On termination of binary CLP programs

Alexander Serebrenik[1] and Fred Mesnard[2]

[1] École Polytechnique (STIX), 91128 Palaiseau Cedex, France
E-mail: Alexander.Serebrenik@stix.polytechnique.fr
[2] IREMIA, université de La Réunion, France
E-mail: fred@univ-reunion.fr

Abstract. Termination of binary CLP programs has recently become an important question in the termination analysis community. The reason for this is due to the fact that a number of approaches to termination of logic programs abstract the input program to a binary CLP program and conclude termination of the input program from termination of the abstracted program. In this paper we introduce a class of binary CLP programs such that their termination can be proved by using linear level mappings. We show that membership to this class is decidable and present a decision procedure. Further, we extend this class to programs such that their termination proofs require a combination of linear functions. In particular we consider as level mappings tuples of linear functions and piecewise linear functions.

1 Introduction

Termination is well-known to be one of the crucial properties of software verification. Logic programming with its strong theoretical basis lends itself easily to termination analysis as witnessed by a very intensive research in the area. Some of the recent approaches to termination [3, 9, 12] proceed in two steps. First, a logic program is abstracted to a CLP($\mathcal{N}$)-program, *i.e.* logic program extended with constraint solving over the domain of natural numbers $\mathcal{N}$. Second, the CLP($\mathcal{N}$)-program is approximated by a *binary* CLP($\mathcal{N}$) program, *i.e.*, a set of clauses of the form $p(\tilde{x}) \leftarrow c, q(\tilde{y})$, where c is a CLP-constraint and p, q are user-defined predicates.

In this paper we study decidability of termination for binary CLP($\mathcal{C}$) programs. In general, it depends on the constraints domain $\mathcal{C}$. On the one hand, Devienne et al. [5] have established undecidability of termination for one-clause binary CLP($\mathcal{H}$) programs, where $\mathcal{H}$ is the domain of Herbrand terms. Similar results can be obtained for other CLP languages such as CLP($\mathcal{N}$) and CLP(Q). On the other hand, Datalog, *i.e.*, logic programming language with no function symbols, provides an example of a constraint programming language such that termination is decidable for it.

For constraint domains with the undecidable termination property, we are interested in subclasses of binary programs such that termination is decidable for these subclasses. A trivial example of such a subclass is the subclass of non-recursive binary programs. After the preliminary remarks of Section 2, in Section 3 we present our main result, namely a non-trivial subclass of terminating binary CLP($\mathcal{C}$) programs such that membership to this subclass is decidable if $\mathcal{C}$ is $\mathcal{Q}$, $\mathcal{Q}^+$ or $\mathcal{R}$. Intuitively, this is the class of binary CLP programs such that there exists a linear function decreasing while traversing the clauses. Two extensions of this class are discussed in Section 4.

2 Preliminaries

2.1 Constraint Logic Programming

We adhere to the definitions of [7]. For sake of completeness we recapitulate them briefly. A constraint domain $\mathcal{C}$ is a tuple $(\Sigma_\mathcal{C}, \mathcal{L}_\mathcal{C}, \mathcal{D}_\mathcal{C}, \mathcal{T}_\mathcal{C}, solv_\mathcal{C})$. The *domain signature* $\Sigma_\mathcal{C}$ is a pair $(F_\mathcal{C}, \Pi_\mathcal{C})$, where $F_\mathcal{C}$ is the set of function symbols and $\Pi_\mathcal{C}$ is the set of predicate symbols. The class of constraints $\mathcal{L}_\mathcal{C}$ is a set of first order formulae closed under conjunction and existential quantification. The domain of computation $\mathcal{D}_\mathcal{C}$ is the intended interpretation of constraints over a set $D_\mathcal{C}$. The constraint theory $\mathcal{T}_\mathcal{C}$ describes the logical semantics of the constraints. Finally, the constraint solver $solv_\mathcal{C}$ maps each formula in $\mathcal{L}_\mathcal{C}$ to $\{\mathit{true}, \mathit{false}, \mathit{unknown}\}$, such that for any $c \in \mathcal{L}_\mathcal{C}$, $solv_\mathcal{C}(c) = \mathit{false}$ implies $\mathcal{T}_\mathcal{C} \models \neg\exists c$, and $solv_\mathcal{C}(c) = \mathit{true}$ implies $\mathcal{T}_\mathcal{C} \models \exists c$. A constraint solver is called *complete* if it only returns true or false. A constraint domain with a complete solver is called *ideal*. A constraint logic programming language over an ideal domain is also called *ideal*.

We consider the following ideal constraint domains:

- $\mathcal{N}$. The predicate symbols are $=$ and $\geq$, the function symbols are 0, 1, and $+$. The constraint theory $\mathcal{T}_\mathcal{N}$ is the theory of Presburger arithmetic, known to be decidable. It should be noted that constraints produced by the abstraction techniques of [3, 9, 12] can be expressed in Presburger arithmetic.
- $\mathcal{Q}$ and $\mathcal{R}$. The predicate symbols are as above, the function symbols are 0, 1, $+$, $-$, $*$, and $/$, however only linear constraints are admitted. $\mathcal{Q}^+$ and $\mathcal{R}^+$ restrict $\mathcal{Q}$ and $\mathcal{R}$ to non-negative numbers.

Given a program P, we define Π_P as the set of user-defined predicate symbols appearing in P. Syntactic objects are viewed modulo renaming of variables. In this paper we restrict our attention to binary programs. We assume that binary rules are in *flat* form: $p(\tilde{x}) \leftarrow c, q(\tilde{y})$, with $\tilde{x} \cap \tilde{y} = \varnothing$ (where $\varnothing$ denotes the empty set and $\tilde{x}$ a tuple of *distinct* variables). Flat facts and flat queries are defined accordingly. An *atomic query* is a flat query of the form $c, q(\tilde{y})$ where $q \in \Pi_P$.

A $\mathcal{C}$-interpretation for a CLP($\mathcal{C}$) program P is an interpretation on the domain signature $(F_{\mathcal{C}}, \Pi_{\mathcal{C}} \cup \Pi_P)$ that agrees with the domain of computation $\mathcal{D}_{\mathcal{C}}$ on the interpretation of the symbols in $\Sigma_{\mathcal{C}}$. Given a CLP($\mathcal{C}$)-program P, the $\mathcal{C}$-base $B_P^{\mathcal{C}}$ is defined as $\{p(d_1,\ldots,d_n) \mid p \in \Pi_P, (d_1,\ldots,d_n) \in (D_{\mathcal{C}})^n\}$. A $\mathcal{C}$-interpretation can be regarded as a subset of the $\mathcal{C}$-base. A $\mathcal{C}$-model of a program P is a $\mathcal{C}$-interpretation that is also a model of P.

A *valuation* θ is a function that maps all variables to $D_{\mathcal{C}}$. For an interpretation J and a formula φ we write $J \models_\theta \varphi$ if $\theta(\varphi)$ is valid with respect to J. For an atomic query Q of the form (c,A), we define $ground_{\mathcal{C}}(Q) = \{\theta(A) | \mathcal{D}_{\mathcal{C}} \models_\theta c\}$. For a binary rule we define $ground_{\mathcal{C}}((A \leftarrow c, B)) = \{\theta(A \leftarrow B) | \mathcal{D}_{\mathcal{C}} \models_\theta c\}$. Similarly, for a program P, $ground_{\mathcal{C}}(P)$ is the set of ground $\mathcal{C}$-instances of the rules of P.

2.2 Termination analysis

In this subsection we present briefly a number of notions related to termination analysis. First of all, we say that a CLP($\mathcal{C}$) program P and a query Q *left-terminate* if every derivation of Q with respect to P via the leftmost selection rule is finite.

One key concept in many (theoretical) approaches lies in the use of *level mappings*, *i.e.*, mappings from ground atoms to natural numbers. We slightly extend this traditional definition and map the elements of the $\mathcal{C}$-base to a well-founded set. We prefer to talk about a general well-founded set rather than about the set of the naturals, in order to be able to consider functions to $\mathcal{R}^+$ and $(\mathcal{R}^+)^m$. Recall that a well-founded set is a partially ordered set $(S, \triangleright)$ such that there is no infinitely decreasing chain $s_1 \triangleright s_2 \triangleright \ldots$ of elements of S. Formally, a *level mapping* for a constraint domain $\mathcal{C}$ is a function $|\cdot| : \mathcal{C}\text{-}base \to S$. It is well-known that termination of a CLP program can be characterised by means of level mappings. The following definition is taken from [13].

Definition 1. *Let $|\cdot| : \mathcal{C}\text{-}base \to S$ be a level mapping, and I be a $\mathcal{C}$-interpretation. A CLP($\mathcal{C}$) program P is* acceptable by $|\cdot|$ and I *if I is a $\mathcal{C}$-model of P, and for every $A \leftarrow B_1,\ldots,B_n$ in $ground_{\mathcal{C}}(P)$, for $i \in [1,n]$, $I \models B_1,\ldots,B_{i-1}$ implies $|A| \triangleright |B_i|$. A query Q is* acceptable by $|\cdot|$ and I *if there exists $k \in S$ such that for every $A_1,\ldots,A_n$ in $ground_{\mathcal{C}}(Q)$, for $i \in [1,n]$, $I \models A_1,\ldots,A_{i-1}$ implies $k \triangleright |A_i|$.*

For binary programs and atomic queries, model and queries (see Proposition 1) can be eliminated from the previous definition. By doing so we can obtain a notion similar to *recurrency*. Originally the notion of recurrency has been introduced in [1] to characterise termination of ground queries to logic programs for all selection rules. For constraint logic programming we introduce the following definition:

Definition 2. *Let P be a binary CLP($\mathcal{C}$) program, and $|\cdot| : \mathcal{C}$-*base $\to S$ *be a level mapping. P is called* recurrent *with respect to $|\cdot|$ if for every $A \leftarrow B \in ground_{\mathcal{C}}(P)$, $|A| \rhd |B|$ holds.*

The following lemma states that for binary programs the notions of acceptability and recurrency coincide.

Lemma 1. *Let P be a binary CLP($\mathcal{C}$) program and $|\cdot| : \mathcal{C}$-*base $\to S$ *be a level mapping. Then, P is acceptable by $|\cdot|$ and the $\mathcal{C}$-base if and only if P is recurrent with respect to $|\cdot|$.*

The relationship between acceptability and termination for ideal CLP languages can be expressed by the following theorems:

Theorem 1. *([13]) Let CLP($\mathcal{C}$) be an ideal CLP language. If a program P and a query Q are both acceptable by some level mapping $|\cdot|$ and a $\mathcal{C}$-model I then they left terminate.*

From here on we consider only ideal constraint logic programming languages. This assumption is quite common in termination analysis for CLP. For binary programs one can use Lemma 1 and replace acceptability with respect to a level mapping and a model by recurrency with respect to a level mapping.

Observe that we do not need to introduce the corresponding notion of recurrency for queries. Instead, in order to take care of the atomic query Q we extend the corresponding binary program P by a clause $q \leftarrow Q$, where q is a *fresh* predicate symbol, *i.e.*, $q \not\in \Pi_P, q \not\in \Pi_{\mathcal{C}}$. The basic idea is that recurrency of $P \cup \{q \leftarrow Q\}$ implies termination of Q with respect to P. Formally, the following proposition holds.

Proposition 1. *Let P be a binary CLP($\mathcal{C}$) program, Q be an atomic query, and q be a fresh predicate symbol as above. If $P \cup \{q \leftarrow Q\}$ is recurent with respect to a level mapping $|\cdot|$ then Q terminates with respect to P.*

2.3 Linear programming

In this subsection we recall briefly some basic notions of linear programming (see [15] for instance) to be applied in Section 3.1. Essentially, linear programming aims at finding the extremum of a linear function of positive numbers, so called the *objective function*, given that a system of linear inequalities on these variables holds. Formally, a minimising linear programming problem can be expressed as follows: *minimise* $\tilde{c}\tilde{x}^T$ *subject to* $A\tilde{x}^T \geq \tilde{b}^T$ *and* $\tilde{x} \geq 0$, where $\tilde{x}$ is a vector of variables, $\tilde{c}$ expresses the objective function, the superscript ^{T} denotes a transposed of a vector, and $A\tilde{x}^T \geq \tilde{b}^T$ denotes the system of linear constraints.

For every minimising linear programming problem over the rationals or the reals, there exists a maximising linear programming problem, called *dual*, such that an optimal solution to one problem ensures the existence of an optimal solution to the other and that the optimal values of the objective functions are equal. This statement is known as *the duality theorem*. Given a minimising linear programming problem as above, the dual linear programming problem has the following form: *maximise* $\tilde{y}\tilde{b}^T$ *subject to* $\tilde{y}A^T \leq \tilde{c}$ *and* $\tilde{y} \geq 0$.

3 Llm-recurrent programs

In this section we consider a special subclass of binary programs and atomic queries. In particular, we are interested in programs and queries that can be analysed by means of linear level mappings. Let $\mathcal{C}$ be $\mathcal{N}$ or $\mathcal{Z}$ or $\mathcal{Q}^+$ or $\mathcal{Q}$ or $\mathcal{R}$. As a range for a level mapping in this section we take $(\mathcal{R}^+, \rhd)$, where $x \rhd y$ holds if $x \geq y+1$.

Definition 3. *A level mapping* $|\cdot|$: $\mathcal{C}$-base $\to \mathcal{R}^+$ *is called* linear *if for any n-ary predicate symbol p, there exist real numbers* μ_p^i, $0 \leq i \leq n$, *such that for any atom* $p(e_1,\ldots,e_n) \in \mathcal{C}$-base, $|p(e_1,\ldots,e_n)| = max(0, \mu_p^0 + \sum_{i=1}^n \mu_p^i e_i)$.

Using the notion of a linear level mapping we can define the class of programs we are going to study.

Definition 4. *Let P be a binary flat CLP($\mathcal{C}$) program. We say that P is* llm-recurrent *if there exists a linear level mapping* $|\cdot|$ *such that P is recurrent with respect to it.*

Example 1. Consider the following program: $\mathtt{p(X)} \leftarrow \mathtt{X} \leq \mathtt{72}, \mathtt{Y} = \mathtt{X}+\mathtt{1}, \mathtt{p(Y)}$. This program is llm-recurrent with respect to $|\mathtt{p(x)}| = max(0, 73-\mathtt{x})$. $\square$

In the next subsection we quickly review the algorithm of Sohn and Van Gelder [17] that aims at checking the existence of a linear level mapping such that P is llm-recurrent with respect to it. This will allow us to show that llm-recurrency is decidable for $\mathcal{Q}$ and $\mathcal{R}$.

3.1 The algorithm SVG

The algorithm of Sohn and Van Gelder (SVG) examines each recursive user-defined predicate symbol p of a CLP($\mathcal{Q}^+$) program in turn (the precise order does not matter) and try to find a level mapping for $p(x_1,\ldots,x_n)$ symbolically defined as $|p(\tilde{x})| = \mu_0 + \sum_{1\leq i\leq n} \mu_i x_i$ where $\mu_i \geq 0$ for all i. For sake of simplicity, we assume that the program is only *directly* recursive. By this we mean that if there exist sequences of predicates $p = r_0, r_1, \ldots, r_n = q$ and $q = r_n, r_{n+1}, \ldots, r_m = p$ such that for all i, $r_i(\tilde{x}) \leftarrow c(\tilde{x}, \tilde{y}), r_{i+1}(\tilde{y})$ is a clause in P, then p is identical to q. Moreover, we may safely ignore the constant μ_0.

For every rule r, say $p(\tilde{x_0}) \leftarrow c, p(\tilde{x_k})$, we assume that the constraint c is satisfiable, already projected onto $\tilde{x_0} \cup \tilde{x_k}$, only contains inequalities of the form $e_1 \geq e_2$, with e_1 and e_2 being arithmetical expressions over $\tilde{x_0} \cup \tilde{x_k}$ and constants. For such a rule recurrency requires that c implies $\sum_{1 \leq i \leq n} \mu_i x_i^0 - \sum_{1 \leq i \leq n} \mu_i x_i^k \geq 1$, where $\tilde{x_0}$ is the vector $(x_1^0, \ldots, x_n^0)$ and $\tilde{x_k}$ is the vector $(x_1^k, \ldots, x_n^k)$. In other words, such a binary rule gives rise to the following *pseudo*[3] linear programming problem

$$\textit{minimise} \;\; \theta = \tilde{\mu}(\tilde{x_0} - \tilde{x_k}) \; \textit{subject to } c, \; \tilde{x_0} \geq 0, \; \tilde{x_k} \geq 0 \qquad (1)$$

where $\tilde{\mu}$ is the vector $(\mu_1, \ldots, \mu_n)$. A level mapping $|\cdot|$ ensuring recurrency exists (at least for this clause) if $\theta^* \geq 1$ where θ^* denotes the minimum of the objective function. Because of the symbolic constants $\tilde{\mu}$, (1) is *not* a linear programming problem. The idea of Sohn and Van Gelder is to consider its dual form:

$$\textit{maximise} \;\; \eta = \beta \tilde{y} \; \textit{subject to } A\tilde{y} \leq (\mu_1, \ldots, \mu_n, -\mu_1, \ldots, -\mu_n), \; \tilde{y} \geq 0 \qquad (2)$$

where β and A are automatically derived while switching to the dual form of (1) and $\tilde{y}$ is the vector of dual variables. By the duality theorem of linear programming, we have $\theta^* = \eta^*$. Now, the authors observe that $\tilde{\mu}$ appears linearly in the dual problem (it is not true for (1)) because no μ_i appears in A. Hence the constraints of (2) can be rewritten, by adding $\eta \geq 1$, $\tilde{y} \geq 0$, as a set of linear inequations, denoted S_r. If the conjunction $S_p = \wedge_k S_r$ (for each clause defining p) is satisfiable, then there exists a linear level mapping for p ensuring recurrency.

Example 2. We consider the CLP(Q^+) program P:

$$\mathtt{p(X_1, X_2)} \leftarrow \mathtt{X_1 + 2 * X_2 \geq 3 * X_3 + 4 * X_4 + 1, p(X_3, X_4)}.$$

The first step is the pseudo-linear program: *minimise* $\theta = a(x_1 - x_3) + b(x_2 - x_4)$ *subject to* $x_1, x_2, x_3, x_4 \geq 0, x_1 + 2x_2 \geq 3x_3 + 4x_4 + 1$. We get: *minimise* $\theta = [a \; b \; -a \; -b]\,[x_1 \; x_2 \; x_3 \; x_4]^T$ *subject to* $A\,[x_1 \; x_2 \; x_3 \; x_4]^T \geq [0 \; 0 \; 0 \; 1]^T$, where A is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & -3 & -4 \end{bmatrix}$. The dual form is: *maximise* $\eta = [0 \; 0 \; 0 \; 0 \; 1]\,[y_1 \; y_2 \; y_3 \; y_4 \; y_5]^T$ *subject to* $A^T\,[y_1 \; y_2 \; y_3 \; y_4 \; y_5]^T \leq [a \; b \; -a \; -b]^T$ *and* $y_1, y_2, y_3, y_4, y_5 \geq 0$. The parameters a and b now appear linearly, they will be considered as new variables and we have: *maximise* $\eta = [0 \; 0 \; 0 \; 0 \; 1 \; 0 \; 0]\,[y_1 \; y_2 \; y_3 \; y_4 \; y_5 \; a \; b]^T$ *subject to*

[3] because *symbolic* parameters appear in the objective function.

$$\begin{bmatrix} 1\,0\,0\,0 & 1 & -1 & 0 \\ 0\,1\,0\,0 & 2 & 0 & -1 \\ 0\,0\,1\,0 & -3 & 1 & 0 \\ 0\,0\,0\,1 & -4 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ a \\ b \end{bmatrix} \leq \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$ *and* $y_1, y_2, y_3, y_4, y_5, a, b \geq 0$. As the system S_p (where $\eta = y_5$):

$$\begin{cases} y_5 & \geq 1 \\ y_1 + y_5 - a & \leq 0 \\ y_2 + 2y_5 - b & \leq 0 \\ y_3 - 3y_5 + a & \leq 0 \\ y_4 - 4y_5 + b & \leq 0 \\ y_1, y_2, y_3, y_4, y_5, a, b & \geq 0 \end{cases}$$

is satisfiable, we conclude that there exists a linear level mapping ensuring recurrency of P. $\square$

So SVG is basically an efficient procedure for deciding in $\mathcal{R}$ (or any other domain such that the duality theorem holds for it) the formula $\exists \tilde{\mu}\, \forall \tilde{x} \cup \tilde{y} [c(\tilde{x}, \tilde{y}) \rightarrow \tilde{\mu}\tilde{x} \geq 1 + \tilde{\mu}\tilde{y}]$ corresponding to the rule $p(\tilde{x}) \leftarrow c(\tilde{x}, \tilde{y}), p(\tilde{y})$. It produces a linear constraint (the system S_p in our example) such that satisfiability of this constraint is *equivalent* to a positive answer for the decision problem.

3.2 Verifying llm-recurrency with SVG

To prove llm-recurrency, we need to find a function satisfying Definition 3, *i.e.*, for every predicate p we are looking for a vector $\tilde{\mu_p}$, such that $max(0, \mu_p^0 + \sum_{i=1}^{n} \mu_p^i e_i)$ decreases while traversing the rules. Hence, we extend SVG to decide the existence of $\tilde{\mu}$ such that for each renamed apart rule $p(\tilde{x}) \leftarrow c(\tilde{x}, \tilde{y}), q(\tilde{y}) \in P$, we have: $\forall \tilde{x} \cup \tilde{y}\ \{c(\tilde{x}, \tilde{y}) \rightarrow [\tilde{\mu_p}\tilde{x} \geq 1 + \tilde{\mu_q}\tilde{y} \wedge \tilde{\mu_q}\tilde{y} \geq 0]\}$. We compute the equivalent constraint corresponding to each rule, and satisfiability of their conjunction is equivalent to llm-recurrency.

Note that for a ground atom $p(\tilde{e})$ we may have $\tilde{\mu_p}\tilde{e} < 0$. But as $|p(\tilde{e})|$ is defined by $max(0, \tilde{\mu_p}\tilde{e})$, we have $|p(\tilde{e})| = 0$. Observe also that $\tilde{\mu_p}\tilde{e} < 0$ may hold only for atoms $p(\tilde{e})$ such that $c(\tilde{e}, \tilde{y})$ is unsatisfiable for all $\tilde{y}$, *i.e.*, atoms with computation of depth 1. The explanation above justifies the following decidability result.

Theorem 2. *SVG is a decision procedure for llm-recurrency of binary constraint logic programs over* $\mathcal{Q}^+$, $\mathcal{Q}$ *and* $\mathcal{R}$.

Example 3. Example 2, continued. We need to find $\mu_p(x,y) = ax + by$ such that

$$ax_1 + bx_2 \geq ax_3 + bx_4 + 1 \textit{ and } ax_3 + bx_4 \geq 0$$
$$\textit{subject to } x_1, x_2, x_3, x_4 \geq 0, x_1 + 2x_2 \geq 3x_3 + 4x_4 + 1$$

One of such solutions is $a = 1, b = 2$ leading to the following linear level mapping $|p(x,y)| = max(0, x + 2y)$. Since constraints solving is done over $\mathcal{Q}^+$ we can further simplify this definition to $|p(x,y)| = x + 2y$. □

Observe that although SVG is not necessarily complete for binary constraint logic programs over $\mathcal{N}$ or $\mathcal{Z}$, it is still a sound way to prove termination of programs over these domains. Indeed, by considering a CLP($\mathcal{N}$) program as a CLP($\mathcal{Q}$) program we enlarge the domain. Hence, if the program terminates over $\mathcal{Q}$ it also terminates over $\mathcal{N}$. The following example illustrates that the converse is not necessarily true.

Example 4. Consider the program:

$$\mathtt{div2(X)} \leftarrow \mathtt{X} > \mathtt{0}, \mathtt{2} * \mathtt{Y} = \mathtt{X}, \mathtt{div2(Y)}.$$

The query $\mathtt{X} = \mathtt{1}, \mathtt{div2(X)}$ terminates with respect to this program if constraint solving is done over $\mathcal{N}$ or $\mathcal{Z}$. This is clearly not the case for $\mathcal{Q}$. □

To estimate the relative importance of this class of binary CLP($\mathcal{N}$) programs we have considered a number of logic programming examples, abstracted them and binarised as proposed in [3]. The class of llm-recurrent programs turned out to be broad enough to include binary CLP($\mathcal{N}$) programs corresponding to *fluctuates*, *mergesort*, *queens*, and *rotate* [2].

4 Extending llm-recurrency

In this section we present two extensions of the class of llm-recurrent programs. Our first extension has been motivated by a local approach for termination [3, 10], while the second one by the previous study of numerical computations [16].

4.1 Tuples of linear functions

The basic idea of our first extension is to consider tuples of linear level mappings. In other words, a level mapping should map any ground atom to an m-tuple of non-negative real numbers, where m is a fixed natural number. As above, we need to guarantee that this set is well-founded. Hence, we combine linear level mappings lexicographically. As a range for tuple-linear level mappings we choose $((\mathcal{R}^+)^m, \triangleright)$, where $x \triangleright y$ holds if $x = (x_1, \ldots, x_m), y = (y_1, \ldots, y_m)$ and there exists $1 \leq i \leq m$ such that for all $j \in [1, i-1]$, $x_j = y_j$ and $x_i \geq y_i + 1$.

Definition 5. *A level mapping* $|\cdot|^m$: $\mathcal{C}$*-base* $\rightarrow ((\mathcal{R}^+)^m, \triangleright)$ *is called* tuple-linear *if for any atom* $p(e_1,\ldots,e_n) \in \mathcal{C}$*-base,*

$$|p(e_1,\ldots,e_n)| = (max(0,\mu_p^{(0,1)} + \sum_{j=1}^{n} \mu_p^{(j,1)} e_j),\ldots,max(0,\mu_p^{(0,m)} + \sum_{j=1}^{n} \mu_p^{(j,m)} e_j))$$

where the coefficients $\mu_p^{(j,i)}$ *are real numbers.*

The order relation on tuples is given as the lexicographic order relationship on $\mathcal{R}^+$. Similarly to the definition above we say that a program P is *tuple llm-recurrent* if there exists a tuple-linear level mapping such that P is recurrent with respect to it. Clearly, tuple llm-recurrency implies termination in the same fashion as in Proposition 1.

Example 5. Consider the following binary CLP($\mathcal{Q}^+$) program:

$$\begin{array}{l}
\mathtt{mul(_,Z)} \leftarrow \mathtt{Z} = \mathtt{0}.\\
\mathtt{mul(N,M)} \leftarrow \mathtt{N} \geq \mathtt{0}, \mathtt{M} > \mathtt{0}, \mathtt{M1} = \mathtt{M} - \mathtt{1}, \mathtt{mul_aux(N,M1,M1)}.\\
\mathtt{mul_aux(X,Y,Z)} \leftarrow \mathtt{X} > \mathtt{0}, \mathtt{Y} > \mathtt{0}, \mathtt{X1} = \mathtt{X}, \mathtt{Y1} = \mathtt{Y} - \mathtt{1}, \mathtt{Z1} = \mathtt{Z},\\
\qquad \mathtt{mul_aux(X1,Y1,Z1)}.\\
\mathtt{mul_aux(X,Y,Z)} \leftarrow \mathtt{X} > \mathtt{0}, \mathtt{Y} = \mathtt{0}, \mathtt{X1} = \mathtt{X} - \mathtt{1}, \mathtt{Y1} = \mathtt{Z}, \mathtt{Z1} = \mathtt{Z},\\
\qquad \mathtt{mul_aux(X1,Y1,Z1)}.\\
\mathtt{mul_aux(X,_,_)} \leftarrow \mathtt{X} = \mathtt{0}.
\end{array}$$

This program is not llm-recurrent. Indeed, if it has been llm recurrent, the maximal depth of any derivation of $\mathtt{mul(n,m)}$ would be linear in $\mathtt{n}$ and $\mathtt{m}$. However, one can see that the maximal depth of such a computation depends on $\mathtt{m} * \mathtt{n}$.

To show that the multiplication example is tuple llm-recurrent we use the following level mapping: $|\mathtt{mul(n,m)}| = (\mathtt{n},\mathtt{m})$, $|\mathtt{mul\_aux(x,y,z)}| = (\mathtt{x},\mathtt{y})$. This level mapping is clearly tuple-linear. To prove the recurrency observe that the following inequalities hold:

$$\begin{array}{l}
|\mathtt{mul(n,m)}| = (\mathtt{n},\mathtt{m}) > (\mathtt{n},\mathtt{m}-\mathtt{1}) = |\mathtt{mul_aux(n,m1,m1)}|\\
|\mathtt{mul_aux(x,y,z)}| = (\mathtt{x},\mathtt{y}) > (\mathtt{x},\mathtt{y}-\mathtt{1}) = |\mathtt{mul_aux(x1,y1,z1)}|\\
|\mathtt{mul_aux(x,y,z)}| = (\mathtt{x},\mathtt{y}) > (\mathtt{x}-\mathtt{1},\mathtt{z}) = |\mathtt{mul_aux(x1,y1,z1)}|
\end{array}$$

An additional example of a program that is tuple llm-recurrent but not llm-recurrent can be obtained by abstracting and binarising *ackermann*. □

Decidability of llm-recurrency implies:

Theorem 3. *Tuple llm-recurrency is decidable for $\mathcal{Q}^+$, $\mathcal{Q}$ and $\mathcal{R}$.*

Proof. Observe that each function of the tuple should decrease at least for one binary clause. Hence, let n_p be the number of binary clauses defining the predicate symbol p, excluding facts. Then m is limited by $max(\{n_p | p \in \Pi_P\})$. Let $\mu^1, \ldots, \mu^m$ be linear functions of the tuple. Then, for each rule $p(\tilde{x}) \leftarrow c, p(\tilde{y})$ with $vars(c) \subseteq \tilde{x} \cup \tilde{y}$, the following should hold:

$$\begin{array}{lc}
\forall \tilde{x} \cup \tilde{y} & \{c \to [\mu^1(\tilde{x} - \tilde{y}) \geq 1 \wedge \mu^1(\tilde{y}) \geq 0]\} \\
 & \vee \\
\forall \tilde{x} \cup \tilde{y} & \{c \to [\mu^1(\tilde{x} - \tilde{y}) = 0 \wedge \mu^2(\tilde{x} - \tilde{y}) \geq 1 \wedge \mu^2(\tilde{y}) \geq 0]\} \\
 & \vee \\
 & \ldots \\
 & \vee \\
\forall \tilde{x} \cup \tilde{y} & \{c \to [\mu^1(\tilde{x} - \tilde{y}) = \ldots = \mu^{m-1}(\tilde{x} - \tilde{y}) = 0 \wedge \mu^m(\tilde{x} - \tilde{y}) \geq 1 \wedge \mu^m(\tilde{y}) \geq 0]\}
\end{array}$$

Each one of the disjuncts is similar to (1), *i.e.*, can be decided by SVG. □

Similarly to the previous case while the method outlined above is not necessarily complete for binary constraint logic programs over $\mathcal{N}$ or $\mathcal{Z}$, it is still a sound way to prove termination of programs over these domains.

4.2 Piecewise linear level mappings

The second extension of the class of the llm recurrent programs has been motivated by our previous study of termination of numerical computations [16]. We have suggested to split the domain of an argument into pairwise disjoint cases, called "adornments" and to specialise the program with respect to the adornments. Termination of the specialised program implies termination of the original one. Moreover, this transformation technique allows us to infer a piecewise linear level mapping proving termination of the original program.

Definition 6. *A level mapping $| \cdot |$: $\mathcal{C}$-base $\to (\mathcal{R}^+, \rhd)$ is called* piecewise linear *if there exist linear level mappings $| \cdot |_1, \ldots, | \cdot |_n$: $\mathcal{C}$-base $\to (\mathcal{R}^+, \rhd)$ such that for all $A \in \mathcal{C}$-base there exists i such that $| A | = | A |_i$ and if $| A | \neq 0$ this i is unique.*

We can also write a piecewise level mapping $| \cdot |$ as follows:

$$|A| = \begin{cases} |A|_1, & |A|_1 \neq 0, \\ \ldots & \ldots \\ |A|_n, & |A|_n \neq 0, \\ 0, & otherwise \end{cases}$$

To see that a piecewise linear level mapping generalises Definition 3 observe that any linear level mapping is a piecewise linear level mapping for $n = 1$. A binary CLP($\mathcal{C}$) program P is called *piecewise llm recurrent* if there exists a piecewise linear level mapping such that P is recurrent with respect to it.

Unlike the results presented in Sections 3 and 4.1 at the moment it is not known to us whether piecewise recurrency is decidable. However, we suggest a technique that upon success allows us to prove piecewise recurrency (and hence, termination). Moreover, this technique finds a piecewise level mapping such that the program is recurrent with respect to it. We present the technique by means of example. Consider the following CLP($\mathcal{N}$) program P:

$$q(X) \leftarrow X + X + Y = 50, q(Y).$$

We are interested in showing termination of $S = \{q(X)\}$.

1. First, we identify the reference points with respect to this clause. *Reference points* with respect to a clause $p(\tilde{x}) \leftarrow c(\tilde{x}, \tilde{y}), p(\tilde{y})$ are solutions of $c(\tilde{x}, \tilde{y}), \tilde{x} = \tilde{y}$. If there is no solution over $\mathcal{N}$ ($\mathcal{Z}$) but there exists a solution $\tilde{x_0}$ over $\mathcal{Q}$ take $\lfloor \tilde{x_0} \rfloor$. For our example, we need to solve $x + x + y = 50, x = y$ which does not have solutions over $\mathcal{N}$ but it has one solution over $\mathcal{Q}$, namely $\frac{50}{3}$. Hence, we take 16 as the reference point.
2. The following step considers collecting the constraints and constructing the adornments. The set of constraints C is defined as the union of the following three sets of inequalities:
   - set of $\tilde{x} \leq \tilde{x_0}$ for every reference point of the form $\tilde{x_0}$;
   - projection of $c(\tilde{x}, \tilde{y})$ on $\tilde{x}$, *i.e.*, $c'(\tilde{x})$ such that $c(\tilde{x}, \tilde{y}) \models c'(\tilde{x})$ and for every $c''(\tilde{x})$, if $c(\tilde{x}, \tilde{y}) \models c''(\tilde{x})$ then $c'(\tilde{x}) \models c''(\tilde{x})$;
   - if the domain is $\mathcal{N}$, $\mathcal{Q}^+$ or $\mathcal{R}^+$, inequalities of the form $\tilde{x} \geq 0$.

   In our case these sets are $\{X \leq 16\}$, $\{X \leq 25\}$, $\{X \geq 0\}$, respectively. In order to compute the set of adornments $\mathcal{A}_q$ we take all possible conjunctions of the elements of C and their negations. For the running example after simplifying the conjunctions, removing inconsistencies with respect to $\mathcal{N}$ and replacing strict inequalities with the non-strict ones, we obtain $\{0 \leq X \leq 16, 17 \leq X \leq 25, X \geq 26\}$. For the sake of simplicity we denote elements of this set $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. In general, if a number of elements in C is k, the maximal number of adornments is 2^k. Note that k is expected to be small, so the size of the set of adornments should not be problematic in practise.
3. Steps 3-8 have been inspired by the technique we used for numerical computations. Hence, here we present the steps briefly and refer to [16] for further details and proofs. For each binary clause r in P add $\bigvee_{c \in \mathcal{A}_p} c(\tilde{x})$ before a call $p(\tilde{x})$ in the body of r. By the construction above the disjunction is *true*, thus,

the transformed program is equivalent to the original one. In our case, the following program is obtained:

$$q(X) \leftarrow X + X + Y = 50, (0 \leq Y \leq 16 \vee 17 \leq Y \leq 25 \vee Y \geq 26), q(Y).$$

4. For each clause, such that the head of the clause, say $p(\tilde{x})$, has a recursive predicate p, add $\bigvee_{c \in \mathcal{A}_p} c(\tilde{x})$ as the first subgoal in its body. As for the previous step, the introduced call is equivalent to *true*, so that the transformation is obviously correct:

$$\begin{array}{l} q(X) \leftarrow (0 \leq X \leq 16 \vee 17 \leq X \leq 25 \vee X \geq 26), \\ \qquad X + X + Y = 50, (0 \leq Y \leq 16 \vee 17 \leq Y \leq 25 \vee Y \geq 26), q(Y). \end{array}$$

5. Next, moving to an alternative procedural interpretation of disjunction, for each clause in which we introduced a disjunction in one of the previous two steps, and for each such introduced disjunction we split these disjunctions, introducing a separate clause for each disjunct. For our running example we obtain 9 clauses. In general, every binary clause of the original program can produce $(2^k)^2$ adorned clauses. Observe that this transformation is correct for pure CLP programs but it is *not* correct for Prolog programs with non-logical features. For instance, in the presence of "cut", it may produce a different computed answer set.
   To prepare the next step in the transformation, note that, in the program resulting from step 5, for each rule r and for each recursive predicate p:
   – if a call $p(\tilde{x})$ occurs in r, it is immediately preceded by some adornment,
   – if an atom $p(\tilde{x})$ occurs as the head of r, it is immediately followed by some adornment.
   Moreover, since the elements $\mathcal{A}_p$ partition the domain, conjuncts like $c_i(\tilde{x})$, $p(\tilde{x})$ and $c_j(\tilde{x}), p(\tilde{x})$ for $i \neq j$, are mutually exclusive, as well as the analogous initial parts of the rules. This means that we can now safely rename the different cases apart.
6. Replace each occurrence of $c(\tilde{x}), p(\tilde{x})$ in the body of the clause with $c(\tilde{x})$, $p^c(\tilde{x})$ and each occurrence of a rule $p(\tilde{x}) \leftarrow c(\tilde{x}), Q$ with the corresponding rule $p^c(\tilde{x}) \leftarrow c(\tilde{x}), Q$. Because of the arguments presented above the LD-trees that exist for the given program and for the renamed program are identical, except for the names of the predicates and for a number of failing 1-step derivations (due to entering clauses that fail in their guard in the given program). As a result, both the semantics (up to renaming) and the termination behaviour of the program are preserved.
7. Remove all rules $p(\tilde{x}) \leftarrow c, \ldots$ with an inconsistent constraint c. We get:

$$\begin{array}{l} q^{\mathsf{a}}(X) \leftarrow 0 \leq X \leq 16, X + X + Y = 50, 17 \leq Y \leq 25, q^{\mathsf{b}}(Y). \\ q^{\mathsf{a}}(X) \leftarrow 0 \leq X \leq 16, X + X + Y = 50, 26 \leq Y, q^{\mathsf{c}}(Y). \\ q^{\mathsf{b}}(X) \leftarrow 17 \leq X \leq 25, X + X + Y = 50, 0 \leq Y \leq 16, q^{\mathsf{a}}(Y). \end{array}$$

which is the *adorned* program, P^a.

8. Next we need to prove termination of the adorned program with respect to the set of adorned queries $S^a = \{0 \le X \le 16, q^{\mathsf{a}}(X)\} \cup \{17 \le X \le 25, q^{\mathsf{b}}(X)\} \cup \{X \ge 26, q^{\mathsf{c}}(X)\}$. Observe that for every adornment c, $p^c(e_1,\ldots,e_n)$ is called in a computation of a query in S^a with respect to P^a if and only if $p(e_1,\ldots,e_n)$ is called in a computation of the corresponding query in S with respect to P, and c holds for $e_1,\ldots,e_n$. In our particular case termination can be proved by applying SVG. The following is one of the level mappings obtained.
$$|q^{\mathsf{a}}(x)|^{\mathsf{a}} = \begin{cases} 3x+50 & 0 \le x \le 16, \\ 0 & \textit{otherwise} \end{cases}$$
$$|q^{\mathsf{b}}(x)|^{\mathsf{b}} = \begin{cases} 150-3x, & 17 \le x \le 25, \\ 0 & \textit{otherwise} \end{cases}$$
$$|q^{\mathsf{c}}(x)|^{\mathsf{c}} = 0$$
9. Finally, we combine the linear level mappings found to obtain a piecewise linear level mapping. In our running example we can write the resulting level mapping as
$$|q(x)| = \begin{cases} 3x+50 & 0 \le x \le 16 \\ 150-3x & 17 \le x \le 25 \\ 0 & x \ge 26 \end{cases}$$

   Since this level-mapping exists we conclude termination of our CLP($\mathcal{N}$) program for all queries $c, q(X)$. This would not be the case if a different constraint domain such as $\mathcal{Q}$ have been considered.

   Correctness of this transformation follows from [16]:

Theorem 4. *Let P be a binary CLP program, S be a set of atomic CLP-queries, P^a and S^a the adorned program and the set of adorned queries, respectively. Then, all queries in S terminate with respect to P if and only if all queries in S^a terminate with respect to P^a.*

5 Conclusion

In this paper we have identified a class of CLP programs such that a linear level mapping is sufficient to prove their termination. We have seen that membership to this class is decidable and suggested a decision technique. We have further extended this class by considering tuples of linear functions. We have seen that membership to this class is also decidable. Finally, we have discussed piecewise level mappings.

The basic idea of identifying decidable and undecidable subsets of logic programs goes back to [4, 5, 14]. We generalise the class of programs considered to constraint logic programming (recall that logic programming can be seen as constraint logic programming over the domain of Herbrand terms). The restriction we pose is not syntactic. We have seen that llm-recurrency is a decidable condition sufficient for termination for all the domains considered. This condition can be automatically verified by cTI [12].

The idea of using mappings to domains more general than the natural numbers originated in early works on termination analysis [6, 8]. Tuple llm recurrency condition can be seen as a particular instance of this framework. Using tuples has been motivated by [3, 10] that do not compare sizes of atoms but sizes of arguments of these atoms. In [3, 10] a local approach to termination has been suggested, *i.e.* termination proof was based on a (locally verified) property of the computation abstraction. We follow a global approach to termination, *i.e.*, require the existence of a function (level mapping) decreasing along all possible computation paths. A related technique of using two level-mappings has been recently investigated in [11]. The main difference is that Martin and King use the two level mappings separately for two different goals, *i.e.* proving decrease and boundedness, while we use a lexicographic combination of level mappings to achieve both goals at the same time.

The adornments method presented above has been first presented in context of the numerical computations [16] and in its turn is related to the previous work on splitting predicates [17]. This technique can be seen as a variant of multiple specialisation [18]. However, to the best of our knowledge none of the existing specialisation tools considered constraint logic programming.

A number of interesting questions are considered as future work. First of all, we would like to understand whether the adornments technique is complete for piecewise llm recurrent programs. On a more practical side, we would like to implement these two extensions in the termination analyser cTI [12] and evaluate our approach experimentally.

References

1. K. R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9(3/4):335–364, 1991.
2. M. Codish. TerminWeb. Collection of benchmarks available at: http://lvs.cs.bgu.ac.il/∼mcodish/suexec/terminweb/bin/terminweb.cgi?command=examples.
3. M. Codish and C. Taboch. A semantic basis for termination analysis of logic programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
4. D. De Schreye, K. Verschaetse, and M. Bruynooghe. A practical technique for detecting non-terminating queries for a restricted class of Horn clauses, using directed, weighted graphs. In D. H. Warren and P. Szeredi, editors, *Logic Programming, Proceedings of the Seventh International Conference*, pages 649–663. MIT Press, 1990.

5. P. Devienne, P. Lebègue, and J.-C. Routier. Halting problem of one binary horn clause is undecidable. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings.*, volume 665 of *Lecture Notes in Computer Science*, pages 48–57. Springer Verlag, 1993.
6. R. W. Floyd. Assigning meanings to programs. In J. Schwartz, editor, *Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Society, 1967. Proceedings of Symposium in Applied Mathematics; v. 19.
7. J. Jaffar and M. J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–582, May/July 1994.
8. S. Katz and Z. Manna. A closer look at termination. *Acta Informatica*, 5:333–352, 1975.
9. V. Lagoon, F. Mesnard, and P. J. Stuckey. Termination analysis with types is more accurate. In C. Palamidessi, editor, *Logic Programming, 19th International Conference on Logic Programming*, pages 254–269. Springer Verlag, 2003.
10. N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In L. Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 63–77. MIT Press, July 1997.
11. J. C. Martin and A. King. On the inference of natural level mappings. In M. Bruynooghe and K.-K. Lau, editors, *Program Development in Computational Logic*, volume 3049 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.
12. F. Mesnard and U. Neumerkel. Applying static analysis techniques for inferring termination conditions of logic programs. In P. Cousot, editor, *Static Analysis, 8th International Symposium, SAS 2001*, volume 2126 of *Lecture Notes in Computer Science*, pages 93–110. Springer Verlag, 2001.
13. F. Mesnard and S. Ruggieri. On proving left termination of constraint logic programs. *ACM Transaction on Computational Logic*, 4(2):207–259, 2003.
14. S. Ruggieri. Decidability of logic program semantics and applications to testing. *Journal of Logic Programming*, 46(1–2):103–137, November/December 2000.
15. A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.
16. A. Serebrenik and D. De Schreye. Inference of termination conditions for numerical loops in Prolog. *Theory and Practice of Logic Programming*, 2004. to appear.
17. K. Sohn and A. Van Gelder. Termination detection in logic programs using argument sizes. In *Proceedings of the Tenth ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems*, pages 216–226. ACM Press, 1991.
18. W. Winsborough. Multiple specialization using minimal-function graph semantics. *Journal of Logic Programming*, 13(2/3):259–290, 1992.