strictly decreasing. Programs in this situation are called bi-periodic programs.

Situation (III) means that $\phi_1$, $\phi_2$ and $\phi_3$ are strictly monotonic constraints. Programs in this situation are called strictly monotonic programs.

This provides a classification of programs $\pi$ into three classes of programs.

## 6.5 Algebraic Characterization of the classes

Consider the program $\pi$:

| | | | |
|---|---|---|---|
| $p(x,y,z)$ | :– | $\xi(x,y,z)$. | rule $R_0$ |
| $p(x+a_1, y+b_1, z+c_1)$ | :– | $x \geq 0,\ p(x,y,z)$ | rule $R_1$ |
| $p(x+a_2, y+b_2, z+c_2)$ | :– | $y \geq 0,\ p(x,y,z)$ | rule $R_2$ |
| $p(x+a_3, y+b_3, z+c_3)$ | :– | $z \geq 0,\ p(x,y,z)$ | rule $R_3$ |

Let $\alpha$, $\beta$ and $\gamma$ be respectively the number of applications of rules $R_1$, $R_2$ and $R_3$ in the bottom-up evaluation of $\pi$ and consider the equations below:

$$a_1\alpha + a_2\beta + a_3\gamma \ = \ 0 \qquad \text{equation } (\star)$$
$$b_1\alpha + b_2\beta + b_3\gamma \ = \ 0 \qquad \text{equation } (\star\star)$$
$$c_1\alpha + c_2\beta + c_3\gamma \ = \ 0 \qquad \text{equation } (\star\star\star)$$

Equation $(\star)$ states that the value of $x$ (and consequently the constraint of rule $R_1$) has not changed after $\alpha$ applications of $R_1$, $\beta$ applications of $R_2$ and $\gamma$ applications of $R_3$. Equation $(\star\star)$ states that the value of $y$ has not changed and equation $(\star\star\star)$ states that the value of $z$ has not changed.

Let $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ be the systems:

$$\Sigma_1 \quad \begin{cases} a_1\alpha + a_2\beta + a_3\gamma = 0 & \text{equation } (\star) \\ b_1\alpha + b_2\beta + b_3\gamma = 0 & \text{equation } (\star\star) \end{cases}$$

$$\Sigma_2 \quad \begin{cases} a_1\alpha + a_2\beta + a_3\gamma = 0 & \text{equation } (\star) \\ c_1\alpha + c_2\beta + c_3\gamma = 0 & \text{equation } (\star\star\star) \end{cases}$$

$$\Sigma_3 \quad \begin{cases} b_1\alpha + b_2\beta + b_3\gamma = 0 & \text{equation } (\star\star) \\ c_1\alpha + c_2\beta + c_3\gamma = 0 & \text{equation } (\star\star\star) \end{cases}$$

We distinguish the following situations:

- There exists a nontrivial solution $(\alpha, \beta, \gamma)$ for system $\Sigma_1$ or $\Sigma_2$ or $\Sigma_3$, with $\alpha \geq 0$, $\beta \geq 0$ and $\gamma \geq 0$.
- None of the systems $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ admits a nontrivial solution $(\alpha, \beta, \gamma)$ with $\alpha \geq 0$, $\beta \geq 0$ and $\gamma \geq 0$, but one of the equations $(\star)$, $(\star\star)$ and $(\star\star\star)$ admits a positive solution.
- None of the equations $(\star)$, $(\star\star)$ and $(\star\star\star)$ admits a positive solution.

It is clear that the first situation corresponds to the class (I) described in the previous subsection. For programs in this class, there exists a co-pattern for two constraints (see example in subsection 6.1). More precisely, $\Sigma_1$ admits a positive solution iff we have a co-pattern for $\phi_1$ and $\phi_2$; $\Sigma_2$ admits a positive solution iff we have a co-pattern for $\phi_1$ and $\phi_3$; $\Sigma_3$ admits a positive solution iff we have a co-pattern for $\phi_2$ and $\phi_3$.

It can be proved that the second situation corresponds to the class (II) described in the previous subsection. For programs in this class, there exits two patterns for one constraint on two different planes (see example in section 6.2).

It is also clear that the last situation corresponds to the class (III) described in the previous subsection. For programs in this class the constraints are strictly monotonic (see example in section 6.3).

Hence, we have an algebraic characterization of each class of programs with three recursive rules.

## 6.6 General Results

In this subsection, we generalize the results obtained for program $\pi_3$, $\pi_4$ and $\pi_5$.

**Theorem 6.7 (Co-periodic Programs)** *Given a program in class (I) (with co-pattern for $\phi_i$ and $\phi_j$), the subset $\Delta'(\sigma, \theta, v)$ of characteristic paths is depicted in figure 21.*

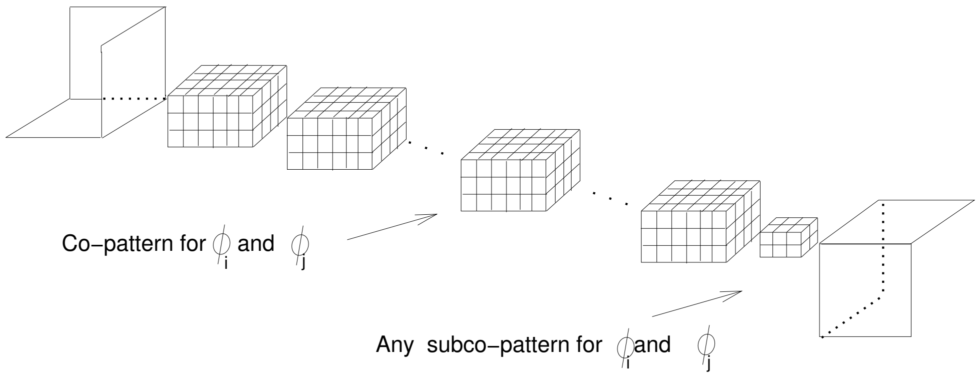

Co–pattern for $\phi_i$ and $\phi_j$
Any subco–pattern for $\phi_i$ and $\phi_j$

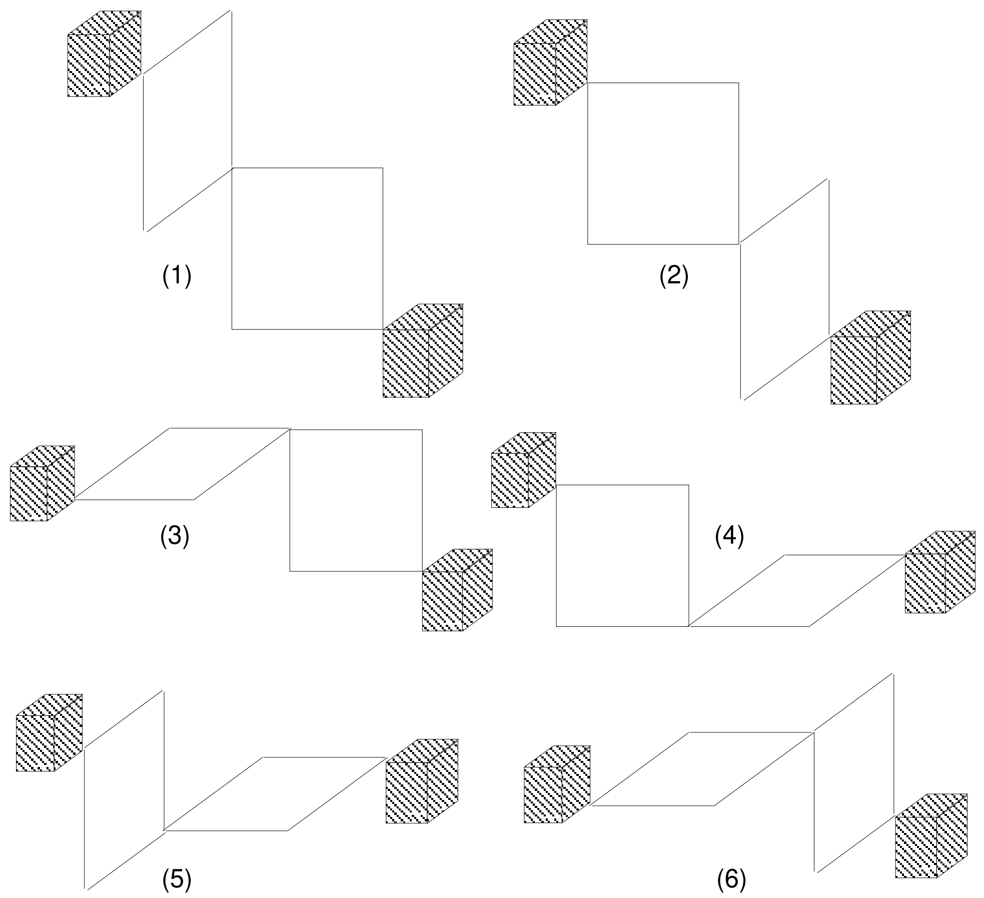
*Figure 21:*

As in theorem 6.2, there are nine possible combinations of planar moves before and after the composition of the co-pattern to be considered.

There are two subclasses of programs depending on whether $\phi_k$ $(k \neq i, j)$ is co-periodically increasing or decreasing. This monotonicity property is useful for the second step of $\bigwedge$-simplification of we reduce $\Gamma(\sigma, \theta, v)$ (see section 3).

For class II, we have:

**Theorem 6.8 (Bi-periodic Programs)** *Given a program in class (II), the subset $\Delta'(\sigma, \theta, v)$ of characteristic paths is depicted in figure 22.*



(1) *Bi-periodicity of* $\phi_i$ *with* $\tau_1^1 = 0$ *and* $\tau_3^2 = 0$
(2) *Bi-periodicity of* $\phi_i$ *with* $\tau_3^1 = 0$ *and* $\tau_1^2 = 0$
(3) *Bi-periodicity of* $\phi_i$ *with* $\tau_2^1 = 0$ *and* $\tau_3^2 = 0$
(4) *Bi-periodicity of* $\phi_i$ *with* $\tau_3^1 = 0$ *and* $\tau_2^2 = 0$
(5) *Bi-periodicity of* $\phi_i$ *with* $\tau_1^1 = 0$ *and* $\tau_2^2 = 0$
(6) *Bi-periodicity of* $\phi_i$ *with* $\tau_2^1 = 0$ *and* $\tau_1^2 = 0$

*Figure 22:*

The grey boxes at the beginning and at the end this paths represent a certain set of paths contained in a bounded number of planes. The size and form of these boxes are determined by the values of the coefficients $a_i$, $b_i$ and $c_i$ (being therefore bounded).

For class III, we have:

**Theorem 6.9 (Strictly Monotonic Programs)** *Given a program in class III, the subset* $\Delta'(\sigma, \theta, \upsilon)$ *of characteristic paths is depicted in figure 23.*

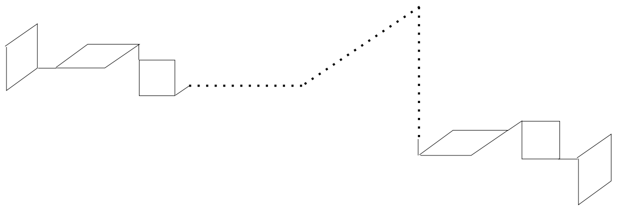
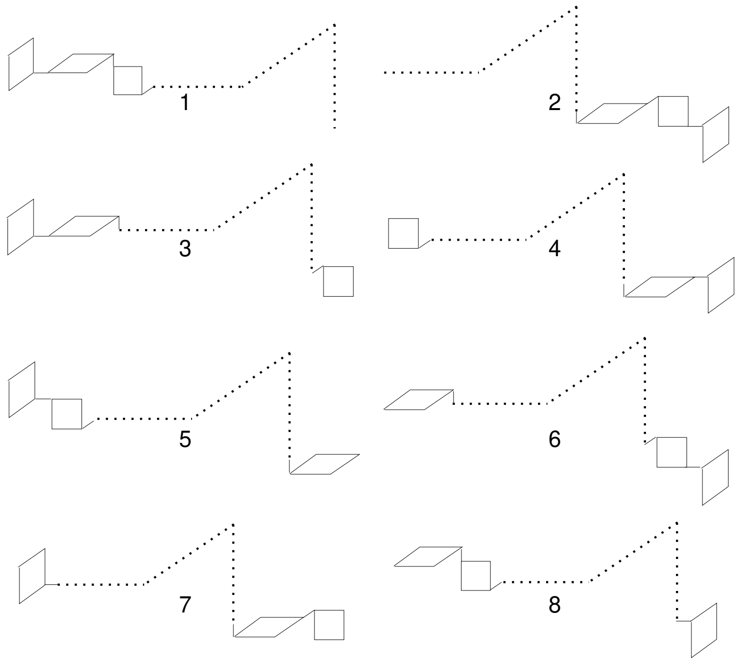


*Figure 23:*

We can simplify the form of the paths of figure 23, according to the signs of $a_i$, $b_i$ and $c_i$, as shown in figure 24.



| | | | |
|---|---|---|---|
| (1) | $a_i > 0,\ b_i > 0,\ c_i > 0$ | (2) | $a_i < 0,\ b_i < 0,\ c_i < 0$ |
| (3) | $a_i > 0,\ b_i > 0,\ c_i < 0$ | (4) | $a_i < 0,\ b_i < 0,\ c_i > 0$ |
| (5) | $a_i > 0,\ b_i < 0,\ c_i < 0$ | (6) | $a_i < 0,\ b_i > 0,\ c_i > 0$ |
| (7) | $a_i > 0,\ b_i < 0,\ c_i > 0$ | (8) | $a_i < 0,\ b_i > 0,\ c_i < 0$ |

Figure 24:

We can algebraically characterize the classes of programs of three recursive rules as shown in tables 6.6, 6.6 and 6.6.

Let us denote by:

$$\text{Inequations 1}\left\{\begin{array}{lll}\dfrac{a_2b_3-a_3b_2}{a_1b_2-a_2b_1}\geq 0 & \dfrac{a_3b_1-a_1b_3}{a_1b_2-a_2b_1}\geq 0 & \dfrac{a_2b_3-a_3b_2}{a_3b_1-a_1b_3}\geq 0\\[2ex] \dfrac{a_1b_2-a_2b_1}{a_2b_3-a_3b_2}\geq 0 & \dfrac{a_1b_2-a_2b_1}{a_3b_1-a_1b_3}\geq 0 & \dfrac{a_3b_1-a_1b_3}{a_2b_3-a_3b_2}\geq 0\end{array}\right.$$

$$\text{Inequations 2}\left\{\begin{array}{lll}\dfrac{a_2c_3-a_3c_2}{a_1c_2-a_2c_1}\geq 0 & \dfrac{a_3c_1-a_1c_3}{a_1c_2-a_2c_1}\geq 0 & \dfrac{a_3c_1-a_1c_3}{a_2c_3-a_3c_2}\geq 0\\[2ex] \dfrac{a_1c_2-a_2c_1}{a_2c_3-a_3c_2}\geq 0 & \dfrac{a_1c_2-a_2c_1}{a_3c_1-a_1c_3}\geq 0 & \dfrac{a_2c_3-a_3c_2}{a_3c_1-a_1c_3}\geq 0\end{array}\right.$$

$$\text{Inequations 3}\left\{\begin{array}{lll}\dfrac{b_2c_3-b_3c_2}{b_1c_2-b_2c_1}\geq 0 & \dfrac{b_3c_1-b_1c_3}{b_1c_2-b_2c_1}\geq 0 & \dfrac{b_3c_1-b_1c_3}{b_2c_3-b_3c_2}\geq 0\\[2ex] \dfrac{b_1c_2-b_2c_1}{b_2c_3-b_3c_2}\geq 0 & \dfrac{b_1c_2-b_2c_1}{b_3c_1-b_1c_3}\geq 0 & \dfrac{b_3c_1-b_1c_3}{b_2c_3-b_3c_2}\geq 0\end{array}\right.$$

For the class of programs I, we have:

| | |
|---|---|
| Inequations 1 hold | Co-periodic constraint $\phi_1$ and $\phi_2$ |
| Inequations 2 hold | Co-periodic constraint $\phi_1$ and $\phi_3$ |
| Inequations 3 hold | Co-periodic constraint $\phi_2$ and $\phi_3$ |

For the class of programs II, when equation ($\star$) admits a positive solution (analogous for equations ($\star\star$) and ($\star\star\star$)), we have:

| | |
|---|---|
| $a_1 > 0,\ a_2 \leq 0,\ a_3 \leq 0,\ \dfrac{a_3b_1-a_1b_3}{a_1b_2-a_2b_1} < 0,\ \dfrac{a_3c_1-a_1c_3}{a_1c_2-a_2c_1} < 0$ | bi-periodicity of $\phi_1$ with $\tau_2 = 0$ and $\nu_3 = 0$ |
| $a_1 < 0,\ a_2 \geq 0,\ a_3 \geq 0,\ \dfrac{a_3b_1-a_1b_3}{a_1b_2-a_2b_1} < 0,\ \dfrac{a_3c_1-a_1c_3}{a_1c_2-a_2c_1} < 0$ | bi-periodicity of $\phi_1$ with $\tau_2 = 0$ and $\nu_3 = 0$ |
| $a_1 \leq 0,\ a_2 > 0,\ a_3 \leq 0,\ \dfrac{a_2b_3-a_3b_2}{a_1b_2-a_2b_1} < 0,\ \dfrac{a_2c_3-a_3c_2}{a_1c_2-a_2c_1} < 0$ | bi-periodicity of $\phi_1$ with $\tau_1 = 0$ and $\nu_3 = 0$ |
| $a_1 \geq 0,\ a_2 < 0,\ a_3 \geq 0,\ \dfrac{a_2b_3-a_3b_2}{a_1b_2-a_2b_1} < 0,\ \dfrac{a_2c_3-a_3c_2}{a_1c_2-a_2c_1} < 0$ | bi-periodicity of $\phi_1$ with $\tau_1 = 0$ and $\nu_3 = 0$ |
| $a_1 \leq 0,\ a_2 \leq 0,\ a_3 > 0,\ \dfrac{a_2b_3-a_3b_2}{a_3b_1-a_1b_3} < 0,\ \dfrac{a_2c_3-a_3c_2}{a_3c_1-a_1c_3} < 0$ | bi-periodicity of $\phi_1$ with $\tau_1 = 0$ and $\nu_2 = 0$ |
| $a_1 \geq 0,\ a_2 \geq 0,\ a_3 < 0,\ \dfrac{a_2b_3-a_3b_2}{a_3b_1-a_1b_3} < 0,\ \dfrac{a_2c_3-a_3c_2}{a_3c_1-a_1c_3} < 0$ | bi-periodicity of $\phi_1$ with $\tau_1 = 0$ and $\nu_2 = 0$ |

For the classes of programs III, we have:

| | | | |
|---|---|---|---|
| $a_i > 0,\ b_i > 0,\ c_i > 0$ | $\phi_1$ strictly increasing | $\phi_2$ strictly increasing | $\phi_3$ strictly increasing |
| $a_i < 0,\ b_i < 0,\ c_i < 0$ | $\phi_1$ strictly decreasing | $\phi_2$ strictly decreasing | $\phi_3$ strictly decreasing |
| $a_i > 0,\ b_i > 0,\ c_i < 0$ | $\phi_1$ strictly increasing | $\phi_2$ strictly increasing | $\phi_3$ strictly decreasing |
| $a_i < 0,\ b_i < 0,\ c_i > 0$ | $\phi_1$ strictly decreasing | $\phi_2$ strictly decreasing | $\phi_3$ strictly increasing |
| $a_i > 0,\ b_i < 0,\ c_i < 0$ | $\phi_1$ strictly increasing | $\phi_2$ strictly decreasing | $\phi_3$ strictly decreasing |
| $a_i < 0,\ b_i > 0,\ c_i > 0$ | $\phi_1$ strictly decreasing | $\phi_2$ strictly increasing | $\phi_3$ strictly increasing |
| $a_i > 0,\ b_i < 0,\ c_i > 0$ | $\phi_1$ strictly increasing | $\phi_2$ strictly decreasing | $\phi_3$ strictly increasing |
| $a_i < 0,\ b_i > 0,\ c_i < 0$ | $\phi_1$ strictly decreasing | $\phi_2$ strictly increasing | $\phi_3$ strictly decreasing |

## 7 Optimization

As seen in the previous section, if the systems $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ do not admit a positive trivial solution, we can replace $\Delta(\sigma,\theta,\upsilon)$ in the disjunction $\bigvee_{P\in\Delta(\sigma,\theta,\upsilon)} C_P$ by a subset $\Delta'(\sigma,\theta,\upsilon)$, where all the paths in $\Delta'(\sigma,\theta,\upsilon)$ are contained on a fixed number of planes.

Using the notion of constraint implication [11], we show in this section that this replacement can sometimes be done even if the systems $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$ admit a positive solutions.

This simplification is due to a combination of directions of the constraint implications of the program. Unfortunately, this simplification does not work for any possible combination. Let us first explain the notion of constraint implication.

*Constraint implications* are ordering relations on the constraints of adjacent points of $\mathbb{N}\times\mathbb{N}\times\mathbb{N}$. For programs of form $\pi$, the existence of those constraints is obvious. Proposition 7.1 states the existence of these constraint implications when the recursive rule $R_k$ $(1 \leq n \leq 3)$ is of the form:

$$p(x+a_k, y+b_k, z+c_k) \quad :- \quad \Phi_k(x,y,z),\ p(x,y,z)$$

where $\Phi_k(x,y,z)$ denotes an inequality of the form $d_k x + e_k y + f_k z + g_k \odot_k 0$, with $\odot_k \in \{<,>,\leq,\geq\}$.

**Proposition 7.1** *Consider a program $\pi$ and $\Phi_k(x,y,z)$ $(1 \geq k \geq 3)$ be the constraint of the recursive rule $R_k$. For all $\alpha$, $\beta$ and $\gamma$:*

*I. Either $\phi_k(\alpha,\beta,\gamma)$ is stronger than $\phi_k(\alpha+1,\beta,\gamma)$ or vice-versa.*

*II. Either $\phi_k(\alpha,\beta,\gamma)$ is stronger than $\phi_k(\alpha,\beta+1,\gamma)$ or vice-versa.*

*III. Either $\phi_k(\alpha,\beta,\gamma)$ is stronger than $\phi_k(\alpha,\beta,\gamma+1)$ or vice-versa.*

Proposition 7.1 defines 9 *constraint implications*, 3 for each constraint $\phi_k(\alpha,\beta,\gamma)$. These constraint can be represented graphically in figure 25.

**Example 7.2** For the program $\pi_3$ of subsection 6.1, the directions of the constraint implications are represented in figure 26.

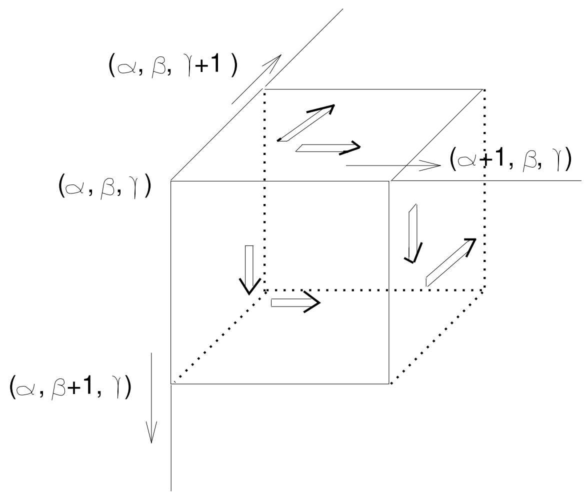


Figure 25: Constraint implications

If a program has its constraint implications oriented in a certain direction, we can determine the form of the paths in $\Delta(\sigma, \theta, v)$ without solving the systems $\Sigma_1$, $\Sigma_2$ and $\Sigma_3$.

This optimization consists in simplifying the forms of the paths in $\Delta'(\sigma, \theta, v)$, i.e, to obtain a "smaller" [2] subset $\Delta''(\sigma, \theta, v)$ such that:

$$\bigvee_{\mathcal{P} \in \Delta(\sigma,\theta,v)} C_{\mathcal{P}} = \bigvee_{\mathcal{P} \in \Delta''(\sigma,\theta,v)} C_{\mathcal{P}}$$

For programs with two recursive rules, the form of these paths sets was studied at [11].

In the next three subsections, we consider some combinations of the directions of constraint implications of programs with three recursive rules, for which the form of the paths in $\Delta(\sigma, \theta, v)$ does not depend on the values of $a_1$, $a_2$, $a_3$, $b_1$, $b_2$, $b_3$, $c_1$, $c_2$ and $c_3$. These three classes cover approximately 80 % of all the possible combinations of these implications. (Further optimizations allow to simplify $\Delta(\sigma, \theta, v)$ in a similar way more than 87 % of all the possible combinations.)

Note that a program $\pi$ can belong to more than one of this classes.

## 7.1 Three Constraint Implications in a Same Direction

For a program $\pi$ having constraint implications as shown in figure 27 (left), the set of paths in $\Delta''(\sigma, \theta, v)$ are of the form depicted in figure 27 (right).

There are 6 combinations of constraint implications that lead to simplifications analogous to this one.

[2] By "smaller" subset, we mean a subset that has a simpler form than the one of $\Delta'(\sigma, \theta, v)$

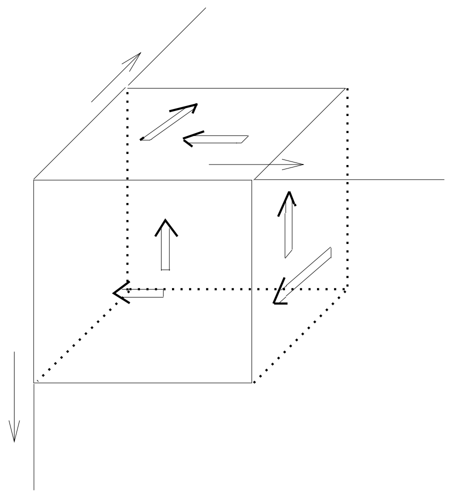


Figure 26: Constraint implications for program $\pi_3$

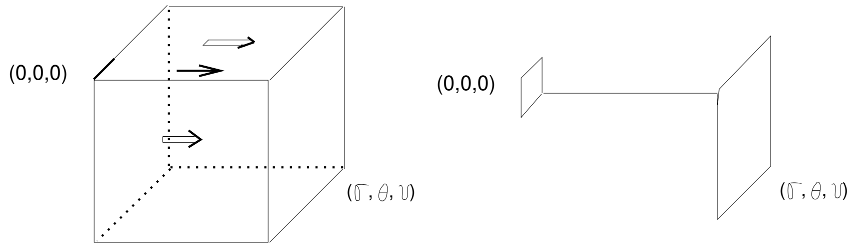


Figure 27:

## 7.2 Two co-planar constraint implications with same direction

For a program $\pi$ having constraint implications as shown in figure 28 (left), the set of paths in $\Delta''(\sigma, \theta, v)$ are of the form depicted in figure 28 (right).

There are 6 combinations of constraint implications that lead to simplifications analogous to this one. The remaining 5 cases are depicted in figure 29.

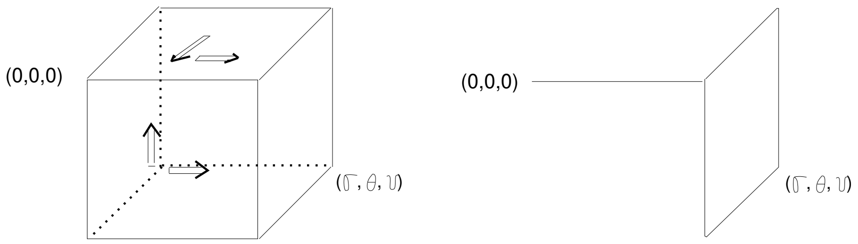


Figure 28:

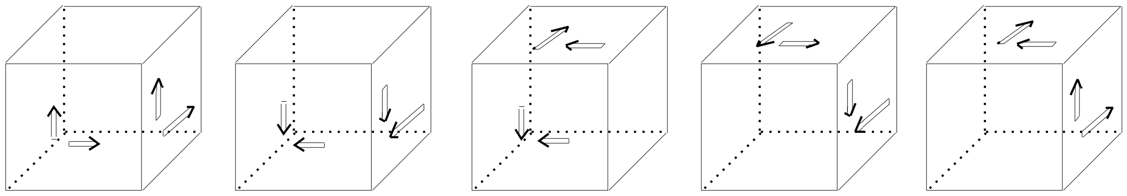


Figure 29:

## 7.3 Two co-linear constraint implications with opposed directions

For a program $\pi$ having constraint implications as shown in figure 30 (top), the set of paths in $\Delta''(\sigma, \theta, \upsilon)$ are of the form depicted in figure 30 (bottom).

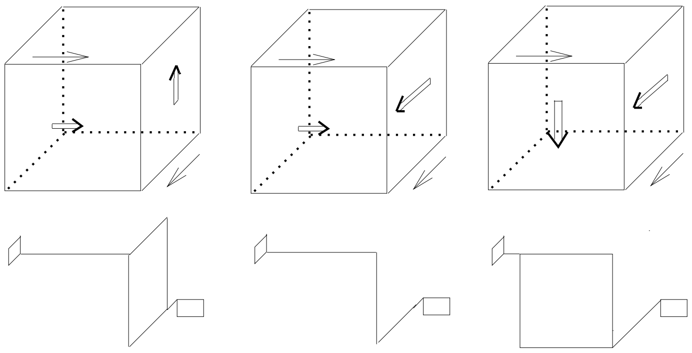


Figure 30:

There are 15 others combinations of constraint implications like the ones shown in figure 30, corresponding to other positions of the two single arrows.

## 8 Applications

### 8.1 Generation of lemmas

As explained at the beginning of this paper, our initial motivation for studying bottom-up evaluation of Datalog programs with integers, is its applicability to the automatic generation of lemmas. More precisely, given a logic program defining a predicate of the form $p(L,x,y,z)$ where $L$ is a variable denoting a list of integers, and $x,y,z$ are variables denoting integers, it is often possible to transform the atom $p(L,x,y,z)$ into an atom of the form $p'(x,y,z)$ defined only over integer arguments (see [10]). The bottom-up evaluation of $p'(x,y,z)$ then yields an arithmetic relation $\psi(x,y,z)$ that also holds for $p(L,x,y,z)$, generating the lemma: $p(L,x,y,z) \Rightarrow \psi(x,y,z)$.
In the motivation section, we have thus sketched out how to generate the lemma $p(L,v,w,x,y,z) \Longrightarrow (v \neq w \Longrightarrow z \leq y \; div \; 2)$. The method applies also if the hypothesis contains more than one predicate. For example, one can generate lemmas of the form $p_1(L,x_1,y_1,z_1) \wedge p_2(L,x_2,y_2,z_2) \Rightarrow \psi(x_1,y_1,z_1,x_2,y_2,z_2)$; it suffices to replace the conjunction $p_1(L,x_1,y_1,z_1) \wedge p_2(L,x_2,y_2,z_2)$ by an equivalent atom $p_3(L,x_1,y_1,z_1,x_2,y_2,z_2)$ and to apply the previous method to this new atom. This application to the automatic generation of lemmas is useful for proving safety properties of communicating automata and protocols [12].

### 8.2 Termination of Prolog programs

As seen in section 2, bottom-up evaluation of Datalog programs with integers is useful for proving the termination of logic programs. The Datalog programs correspond to logic programs in which the original arguments have been replaced by their sizes. (This can be viewed as a form of "abstract interpretation" [6].) As an illustration consider the following Prolog program (borrowed from [26]).

$$\begin{array}{lll}
split([\,],y,[\,],[\,]). & & \\
split([u|L],y,[u|M],N) & :- & u \leq y, split(L,y,M,N). \\
split([u|L],y,M,[u|N]) & :- & u > y, split(L,y,M,N).
\end{array}$$

By replacing the list arguments by their sizes (numbers of elements) and removing the auxiliary constraints $u \leq y$ and $u > y$, one transforms the above program into the Datalog program:

$$\begin{array}{lll}
split'(x,y,z,v) & :- & x=0, z=0, v=0. \\
split'(x+1,y,z+1,v) & :- & split'(x,y,z,v). \\
split'(x+1,y,z,v+1) & :- & split'(x,y,z,v).
\end{array}$$

Our method applied to *split'* generates an arithmetical formula equivalent to $x = z + v$. This means that the size of the fourth argument of *split* is the sum of the sizes of the first and third arguments. This relation among the sizes of the arguments of *split* can be used for showing the termination of a *quicksort* program (see [26]). Note that Plümer's (top-down) procedure generates the weaker relation $x \geq z + v$.

## 8.3 Temporal Deductive Databases

A Datalog program is a logic program without function symbols. A Temporal Datalog program is a logic program where function symbols (e.g. *succ*, $+$) are allowed for the "temporal" arguments. Given a Temporal Datalog program and a query $p(x, y, z)$, the query processing problem consists to determine whether or not there exists a ground substitution $\sigma$ such that $\sigma(p(x, y, z))$ logically follows from the program (and to determine all of these substitutions, if any). This problem is undecidable for general temporal Datalog programs, but has been proven decidable for temporal Datalog programs which have a *unique* temporal argument over which incrementation is done systematically (no arithmetic constraint) [8]. Our process of bottom-up evaluation makes the query processing problem decidable for a new class of temporal Datalog programs: these programs have an *arbitrary number* of temporal arguments over which incrementation is done conditionally (according to the satisfaction of an arithmetic constraint); on the other hand, our programs contain *at most three* recursive rules.

## 9 Conclusion

We have described a bottom-up evaluation method for a special class of recursive Datalog programs over integers that are often met in Temporal Deductive Databases, or when proving termination of logic programs.
The method shows how to construct a linear arithmetic formula which characterizes the least fixpoint associated with the program. The recursive rules of the programs that we consider increment the arithmetic arguments according to one arithmetic constraint. We have described here the method for the case where programs were made of at most three recursive rules. We believe that the underlying idea of the method is general and can be applied to the general case of $n$ recursive rules.
Moreover, using additional techniques such as constraint "pushing" (see, e.g., [29]), our method can be applied in many cases where the programs have more than one constraint per rule.
We have assumed in this paper that the domain of the variables and constants were the domain of integer numbers, but our method can be applied if the domain of the variables and constants is the domain of *rational* or *real* numbers. We have also assumed that $\xi$ was a linear arithmetic formula, but the method applies as well for *general arithmetic* formulas. Further generalizations on the form of program $\pi$ are given in [11].

## References

[1] K. R. Apt, M. H. Van Emden. "Contributions to the Theory of Logic Programming". *J.ACM* 29, 1982, pp. 841-862.

[2] M. Baudinet. "Temporal Logic Programming is Complete and Expressive". *Proc. 16th ACM Symp. on Principles of Programming Languages*, Austin, 1989, pp. 267-280.

[3] M. Baudinet, M. Niezette and P. Wolper. "On the Representation of Infinite Temporal Data Queries". *Proc. 10th ACM Symp. on Principles of Database Systems*, Denver, 1991, pp. 280-290.

[4] F. Bancillon and R. Ramakrishnan. "An Amateur's Introduction to Recursive Query Processing Strategies", *Proc. ACM Conf. on Management of Data*, Washington, 1986, pp. 16-52.

[5] A. Brodsky and Y. Sagiv. "Inference of Inequality Constraints in Logic Programs". *Proc. 10th ACM Symp. on Principles of Database Systems*, Denver, 1991, pp. 227-240.

[6] P. Cousot and R. Cousot. "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints", *Conference Record of the 4th ACM Symposium on Principles of Programming Languages*, Los Angeles, 1977, pp. 238-252.

[7] P. Cousot and N. Halbwachs. "Automatic Discovery of Linear Restraints among Variables of a Program". *Conference Record 5th ACM Symp. on Principles of Programming Languages*, Tucson, 1978, pp. 84-96.

[8] J. Chomicki and T. Imielinski. "Temporal Deductive Databases and Infinite Objects". *Proc. 7th ACM Symp. on Principles of Database Systems*, Austin, 1988, pp. 61-81.

[9] M. H. Van Emden and R. A. Kowalski. "The Semantics of Predicate Logic as a Programming Language", *J.ACM* 23:4, 1976, pp. 733-742.

[10] L. Fribourg. "Mixing List Recursion and Arithmetic". *Proc. 7th IEEE Symp. on Logic in Computer Science*, Santa Cruz, 1992, pp. 419-429.

[11] L. Fribourg and M. Veloso Peixoto. "Bottom-up Evaluation of Datalog Programs with Arithmetic Constraints". Technical report, LIENS 92-13, June 1992. Short version in Proc. CADE-12, LNAI 814, pp. 311-325, June 1994.

[12] L. Fribourg and M. Veloso Peixoto. "Concurrent Constraint Automata". Technical report, LIENS 93-10, May 1993.

[13] A. Van Gelder. "Deriving Constraints among Argument Sizes in Logic Programs", *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 47-60.

[14] J. Jaffar and J.L. Lassez. "Constraint Logic Programming", *Proc. 14th ACM Symp. on Principles of Programming Languages*, 1987, pp. 111-119.

[15] G. Kuper. "On the Expressive Power of the Relational Calculus with Arithmetic Constraints". *Proc. 3rd International Conference on Database Theory*, Paris, 1990, pp. 203-313.

[16] P. Kanellakis, G. Kuper and P. Revesz. "Constraint Query Languages". *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 299-313.

[17] D.B. Kemp and P.J. Stuckey. "Analysis Based Constraint Query Optimization". *Proc. 10th Intl. Conf. on Logic Programming*, Budapest, The MIT Press, 1993, pp. 666-682.

[18] F. Kabanza, J.M. Stevenne and P. Wolper. "Handling Infinite Temporal Data". *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 392-403.

[19] J-L Lassez. “Querying Constraints”. *Proc. 9th ACM Symp. on Principles of Database Systems*, Nashville, 1990, pp. 288-298.

[20] J-L Lassez. “Parametric Queries, linear constraints and variable elimination”. *Proc. Conference on Design and Implementation of Symbolic Computer*, LNCS 429, 1990, pp. 164-173.

[21] J. Lloyd. “Foundations of Logic Programming”. Second Edition, Springer Verlag, Berlin, 1987.

[22] A. Levy and Y. Sagiv. “Constraints and Redundancy in Datalog”. *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego, 1992, pp. 67-80.

[23] R. Van der Meyden. “Reasoning with Recursive Relations: Negation, Inequality and Linear Order (Extended Abstract)”. *Proc. ILPS Workshop on Deductive Databases*, San Diego, 1991.

[24] J. Misra and D. Gries. “Finding Repeated Elements”, *Science of Computer Programming 2*, 1982, pp. 143-152.

[25] K. Marriot and P.J. Stuckey. “The 3 R’s of Optimizing Constraint Logic Programs: Refinement, Removal and Reordering”. *Proc. 20th ACM Symp. on Principles of Programming Languages*, Charleston, 1993, pp. 334-344.

[26] L. Plümer. “Termination Proofs for Logic Programs based on Predicate Inequalities”.*Proc. 7th International Conference on Logic Programming*, Jerusalem, 1990, pp. 634-648.

[27] P. Revesz . “A Closed Form for Datalog Queries with Integer Order”. *Proc. 3rd International Conference on Database Theory*, Paris, 1990, pp. 187-201.

[28] D. Srivastava. “Subsumption in Constraint Query Languages with Linear Arithmetic Constraints”. *Proc. 2nd International Symp. on Artificial Intelligence and Mathematics*, Fort Lauderdale, 1992.

[29] D. Srivastava and R. Ramakrishnan. “Pushing Constraint Selections”. *Proc. 11th ACM Symp. on Principles of Database Systems*, San Diego, 1992, pp. 301-315.

[30] J. D. Ullman and A. Van Gelder. “Efficient Test for Top-Down Termination”, *J.ACM* 35:2, 1988, pp. 345-373.