TLP **9** (2): 145–164, 2009.
doi:10.1017/S1471068409003652 First published online 18 February 2009 Printed in the United Kingdom

A non-termination criterion for binary constraint logic programs

ÉTIENNE PAYET and FRED MESNARD
IREMIA - LIM - Université de la Réunion, France
(*e-mail:* {epayet,fred}@univ-reunion.fr)

submitted 9 July 2007; revised 6 August 2008; accepted 9 January 2009

Abstract

On the one hand, termination analysis of logic programs is now a fairly established research topic within the logic programming community. On the other hand, non-termination analysis seems to remain a much less attractive subject. If we divide this line of research into two kinds of approaches, dynamic versus static analysis, this paper belongs to the latter. It proposes a criterion for detecting non-terminating atomic queries with respect to binary constraint logic programming (CLP) rules, which strictly generalizes our previous works on this subject. We give a generic operational definition and an implemented logical form of this criterion. Then we show that the logical form is correct and complete with respect to the operational definition.

KEYWORDS: constraints, constraint logic programming, non-termination

1 Introduction

On the one hand, termination analysis of logic programs is a fairly established research topic within the logic programming community; see the surveys, for example, (De Schreye and Decorte 1994; Mesnard and Ruggieri 2003). Various termination analyzers are now available via web interfaces, and we note that the Mercury compiler, designed with industrial goals in mind, includes a termination analysis (Speirs *et al*. 1997) available as a compiler option.

On the other hand, non-termination analysis in logic programming seems to remain a much less attractive subject. We can divide this line of research into two kinds of approaches: dynamic versus static analysis. In the former one, Bol *et al.* (1991) sets up some solid foundations for loop checking, while some recent work is presented in Shen *et al.* (2001). The main idea is to *prune* infinite derivations *at runtime*. (Some finite derivations may also be pruned by some loop checkers.) In the latter approach, which includes the work we present in this paper, one tries to *compute at compile time* queries which admit at least one infinite derivation. One of the earliest works on the static approach is described in De Schreye *et al.* (1989), where the authors presented an algorithm for detecting non-terminating

atomic queries with respect to (w.r.t.) a binary clause of the form $p(\tilde{s}) \leftarrow p(\tilde{t})$. The condition is described in terms of rational trees, while we aim at generalizing non-termination analysis for the generic CLP(X) framework. Non-termination has also been studied in other paradigms, such as term rewrite systems (Waldmann 2004; Giesl *et al*. 2005; Zantema 2005; Waldmann 2007; Zankl and Middeldorp 2007; Payet 2008); the technique described in Payet (2008) is close to that of this paper. In Gupta *et al.* (2008), the authors consider non-termination of C programs, and Godefroid *et al.* (2005) and Sen *et al.* (2005) provide some techniques that detect crashes, assertion violation and non-termination in C programs.

Our analysis shares with the work on termination analysis presented in Codish and Taboch (1999) a key component: the binary unfoldings of a logic program (Gabbrielli and Giacobazzi 1994), which transform a finite set of definite clauses into a possibly infinite set offacts and binary definite clauses. Some termination analyses compute a finite over-approximation of the binary unfolding semantics, over a constraint domain such as CLP($\mathcal{N}$). In contrast, the non-termination analysis we have presented in Payet and Mesnard (2006) starts from a finite subset BP of the binary unfoldings of the concrete program P; of course, a larger subset may increase the precision of the analysis – Payet and Mesnard (2006) provide some experimental evidence. This non-termination analysis first detects patterns of non-terminating atomic queries from the binary recursive clauses and then propagates this non-termination information to compute classes of atomic queries for which we have a finite proof that there exists at least one infinite derivation w.r.t. BP. The equivalence between the termination of a logic program and that of its binary unfoldings (Codish and Taboch 1999) is the cornerstone of the analysis; it allows us to conclude that any atomic query belonging to the identified above classes admits an infinite left derivation w.r.t. P. The basic idea in Payet and Mesnard (2006) relies on checking, for each recursive clause in BP, that the body is more general than the head; if this test succeeds, we can conclude that the head is an atomic query which has an infinite derivation w.r.t. BP. A key observation consists in considering *neutral argument positions*, i.e. argument positions of the predicate symbols defined in P that do not have any effect on the derivation process when they are filled with a term that satisfies a given condition. The subsumption test presented in Payet and Mesnard (2006) only considers the arguments that are in the non-neutral positions and checks that the arguments in the neutral positions satisfy their associated condition. This extension of the classical subsumption test considerably increases the power of the approach in the sense that it allows one to compute more classes of non-terminating atomic queries.

The initial motivation in Payet and Mesnard (2006) was to complement terminationanalysis with non-termination inside the logic programming paradigm in order to detect optimal termination conditions expressed in a language describing classes of queries. Although we obtained interesting experimental results, the overall approach remains quite syntactic, with an *ad hoc* flavor and tight links to some basic logic programming machinery such as the unification algorithm. So in the present paper our aim is to generalize the approach to the constraint logic programming (CLP) setting, and the main contribution of this work consists in a strict generalization of the logical criterion defined in Payet and Mesnard (2004).

The paper is organized as follows: In Section 2 we give some preliminary definitions, and in Section 3 we recall in CLP terms the subsumption test to detect looping queries. In Section 4 we introduce the neutral argument positions; the operational definition we give (Section 4.3) is useless in practice; hence we propose a sufficient condition for neutrality, expressed as a logical formula related to the constraint binary clause under consideration (Section 4.4). For some constraint domains, we show that the condition is also necessary (Section 4.5). Depending on the constraint theory, the validity of such a condition can be automatically decided. In Section 4.6, we describe an algorithm that uses the logical formula of the sufficient condition to compute neutral argument positions. Finally, in Section 5 we describe our prototype, and we conclude the paper in Section 6. The detailed proofs of the results can be found in the long version of this paper which is available as a CoRR[1] archive.

Notice that our approach consists in computing a finite subset BP of the binary unfoldings of the program of interest and then in inferring non-terminating queries using BP only; hence, we deliberately choose to *restrict* the analysis to *binary CLP rules* and *atomic CLP queries*, as the result we obtain can be *lifted* to *full CLP*.

2 Preliminaries

For any non-negative integer n, $[1,n]$ denotes the set $\{1,\ldots,n\}$. If $n = 0$, then $[1,n] = \emptyset$. We recall some basic definitions of CLP; see Jaffar *et al.* (1998) for more details. From now on, we fix an infinite countable set $\mathscr{V}$ of *variables* together with a *signature* Σ, i.e. a pair $\langle F, \Pi\rangle$, where F is a set of *function symbols* and Π is a set of *predicate symbols* with $F \cap \Pi = \emptyset$ and $(F \cup \Pi) \cap \mathscr{V} = \emptyset$. Every element of $F \cup \Pi$ has an *arity* which is the number of its arguments. We write $f/n \in F$ (resp. $p/n \in \Pi$) to denote that f (resp. p) is an element of F (resp. Π) whose arity is $n \geqslant 0$. A *constant symbol* is an element of F whose arity is 0.

A *term* is a variable, a constant symbol or an object of the form $f(t_1,\ldots,t_n)$ where $f/n \in F$, $n \geqslant 1$ and $t_1,\ldots,t_n$ are terms. An *atomic proposition* is an element $p/0$ of Π or an object of the form $p(t_1,\ldots,t_n)$, where $p/n \in \Pi$, $n \geqslant 1$ and $t_1,\ldots,t_n$ are terms. A first-order *formula* on Σ is built from atomic propositions in the usual way, using the logical connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$ and the quantifiers $\exists$ and $\forall$. If ϕ is a formula and $W := \{X_1,\ldots,X_n\}$ is a set of variables, then $\exists_W \phi$ (resp. $\forall_W \phi$) denotes the formula $\exists X_1 \ldots \exists X_n \phi$ (resp. $\forall X_1 \ldots \forall X_n \phi$). We let $\exists\phi$ (resp. $\forall\phi$) denote the existential (resp. universal) closure of ϕ.

We fix a Σ-*structure* $\mathscr{D}$, i.e. a pair $\langle D, [\cdot]\rangle$ which is an interpretation of the symbols in Σ. The set D is called the *domain* of $\mathscr{D}$; $[\cdot]$ maps each $f/0 \in F$ to an element of D, each $f/n \in F$ with $n \geqslant 1$ to a function $[f] : D^n \rightarrow D$, each $p/0 \in \Pi$ to an element of $\{0,1\}$ and each $p/n \in \Pi$ with $n \geqslant 1$ to a boolean function $[p] : D^n \rightarrow \{0,1\}$. We assume that the predicate symbol $=$ is in Σ and is interpreted as identity in D. A *valuation* is a mapping from $\mathscr{V}$ to D. Each valuation v extends by morphism to

[1] http://arxiv.org/ – Paper ID is `0807.3451` .

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

terms. As usual, a valuation v induces a valuation $[\cdot]_v$ of terms to D and of formulas to $\{0,1\}$.

Given a formula ϕ and a valuation v, we write $\mathscr{D} \models_v \phi$ when $[\phi]_v = 1$. We write $\mathscr{D} \models \phi$ when $\mathscr{D} \models_v \phi$ for all valuation v. Notice that $\mathscr{D} \models \forall\phi$ if and only if $\mathscr{D} \models \phi$, that $\mathscr{D} \models \exists\phi$ if and only if there exists a valuation v such that $\mathscr{D} \models_v \phi$ and that $\mathscr{D} \models \neg\exists\phi$ if and only if $\mathscr{D} \models \neg\phi$. We say that a formula ϕ is *satisfiable* (resp. *unsatisfiable*) in $\mathscr{D}$ when $\mathscr{D} \models \exists\phi$ (resp. $\mathscr{D} \models \neg\phi$).

We fix a set $\mathscr{L}$ of admitted formulas, the elements of which are called *constraints*. We suppose that $\mathscr{L}$ is closed under variable renaming, existential quantification and conjunction and that it contains all the atomic propositions,the always satisfiable formula *true* and the unsatisfiable formula *false*. We assume that there is a computable function *solv* which maps each $c \in \mathscr{L}$ to one of `true` or `false`, indicating whether c is satisfiable or unsatisfiable in $\mathscr{D}$. We call *solv* the *constraint solver*.

Example 2.1 ($\mathscr{Q}_{\text{lin}}$)

The constraint domain $\mathscr{Q}_{\text{lin}}$ has $<$, $\leqslant$, $=$, $\geqslant$, $>$ as predicate symbols, $+$, $-$, $*$, $/$ as function symbols and sequences of digits as constant symbols. Only linear constraints are admitted. The domain of computation is the structure with the set of rationals, denoted by $\mathbb{Q}$, as domain and where the predicate symbols and the function symbols are interpreted as the usual relations and functions over the rationals. A constraint solver for $\mathscr{Q}_{\text{lin}}$ always returning either `true` or `false` is described in Refalo and Hentenryck (1996).

Sequences of distinct variables are denoted by $\tilde{X}$, $\tilde{Y}$ or $\tilde{Z}$ and are sometimes considered as sets of variables: we may write $\forall_{\tilde{X}}$, $\exists_{\tilde{X}}$ or $\tilde{X} \cup \tilde{Y}$. Sequences of (not necessarily distinct) terms are denoted by $\tilde{s}$, $\tilde{t}$ or $\tilde{u}$. Given two sequences of n terms $\tilde{s} := (s_1, \ldots, s_n)$ and $\tilde{t} := (t_1, \ldots, t_n)$, we write $\tilde{s} = \tilde{t}$ either to denote the constraint $s_1 = t_1 \wedge \cdots \wedge s_n = t_n$ or as a shorthand for "$s_1 = t_1$ and ... and $s_n = t_n$". Given a valuation v, we write $v(\tilde{s})$ to denote the sequence $(v(s_1), \ldots, v(s_n))$ and $[\tilde{s}]_v$ to denote the sequence $([s_1]_v, \ldots, [s_n]_v)$.

The signature in which all programs and queries under consideration are includedis $\Sigma_L := \langle F, \Pi \cup \Pi' \rangle$ where Π' is the set of predicate symbols that can be defined in programs, with $\Pi \cap \Pi' = \varnothing$.

An *atom* has the form $p(t_1, \ldots, t_n)$, where $p/n \in \Pi'$ and $t_1, \ldots, t_n$ are terms. A *program* is a finite set of clauses. A *clause* has the form $H \leftarrow c \diamond B$, where H and B are atoms and c is a finite conjunction of atomic propositions such that $\mathscr{D} \models \exists c$. A *query* has the form $\langle A \mid d \rangle$, where A is an atom and d is a finite conjunction of atomic propositions. Given an atom $A := p(\tilde{t})$, we write $rel(A)$ to denote the predicate symbol p. Given a query $Q := \langle A \mid d \rangle$, we write $rel(Q)$ to denote the predicate symbol $rel(A)$. The set of variables occurring in some syntactic objects $O_1, \ldots, O_n$ is denoted $\text{Var}(O_1, \ldots, O_n)$.

We consider the following operational semantics given in terms of *derivations* from queries to queries: Let $\langle p(\tilde{u}) \mid d \rangle$ be a query and $p(\tilde{s}) \leftarrow c \diamond q(\tilde{t})$ be a fresh copy

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

of a clause r. When $solv(\tilde{s} = \tilde{u} \wedge c \wedge d) = \texttt{true}$,

$$\langle p(\tilde{u}) \mid d \rangle \underset{r}{\Longrightarrow} \langle q(\tilde{t}) \mid \tilde{s} = \tilde{u} \wedge c \wedge d \rangle$$

is a *derivation step* of $\langle p(\tilde{u}) \mid d \rangle$ w.r.t. r with $p(\tilde{s}) \leftarrow c \diamond q(\tilde{t})$ as its *input clause*. We write $Q \overset{+}{\underset{P}{\Longrightarrow}} Q'$ to summarize a finite number (> 0) of derivation steps from Q to Q', where each input clause is a variant of a clause from program P. Let Q_0 be a query. A sequence of derivation steps $Q_0 \underset{r_1}{\Longrightarrow} Q_1 \underset{r_2}{\Longrightarrow} \cdots$ of maximal length is called a *derivation* of $P \cup \{Q_0\}$ when $r_1, r_2, \ldots$ are clauses from P and when the *standardization apart* condition holds; i.e. each input clause used is variable disjoint from the initial query Q_0 and from the input clauses used at earlier steps. We say Q_0 *loops* w.r.t. P when there exists an infinite derivation of $P \cup \{Q_0\}$.

3 Loop inference with constraints

In the logic programming framework, the subsumption test provides a simple way to infer looping queries: if, in a logic program P, there is a clause $p(\tilde{s}) \leftarrow p(\tilde{t})$ such that $p(\tilde{t})$ is more general than $p(\tilde{s})$, then the query $p(\tilde{s})$ loops w.r.t. P. In this section, we extend this result to the constraint logic programming framework.

3.1 A "more general than" relation

A query can be viewed as a finite description of a possibly infinite set of atoms, the arguments of which are values from D.

Example 3.1
In the constraint domain $\mathcal{Q}_{\text{lin}}$, the query $Q := \langle p(X, Y) \mid Y \leqslant X + 2 \rangle$ describes the set of atoms $p(x, y)$, where x and y are rational numbers and X and Y can be made equal to x and y respectively while the constraint $Y \leqslant X + 2$ is satisfied. For instance, $p(0, 2)$ is an element of the set described by Q.

In order to capture this intuition, we introduce the following definition.

Definition 3.2 (*Set described by a query*)
The set of atoms that is described by a query $Q := \langle p(\tilde{t}) \mid d \rangle$ is denoted by $Set(Q)$ and is defined as $Set(Q) = \{p([\tilde{t}]_v) \mid \mathcal{D} \models_v d\}$.

Clearly, $Set(\langle p(\tilde{t}) \mid d \rangle) = \emptyset$ if and only if d is unsatisfiable in $\mathcal{D}$. Moreover, two variants describe the same set:

Lemma 3.3
Let Q and Q' be two queries such that Q' is a variant of Q. Then, $Set(Q) = Set(Q')$.

Notice that the operational semantics we introduced above can be expressed using sets described by queries:

Lemma 3.4
Let Q be a query and $r := H \leftarrow c \diamond B$ be a clause. There exists a derivation step of Q w.r.t. r if and only if $Set(Q) \cap Set(\langle H \mid c \rangle) \neq \emptyset$.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

The "more general than" relation we consider is defined as follows:

Definition 3.5 (*More General*)
We say that a query Q_1 is *more general than* a query Q when $Set(Q) \subseteq Set(Q_1)$.

Example 3.6
In $\mathcal{Q}_{\text{lin}}$, the query $Q_1 := \langle p(X,Y) \mid Y \leqslant X + 3 \rangle$ is more general than the query $Q := \langle p(X,Y) \mid Y \leqslant X + 2 \rangle$. However, Q is not more general than Q_1; for instance, $p(0,3) \in Set(Q_1)$ but $p(0,3) \notin Set(Q)$.

3.2 Loop inference

Suppose we have a derivation step $Q \underset{r}{\Longrightarrow} Q_1$, where $r := H \leftarrow c \diamond B$. Then, by Lemma 3.4, $Set(Q) \cap Set(\langle H \mid c \rangle) \neq \emptyset$. Hence, if Q' is a query that is more general than Q, as $Set(Q) \subseteq Set(Q')$, we have $Set(Q') \cap Set(\langle H \mid c \rangle) \neq \emptyset$. So, by Lemma 3.4, there exists a query Q'_1 such that $Q' \underset{r}{\Longrightarrow} Q'_1$. The following lifting result says that, moreover, Q'_1 is more general than Q_1.

Theorem 3.7 (*Lifting*)
Consider a derivation step $Q \underset{r}{\Longrightarrow} Q_1$ and a query Q' that is more general than Q. Then, there exists a derivation step $Q' \underset{r}{\Longrightarrow} Q'_1$, where Q'_1 is more general than Q_1.

From this theorem, we derive two corollaries that can be used to infer looping queries just from the text of a program.

Corollary 3.8
Let $r := H \leftarrow c \diamond B$ be a clause. If $\langle B \mid c \rangle$ is more general than $\langle H \mid c \rangle$, then $\langle H \mid c \rangle$ loops w.r.t. $\{r\}$.

The intuition of Corollary 3.8 is that we have $\langle H \mid c \rangle \underset{r}{\Longrightarrow} Q_1$, where Q_1 is a variant of $\langle B \mid c \rangle$; hence, Q_1 is more general than $\langle H \mid c \rangle$; so, by the Lifting Theorem 3.7, there exists a derivation step $Q_1 \underset{r}{\Longrightarrow} Q_2$, where Q_2 is more general than Q_1; by repeatedly using this reasoning, one can build an infinite derivation of $\{r\} \cup \{\langle H \mid c \rangle\}$.

Corollary 3.9
Let $r := H \leftarrow c \diamond B$ be a clause from a program P. If $\langle B \mid c \rangle$ loops w.r.t. P, then $\langle H \mid c \rangle$ loops w.r.t. P.

The intuition of Corollary 3.9 is that we have $\langle H \mid c \rangle \underset{r}{\Longrightarrow} Q_1$, where Q_1 is a variant of $\langle B \mid c \rangle$, which implies that Q_1 is more general than $\langle B \mid c \rangle$; as there exists an infinite derivation ξ of $P \cup \{\langle B \mid c \rangle\}$, by successively applying the Lifting Theorem 3.7 to each step of ξ one can construct an infinite derivation of $P \cup \{Q_1\}$.

Example 3.10
Consider the following recursive clause r in $\mathcal{Q}_{\text{lin}}$:

$$p(N) \leftarrow N \geqslant 1 \wedge N = N_1 + 1 \ \diamond p(N_1).$$

The query $Q_1 := \langle p(N_1) \mid N \geqslant 1 \wedge N = N_1 + 1 \rangle$ is more general than the query $Q := \langle p(N) \mid N \geqslant 1 \wedge N = N_1 + 1 \rangle$ (for instance, $p(0) \in Set(Q_1)$ but $p(0) \notin Set(Q)$).

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

So, by Corollary 3.8, Q loops w.r.t. $\{r\}$. Therefore, there exists an infinite derivation ξ of $\{r\} \cup \{Q\}$. Then, if Q' is a query that is more general than Q, by successively applying the Lifting Theorem 3.7 to each step of ξ, one can construct an infinite derivation of $\{r\} \cup \{Q'\}$. So, Q' also loops w.r.t. $\{r\}$.

4 Loop inference using filters

The condition provided by Corollary 3.8 is rather weak because it fails at inferring looping queries in some simple cases. This is illustrated by the following example.

Example 4.1
Consider the following recursive clause r in $\mathscr{Q}_{\text{lin}}$:

$$p(N, T) \leftarrow N \geqslant 1 \wedge N = N_1 + 1 \wedge T_1 = 2 * T \wedge T \geqslant 1 \ \diamond \ p(N_1, T_1)$$

Let c denote the constraint in r. The query $\langle p(N, T) \mid c \rangle$ loops w.r.t. $\{r\}$ because only the first argument of p decreases in r, and in this query it is unspecified. But we cannot infer that $\langle p(N, T) \mid c \rangle$ loops w.r.t. $\{r\}$ from Corollary 3.8, as in r $\langle p(N_1, T_1) \mid c \rangle$ is not more general than $\langle p(N, T) \mid c \rangle$ because of the second argument of p: for instance, $p(1, 1) \in Set(\langle p(N, T) \mid c \rangle)$ but $p(1, 1) \notin Set(\langle p(N_1, T_1) \mid c \rangle)$.

In what follows, we extend the relation "is more general." Instead of comparing atoms in all positions using the "more general" relation, we distinguish some predicate argument positions for which we just require that a certain property must hold, while for the other positions we use the "more general" relation as before. Doing so, we aim at inferring more looping queries.

Example 4.2 (*Example 4.1 continued*)
Let us consider argument position 2 of predicate symbol p. In the clause r, the projection of c on T is equivalent to $T \geqslant 1$; this projection expresses the constraint placed upon the second argument of p to get a derivation step with r. Notice that the projection of c on T_1 is equivalent to $T_1 \geqslant 2$, which implies $T_1 \geqslant 1$. Therefore, the requirements on the head variable T propagates to the body variable T_1. Moreover, the "piece" $\langle p(N_1) \mid c \rangle$ of $\langle p(N_1, T_1) \mid c \rangle$ is more general than the "piece" $\langle p(N) \mid c \rangle$ of $\langle p(N, T) \mid c \rangle$. Consequently, $\langle p(N_1, T_1) \mid c \rangle$ is more general than $\langle p(N, T) \mid c \rangle$ up to the second argument of p which, in $\langle p(N_1, T_1) \mid c \rangle$, satisfies $T_1 \geqslant 1$, the condition to get a derivation step with r. Hence, by an extended version of Corollary 3.8 we could infer that $\langle p(N, T) \mid c \rangle$ loops w.r.t. $\{r\}$.

4.1 Sets of positions

A basic idea in Example 4.2 lies in identifying argument positions of predicate symbols. Below, we introduce a formalism to do so.

Definition 4.3 (*Set of positions*)
A *set of positions*, denoted by τ, is a function that maps each $p/n \in \Pi'$ to a subset of $[1, n]$.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Example 4.4
If we want to distinguish the second argument position of the predicate symbol p defined in Example 4.1, we set $\tau := \langle p \mapsto \{2\}\rangle$. If we do not want to distinguish any argument position of p, we set $\tau' := \langle p \mapsto \emptyset\rangle$.

Definition 4.5
Let τ be a set of positions. Then, $\overline{\tau}$ is the set of positions whose definition is that for each $p/n \in \Pi'$, $\overline{\tau}(p) = [1, n] \setminus \tau(p)$.

Example 4.6
If we set $\tau := \langle p \mapsto \{2\}\rangle$ and $\tau' := \langle p \mapsto \emptyset\rangle$, where the arity of p is 2, then $\overline{\tau} = \langle p \mapsto \{1\}\rangle$ and $\overline{\tau}' = \langle p \mapsto \{1, 2\}\rangle$.

Using a set of positions τ, one can *project* syntactic objects:

Definition 4.7 (*Projection*)
Let τ be a set of positions.

- The projection of $p \in \Pi'$ on τ is the predicate symbol denoted by p_τ. Its arity is the number of elements of $\tau(p)$.
- Let $p/n \in \Pi'$ and $\tilde{t} := (t_1, \ldots, t_n)$ be a sequence of n terms. The *projection of* $\tilde{t}$ *on* $\tau(p)$, denoted by $\tilde{t}_{\tau(p)}$, is the sequence $(t_{i_1}, \ldots, t_{i_m})$, where $\{i_1, \ldots, i_m\} = \tau(p)$ and $i_1 < \cdots < i_m$.
- Let $A := p(\tilde{t})$ be an atom. The projection of A on τ, denoted by A_τ, is the atom $p_\tau(\tilde{t}_{\tau(p)})$.
- The projection of a query $\langle A \mid d\rangle$ on τ, denoted by $\langle A \mid d\rangle_\tau$, is the query $\langle A_\tau \mid d\rangle$.

Example 4.8 (*Example 4.4 continued*)
The projection of the query $\langle p(N, T) \mid c\rangle$ on τ (resp. τ') is the query $\langle p_\tau(T) \mid c\rangle$ (resp. the query $\langle p_{\tau'} \mid c\rangle$).

Projection preserves inclusion and non-disjointness of sets described by queries:

Lemma 4.9 (*Inclusion*)
Let τ be a set of positions and Q and Q' be two queries. If $Set(Q) \subseteq Set(Q')$, then $Set(Q_\tau) \subseteq Set(Q'_\tau)$.

Lemma 4.10 (*Non-disjointness*)
Let τ be a set of positions and Q and Q' be two queries. If $Set(Q) \cap Set(Q') \neq \emptyset$, then $Set(Q_\tau) \cap Set(Q'_\tau) \neq \emptyset$.

4.2 Filters

A second idea in Example 4.2 consists in associating constraints with argument positions ($T \geqslant 1$ for position 2 in Example 4.2). We define a filter to be the combination of sets of positions with their associated constraint:

Definition 4.11 (*Filter*)
A *filter*, denoted by Δ, is a pair (τ, δ), where τ is a set of positions and δ is a function that maps each $p \in \Pi'$ to a query of the form $\langle p_\tau(\tilde{t}) \mid d\rangle$, where $\mathscr{D} \models \exists d$.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Example 4.12
Consider $\tau := \langle p \mapsto \{2\}\rangle$ and $\tau' := \langle p \mapsto \emptyset\rangle$. Let $\delta := \langle\ p \mapsto \langle p_\tau(B) \mid B \geqslant 1\rangle\ \rangle$ and $\delta' := \langle\ p \mapsto \langle p_{\tau'} \mid true\rangle\ \rangle$. Then, $\Delta := (\tau, \delta)$ and $\Delta' := (\tau', \delta')$ are filters.

Note that $\delta(p)$ is given in the form of a query $\langle p_\tau(\tilde{t}) \mid d\rangle$, instead of just a constraint d, because we need to indicate that the entry points of d are the terms in $\tilde{t}$. Indeed, the function δ is used to "filter" queries: we say that a query Q *satisfies* Δ when the set of atoms described by Q_τ, the projection of Q on the positions τ, is included in the set of atoms described by $\delta(rel(Q))$, the query defined for Q's predicate symbol by Δ. More formally:

Definition 4.13 (*Satisfies*)
Let $\Delta := (\tau, \delta)$ be a filter and Q be a query. Let $p := rel(Q)$. We say that Q *satisfies* Δ when $Set(Q_\tau) \subseteq Set(\delta(p))$.

Now we come to the extension of the relation "more general than." Intuitively, $\langle p(\tilde{t'}) \mid d'\rangle$ is Δ-more general than $\langle p(\tilde{t}) \mid d\rangle$ if the "more general than" relation holds for the elements of $\tilde{t}$ and $\tilde{t'}$ whose position is not in τ, while the elements of $\tilde{t'}$ whose position is in τ satisfy δ. More formally:

Definition 4.14 (Δ-*more general*)
Let $\Delta := (\tau, \delta)$ be a filter and Q and Q' be two queries. We say that Q' is Δ-*more general than* Q when $Q'_{\bar{\tau}}$ is more general than $Q_{\bar{\tau}}$ and Q' satisfies Δ.

Example 4.15
Consider the constraint c in the clause

$$p(N, T) \leftarrow N \geqslant 1 \wedge N = N_1 + 1 \wedge T_1 = 2 * T \wedge T \geqslant 1 \diamond p(N_1, T_1)$$

of Example 4.1. The query $Q_1 := \langle p(N_1, T_1) \mid c\rangle$ is Δ-more general than $Q := \langle p(N, T) \mid c\rangle$ for the filter $\Delta := \big(\langle p \mapsto \{2\}\rangle, \langle p \mapsto \langle p_\tau(B) \mid B \geqslant 1\rangle\rangle\big)$. However, Q_1 is not Δ'-more general than Q for the filter $\Delta' := \big(\langle p \mapsto \emptyset\rangle, \langle p \mapsto \langle p_{\tau'} \mid true\rangle\rangle\big)$; indeed, $\tau'(p) = \emptyset$ implies that being Δ'-more general is equivalent to being more general, and by Example 4.1, Q_1 is not more general than Q.

Lemma 4.16 (*Transitivity*)
For any filter Δ, the "Δ-more general than" relation is transitive.

Notice that for any filter $\Delta := (\tau, \delta)$ and any query Q, we have that $Q_{\bar{\tau}}$ is more general than itself (because the "more general than" relation is reflexive), but Q may not satisfy Δ. Hence, the "Δ-more general than" relation is not always reflexive.

Example 4.17
Consider the constraint domain $\mathcal{Q}_{\text{lin}}$. Let $p/1 \in \Pi'$ and $\Delta := (\tau, \delta)$ be the filter defined by $\tau := \langle p \mapsto \{1\}\rangle$ and $\delta := \langle\ p \mapsto \langle p_\tau(X) \mid X \geqslant 1\rangle\ \rangle$. The query $Q := \langle p(0) \mid true\rangle$ is not Δ-more general than itself because $Set(Q_\tau) = \{p_\tau(0)\} \notin \{p_\tau(x) \mid x \text{ is a rational and } x \geqslant 1\} = Set(\delta(p))$. Hence, Q does not satisfy Δ.

The fact that reflexivity does not always hold is an expected property. Indeed, suppose that a filter $\Delta := (\tau, \delta)$ induces a "Δ-more general than" relation that is

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

reflexive. Then for any queries Q and Q', we have that Q' is Δ-more general than Q if and only if $Q'_{\bar{\tau}}$ is more general than $Q_{\bar{\tau}}$ (because, as Q' is Δ-more general than itself, Q' necessarily satisfies Δ). Hence, δ is useless in the sense that it "does not filter anything." Filters equipped with such a δ were introduced in Payet and Mesnard (2004), where for any predicate symbol p, $\delta(p)$ has the form $\langle p_\tau(\tilde{X}) \mid true \rangle$, where $\tilde{X}$ is a sequence of distinct variables. In this paper, we aim at generalizing the approach of Payet and Mesnard (2004). Hence, we also consider functions δ that really filter queries.

4.3 DN filters: an operational definition

Let us now introduce a special kind of filters that we call "derivation neutral" (DN). The name "derivation neutral" stems from the fact that if in a derivation of a query Q we replace Q by a Δ-more general Q', then we get a "similar" derivation.

Definition 4.18 (*Derivation neutral*)
Let r be a clause and Δ be a filter. We say that Δ is *DN* for r when for each derivation step $Q \underset{r}{\Longrightarrow} Q_1$, the query Q_1 satisfies Δ and when for each query Q' that is Δ-more general than Q, there exists a derivation step $Q' \underset{r}{\Longrightarrow} Q'_1$, where Q'_1 is Δ-more general than Q_1. This definition is extended to programs: Δ is *DN* for P when it is DN for each clause of P.

DN filters lead to the following extended version of Corollary 3.8 (to get Corollary 3.8, take $\Delta := (\tau, \delta)$ with $\tau(p) = \varnothing$ for any p):

Theorem 4.19
Let $r := H \leftarrow c \diamond B$ be a clause. Let Δ be a filter that is DN for r. If $\langle B \mid c \rangle$ is Δ-more general than $\langle H \mid c \rangle$ then $\langle H \mid c \rangle$ loops w.r.t. $\{r\}$.

Example 4.20
If the filter Δ of Example 4.15 is DN for the clause $r = p(N, T) \leftarrow c \diamond p(N_1, T_1)$ of Example 4.1, then we can deduce that $\langle p(N, T) \mid c \rangle$ loops w.r.t. $\{r\}$ because $\langle p(N_1, T_1) \mid c \rangle$ is Δ-more general than $\langle p(N, T) \mid c \rangle$ (see Example 4.15).

Computing a DN filter from the text of a program is not straightforward if we use the above definition. Section 4.4 presents a logical characterization that we use in Section 4.6 to compute a filter that is DN for a given recursive clause.

4.4 A logical characterization of DN filters

From now on, we suppose, without loss of generality, that a clause has the form $p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$, where $\tilde{X}$ and $\tilde{Y}$ are disjoint sequences of distinct variables. Hence, c is the conjunction of all the constraints, including unifications. We distinguish the following set of variables that appear inside such a clause:

Definition 4.21
The set of *local variables* of a clause $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$ is $local\_vars(r) := \mathrm{Var}(c) \setminus (\tilde{X} \cup \tilde{Y})$.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

In this section, we aim at characterizing DN filters in a logical way. To this end, we define:

Definition 4.22 (*sat*)
Let $Q := \langle p(\tilde{t}) \mid d \rangle$ be a query and $\tilde{s}$ be a sequence of terms of the same length as $\tilde{t}$. Then, $sat(\tilde{s}, Q)$ denotes a formula of the form $\exists_{\mathrm{Var}(Q')}(\tilde{s} = \tilde{t}' \wedge d')$, where $Q' := \langle p(\tilde{t}') \mid d' \rangle$ is a variant of Q and variable disjoint with $\tilde{s}$.

Intuitively, $sat(\tilde{s}, Q)$ holds when the terms in the sequence $\tilde{s}$ satisfy the constraint d, the entry points of which are the terms in $\tilde{t}$. Clearly, the satisfiability of $sat(\tilde{s}, Q)$ does not depend on the choice of the variant of Q. The set that is described by a query can then be characterized as follows:

Lemma 4.23
Let Q be a query and $p := rel(Q)$. Let $\tilde{u}$ be a sequence of $arity(p)$ terms and v be a valuation. Then, $p([\tilde{u}]_v) \in Set(Q)$ if and only if $\mathscr{D} \models_v sat(\tilde{u}, Q)$.

Now we give a logical definition of derivation neutrality. As we will see later, under certain circumstances, this definition is equivalent to the operational one we gave above.

Definition 4.24 (*Logical derivation neutral*)
We say that a filter $\Delta := (\tau, \delta)$ is *DNlog* for a clause $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$ when

$$\mathscr{D} \models c \rightarrow \forall_{\tilde{X}_{\tau(p)}} \left[sat(\tilde{X}_{\tau(p)}, \delta(p)) \rightarrow \exists_{\mathscr{Y}} c \right] \quad \text{and} \quad \mathscr{D} \models c \rightarrow sat(\tilde{Y}_{\tau(q)}, \delta(q)),$$

where $\mathscr{Y} = \tilde{Y}_{\tau(q)} \cup local\_vars(r)$.

Example 4.25
In $\mathscr{Q}_{\mathrm{lin}}$, the filter $\big(\langle p \mapsto \{2\} \rangle, \langle p \mapsto \langle p_\tau(B) \mid B \geqslant 1 \rangle \rangle\big)$ is DNlog for the clause

$$p(N, T) \leftarrow N \geqslant 1 \wedge N = N_1 + 1 \wedge T_1 = 2 * T \wedge T \geqslant 1 \diamond p(N_1, T_1)$$

of Example 4.1. Indeed, $\tilde{X}_{\tau(p)} = \{T\}$, $\tilde{Y}_{\tau(q)} = \{T_1\}$ and $local\_vars(r) = \{\}$. So, if we let c denote the constraint in this clause, the formulas of Definition 4.24 turn into

$$\mathscr{D} \models c \rightarrow \forall T \left[T \geqslant 1 \rightarrow \exists T_1 \, c \right] \quad \text{and} \quad \mathscr{D} \models c \rightarrow T_1 \geqslant 1$$

which are true.

The first formula in Definition 4.24 has the following meaning: if one holds a solution for constraint c, then, changing the value given to the variables of $\tilde{X}$ distinguished by τ to some value satisfying $\delta(p)$, there exists a value for the local variables and the variables of $\tilde{Y}$ distinguished by τ such that c is still satisfied. This formula expresses the fact that DNlog arguments (i.e. those distinguished by τ) *do not interact* in c with the other arguments. Intuitively, two variables X_1 and X_2 do not interact in a constraint c when the set of values assigned to (X_1, X_2) by all the solutions of c results from the exhaustive combination of the set of values assigned to X_1 by all the solutions of c and the set of values assigned to X_2 by all the solutions of c; more formaly, when

$$\{(v(X_1), v(X_2)) \mid \mathscr{D} \models_v c\} = \{v(X_1) \mid \mathscr{D} \models_v c\} \times \{v(X_2) \mid \mathscr{D} \models_v c\} .$$

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Example 4.26

- In Example 4.25 above, the set of values assigned to (N, T) by all the solutions of c is $\{(a,b) \mid a \geqslant 1,\ b \geqslant 1\}$. We have $\{(a,b) \mid a \geqslant 1,\ b \geqslant 1\} = \{a \mid a \geqslant 1\} \times \{b \mid b \geqslant 1\}$, where $\{a \mid a \geqslant 1\}$ is the set of values assigned to N by all the solutions of c and $\{b \mid b \geqslant 1\}$ is the set of values assigned to T by all the solutions of c. Hence, N and T do not interact.
- Now consider $c = (X \geqslant Z \wedge Z \geqslant Y)$. The set of values assigned to (X, Y) by all the solutions of c is $\{(a,b) \mid a \geqslant b\}$ and the set of values assigned to X and to Y by all the solutions of c is $\mathbb{Q}$. As $\{(a,b) \mid a \geqslant b\} \neq \mathbb{Q} \times \mathbb{Q}$, we have that X and Y do interact.

The second formula in Definition 4.24 means that any solution of c assigns to the variables of $\tilde{Y}$ distinguished by τ a value that satisfies $\delta(q)$. This corresponds to the intuition that neutral argument positions are sorts of "pipes" in which one can place any term satisfying δ with no effect on the derivation process.

The logical definition of derivation neutrality implies the operational one (see also the discussion in the long version of this paper):

Theorem 4.27
Let r be a clause and Δ be a filter. If Δ is DNlog for r, then Δ is DN for r.

4.5 When DN filters are also DNlog

DN filters are not always DNlog as illustrated by the following example:

Example 4.28
Suppose that $\Sigma = \{0, =, \geqslant\}$ and $\mathscr{D} = \mathscr{D}_{\mathscr{Q}_{\text{lin}}}$. Consider

$$r := p(X_1, X_2) \leftarrow X_2 \geqslant X_1 \wedge X_1 \geqslant 0 \wedge Y_1 = X_1 \wedge Y_2 = X_2 \diamond p(Y_1, Y_2)\,.$$

Let c denote the constraint in r. Consider also a filter $\Delta := (\tau, \delta)$, where $\tau(p) = \{1\}$ and $\delta(p) = \langle p_\tau(X) \mid X \geqslant 0 \rangle$. Notice that given the form of Σ, one cannot write a constraint that has only one solution different from 0; more precisely, for any terms t_1 and t_2 and any constraint $d \neq \mathit{false}$,

$$p(0,0) \in \mathit{Set}(\langle p(t_1, t_2) \mid d \rangle)\,. \tag{1}$$

Whatever Q, if there is a derivation step $Q \underset{r}{\Longrightarrow} Q_1$:

- the query Q_1 satisfies Δ because c implies that $Y_1 \geqslant 0$;
- for any Q' that is Δ-more general than Q, $\mathit{Set}(\langle p(X_1, X_2) \mid c \rangle) \cap \mathit{Set}(Q') \neq \emptyset$ because by (1) $p(0,0) \in \mathit{Set}(\langle p(X_1, X_2) \mid c \rangle) \cap \mathit{Set}(Q')$; hence, there exists a derivation step $Q' \underset{r}{\Longrightarrow} Q'_1$. Notice that $Q'_{1\bar{\tau}}$ is more general than $Q_{1\bar{\tau}}$ because $Q'_{\bar{\tau}}$ is more general than $Q_{\bar{\tau}}$ and c demands that $Y_2 = X_2$; moreover, Q'_1 satisfies Δ because c implies that $Y_1 \geqslant 0$; therefore, Q'_1 is Δ-more general than Q_1.

Consequently, Δ is DN for r. However, Δ is not DNlog for r because the first formula of Definition 4.24 does not hold. Indeed, as $\tilde{X}_{\tau(p)} = X_1$, $\tilde{Y}_{\tau(p)} = Y_1$ and $\mathscr{Y} = \{Y_1\}$, this formula is equivalent to $\mathscr{D} \models c \rightarrow \forall X_1 [X_1 \geqslant 0 \rightarrow \exists Y_1 c]$. Let v be a valuation

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

such that $v(X_1) = v(Y_1) = v(X_2) = v(Y_2) = 0$; then, $\mathscr{D} \models_v c$. Let v_1 be a valuation with $v_1(X_1) = 1$, and v_1 matches v on the other variables; then, $\mathscr{D} \models_{v_1} X_1 \geqslant 0$; however, $\mathscr{D} \models_{v_1} \exists Y_1 c$ does not hold because c contains the constraint $X_2 \geqslant X_1$ with $v_1(X_2) = 0$ and $v_1(X_1) = 1$, and it is not possible to change the value that v_1 assigns to Y_1 so that $v_1(X_2) \geqslant v_1(X_1)$. Therefore, $\mathscr{D} \models_v c \rightarrow \forall X_1[X_1 \geqslant 0 \rightarrow \exists Y_1 c]$ does not hold.

The point in Example 4.28 is that the problematic values (for DNlog-ness) cannot be captured by a query; hence they do not prevent Δ from being DN. More precisely, we have $p\big(v(X_1), v(X_2)\big) = p\big(v(Y_1); v(Y_2)\big) = p(0,0)$; and the atom $p(0,0)$ is captured by the query $\langle p(0,0) \mid true\rangle$, i.e. $Set(\langle p(0,0) \mid true\rangle) = \{p(0,0)\}$. However, $p\big(v_1(X_1), v_1(X_2)\big) = p(1,0)$, and there exists no query Q with $Set(Q) = \{p(1,0)\}$. If we had considered r in the constraint domain $\mathscr{Q}_{\text{lin}}$, then Δ would not have been DN as there exists Q_1 such that $\langle p(0,0) \mid true\rangle \underset{r}{\Longrightarrow} Q_1$; the query $\langle p(1,0) \mid true\rangle$ is well formed in $\mathscr{Q}_{\text{lin}}$ and is Δ-more general than $\langle p(0,0) \mid true\rangle$,[2] but there exists no query Q_1' such that $\langle p(1,0) \mid true\rangle \underset{r}{\Longrightarrow} Q_1'$. Hence, an idea for matching DN with DNlog consists in considering domains in which every sequence of values can be captured by a query:

Theorem 4.29
If, for all atoms A whose arguments are elements of D, there exists a query Q such that $Set(Q) = \{A\}$, then every filter that is DN for a clause r is also DNlog for r.

The intuition of the proof of Theorem 4.29 consists in mapping some sequences of values (induced by the considered valuations) to queries that capture them and in using the DN property to prove that DNlog-ness holds. More precisely, let $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$ and $\Delta := (\tau, \delta)$ be a filter that is DN for r. First, we have to prove that

$$\mathscr{D} \models c \rightarrow \forall_{\tilde{X}_{\tau(p)}} \big[sat(\tilde{X}_{\tau(p)}, \delta(p)) \rightarrow \exists_{\mathscr{Y}} c\big] .$$

Let v be a valuation such that $\mathscr{D} \models_v c$ and v' be a valuation such that $v'(V) = v(V)$ for all variable $V \notin \tilde{X}_{\tau(p)}$ and $\mathscr{D} \models_{v'} sat(\tilde{X}_{\tau(p)}, \delta(p))$. Then, there exists a query Q such that $Set(Q) = \{p([\tilde{X}]_v)\}$ and a query Q' such that $Set(Q') = \{p([\tilde{X}]_{v'})\}$. Intuitively, as $\mathscr{D} \models_v c$, there exists a derivation step $Q \Longrightarrow Q_1$; moreover, as v' matches with v on $\tilde{X}_{\overline{\tau}(p)}$ and as the sequence of values that v' assigns to $\tilde{X}_{\tau(p)}$ satisfies Δ, then Q' is Δ-more general than Q. Therefore, as Δ is DN for r, there exists a query Q_1' such that $Q' \underset{r}{\Longrightarrow} Q_1'$ and Q_1' is Δ-more general than Q_1; using these properties of Q' and Q_1', one can deduce that $\mathscr{D} \models_{v'} \exists_{\mathscr{Y}} c$, where $\mathscr{Y} = \tilde{Y}_{\tau(q)} \cup local\_vars(r)$. We also have to prove that

$$\mathscr{D} \models c \rightarrow sat(\tilde{Y}_{\tau(q)}, \delta(q)) .$$

This is a consequence of the fact that for any derivation step $Q \underset{r}{\Longrightarrow} Q_1$, the query Q_1 satisfies Δ (because Δ is DN for r).

[2] Because $\langle p(1,0) \mid true\rangle_{\overline{\tau}} = \langle p_{\overline{\tau}}(0) \mid true\rangle = \langle p(0,0) \mid true\rangle_{\overline{\tau}}$ and $\langle p(1,0) \mid true\rangle_{\tau} = \langle p_{\tau}(1) \mid true\rangle$ with $Set\ \langle p_{\tau}(1) \mid true\rangle = \{p_{\tau}(1)\} \subseteq Set(\delta(p).)$

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Example 4.30
For any rational number x, there exists a term t constructed from the constant and function symbols of $\mathcal{Q}_{\text{lin}}$ such that $[t]_v = x$ for any valuation v. Therefore, for each atom $p(\tilde{a})$, where $\tilde{a}$ is a sequence of rational numbers, there exists a query Q in $\mathcal{Q}_{\text{lin}}$ of the form $\langle p(\tilde{t}) \mid true \rangle$, where the elements of $\tilde{t}$ are constructed from the constant and function symbols of $\mathcal{Q}_{\text{lin}}$, which is such that $Set(Q) = \{p(\tilde{a})\}$. Hence, by Theorem 4.29, in $\mathcal{Q}_{\text{lin}}$ DN is equivalent to DNlog.

4.6 Computing looping queries

For any filter $\Delta := (\tau, \delta)$ and any clause $r := p(\tilde{X}) \leftarrow c \diamond q(\tilde{Y})$, we let

- $\text{DNlog1}(\Delta, r) := \left(c \rightarrow \forall_{\tilde{X}_{\tau(p)}} \left[sat(\tilde{X}_{\tau(p)}, \delta(p)) \rightarrow \exists_{\mathcal{Y}} c\right]\right)$ and
- $\text{DNlog2}(\Delta, r) := \left(c \rightarrow sat(\tilde{Y}_{\tau(q)}, \delta(q))\right)$

denote the formulas in Definition 4.24.

A solution to compute a DNlog filter for a clause $r := p(\tilde{X}) \leftarrow c \diamond p(\tilde{Y})$ is to consider the *projection* of c on the elements of $\tilde{X}$ that we wish to distinguish and to check that DNlog1 and DNlog2 hold for r and the corresponding filter Δ_{proj}. Formally, for any set of variables W, the projection of c onto W is denoted by $\overline{\exists}_W c$ and is the formula $\exists_{\text{Var}(c) \setminus W} c$. If DNlog1 and DNlog2 hold for r and Δ_{proj}, then Δ_{proj} is DNlog for r; hence it is DN for r by Theorem 4.27; so we can try the test of Theorem 4.19 to get a query that loops w.r.t. $\{r\}$. Hence the following algorithm:

An algorithm to compute a looping query

Input: a clause $r := p(\tilde{X}) \leftarrow c \diamond p(\tilde{Y})$.

1. For each $m \subseteq [1, arity(p)]$ do:
2. Set $\tau(p) := m$, $\delta(p) := \langle p_\tau(\tilde{X}_{\tau(p)}) \mid \overline{\exists}_{\tilde{X}_{\tau(p)}} c \rangle$ and $\Delta_{\text{proj}} := (\tau, \delta)$.
3. If $\text{DNlog1}(\Delta_{\text{proj}}, r)$ and $\text{DNlog2}(\Delta_{\text{proj}}, r)$ hold then
4. If $\langle p(\tilde{Y}) \mid c \rangle$ is Δ_{proj}-more general than $\langle p(\tilde{X}) \mid c \rangle$ then
5. return $\langle p(\tilde{X}) \mid c \rangle$, which is a looping query w.r.t. $\{r\}$.

This algorithm *always* finds a DNlog filter. Indeed, for $m = \emptyset$, the corresponding filter $\Delta_{\text{proj}} = (\tau, \delta)$ is such that $\tilde{X}_{\tau(p)}$ is the empty sequence, so $\delta(p) = \langle p_\tau \mid \overline{\exists}_\emptyset c \rangle$, where $\overline{\exists}_\emptyset c$ is equivalent to $\exists_{\text{Var}(c)} c$, i.e. to *true* because in the definition of a clause (see Section 2) we suppose that c is satisfiable; therefore, $\text{DNlog1}(\Delta_{\text{proj}}, r)$ and $\text{DNlog2}(\Delta_{\text{proj}}, r)$ hold, as they are equivalent to $c \rightarrow (true \rightarrow \exists_{local\_vars(r)} c)$ and $c \rightarrow true$ respectively.

Four tests are performed by the above algorithm for each subset m of $[1, arity(p)]$: does $\text{DNlog1}(\Delta_{\text{proj}}, r)$ hold; does $\text{DNlog2}(\Delta_{\text{proj}}, r)$ hold; if these tests succeed, is $\langle p(\tilde{Y}) \mid c \rangle_{\overline{\tau}}$ more general than $\langle p(\tilde{X}) \mid c \rangle_{\overline{\tau}}$; and does $\langle p(\tilde{Y}) \mid c \rangle$ satisfy Δ_{proj}? Actually, only three tests are necessary as we have:

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Lemma 4.31
Let $r := p(\tilde{X}) \leftarrow c \diamond p(\tilde{Y})$ be a clause and $\Delta := (\tau, \delta)$ be a filter. Then, we have $\mathscr{D} \models \mathrm{DNlog2}(\Delta, r)$ if and only if $\langle p(\tilde{Y}) \mid c \rangle$ satisfies Δ.

Example 4.32
Let us consider the constraint domain $\mathscr{Q}_{\mathrm{lin}}$ and the recursive clause

$$r := p(X_1, X_2) \leftarrow X_1 \geqslant X_2 \wedge Y_1 = X_1 + 1 \wedge Y_2 = X_2 \diamond p(Y_1, Y_2)\,.$$

Let c be the constraint in r. Consider $m := \{1, 2\}$. The projection of c onto $\{X_1, X_2\}$ is the constraint $X_1 \geqslant X_2$; hence the algorithm sets $\tau(p) := \{1, 2\}$ and $\delta(p) := \langle p(X_1, X_2) \mid X_1 \geqslant X_2 \rangle$ and $\Delta_{\mathrm{proj}} := (\tau, \delta)$. The formulas $\mathrm{DNlog1}(\Delta_{\mathrm{proj}}, r)$ and $\mathrm{DNlog2}(\Delta_{\mathrm{proj}}, r)$ hold, as they are respectively equivalent to

$$c \rightarrow \forall X_1 \forall X_2 (X_1 \geqslant X_2 \rightarrow \exists Y_1 \exists Y_2 c) \qquad \text{and} \qquad c \rightarrow Y_1 \geqslant Y_2\,.$$

So, Δ_{proj} is DNlog for r. Moreover, as $\langle p(Y_1, Y_2) \mid c \rangle$ is Δ_{proj}-more general than $\langle p(X_1, X_2) \mid c \rangle$, by Theorem 4.19 the query $\langle p(X_1, X_2) \mid c \rangle$ loops w.r.t. $\{r\}$. Notice that by Definition 4.18, every query that is Δ_{proj}-more general than $\langle p(X_1, X_2) \mid c \rangle$ also loops w.r.t. $\{r\}$. Generally speaking, for any predicate symbol q/n, a set of positions $m \subseteq [1, n]$ can be seen as a finite representation of the set of queries of the form $\langle q(t_1, \ldots, t_n) \mid d \rangle$, where for each $i \in m$, d constrains t_i to a ground term. For instance, $\langle p(0, 0) \mid \mathit{true} \rangle$ loops w.r.t. $\{r\}$, as it is Δ_{proj}-more general than $\langle p(X_1, X_2) \mid c \rangle$; this query belongs to the class described by the set of positions $\{1, 2\}$ for p; therefore we say that this class is non-terminating because *there exists* a query in this class that loops. As $\langle p(0, X) \mid \mathit{true} \rangle$, $\langle p(X, 0) \mid \mathit{true} \rangle$ and $\langle p(X, Y) \mid \mathit{true} \rangle$ are more general than $\langle p(0, 0) \mid \mathit{true} \rangle$, by the Lifting Theorem 3.7 these queries also loop w.r.t. $\{r\}$; consequently, the classes described by the sets of positions $\{1\}$, $\{2\}$ and $\{\}$ for p are non-terminating too. So, for *every* set of positions m for p, the class of queries described by m is non-terminating.

Example 4.33
In $\mathscr{Q}_{\mathrm{lin}}$ again, consider the recursive clause (slightly different from that in Example 4.32)

$$r := p(X_1, X_2) \leftarrow X_1 \leqslant X_2 \wedge Y_1 = X_1 + 1 \wedge Y_2 = X_2 \diamond p(Y_1, Y_2).$$

Let c be the constraint in r and v be a valuation with $v(X_1) = v(X_2) = v(Y_2) = 0$ and $v(Y_1) = 1$; then we have $\mathscr{D} \models_v c$.

- Consider $m := \{1, 2\}$. The projection of c onto $\{X_1, X_2\}$ is $X_1 \leqslant X_2$; hence the algorithm sets $\tau(p) := \{1, 2\}$, $\delta(p) := \langle p(X_1, X_2) \mid X_1 \leqslant X_2 \rangle$ and $\Delta_{\mathrm{proj}} := (\tau, \delta)$. The formula $\mathrm{DNlog2}(\Delta_{\mathrm{proj}}, r)$ is equivalent to $c \rightarrow Y_1 \leqslant Y_2$. We have $\mathscr{D} \models_v c$ and $\mathscr{D} \not\models_v Y_1 \leqslant Y_2$, so $\mathscr{D} \not\models_v c \rightarrow Y_1 \leqslant Y_2$. Therefore, $\mathrm{DNlog2}(\Delta_{\mathrm{proj}}, r)$ does not hold, so Δ_{proj} is not DNlog for r.
- Consider $m := \{1\}$. The projection of c onto $\{X_1\}$ is equivalent to the constraint *true*. The algorithm sets $\tau(p) := \{1\}$, $\delta(p) := \langle p_\tau(X_1) \mid \mathit{true} \rangle$ and $\Delta_{\mathrm{proj}} := (\tau, \delta)$. The formula $\mathrm{DNlog1}(\Delta_{\mathrm{proj}}, r)$ is equivalent to $c \rightarrow \forall X_1 (\mathit{true} \rightarrow \exists Y_1 c)$, i.e. $c \rightarrow \forall X_1 \exists Y_1 c$. We have $\mathscr{D} \models_v c$; if we change the value assigned to X_1 to 1,

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

then $X_1 \leqslant X_2$ (a subformula of c) does not hold anymore, and one cannot find any value for Y_1 such that $X_1 \leqslant X_2$ holds again; therefore, we have $\mathscr{D} \not\models_v \forall X_1 \exists Y_1 c$ so $\mathscr{D} \not\models_v c \rightarrow \forall X_1 \exists Y_1 c$. Hence, DNlog1($\Delta_{\text{proj}}, r$) does not hold, so Δ_{proj} is not DNlog for r.

- Consider $m := \{2\}$. The projection of c onto $\{X_2\}$ is equivalent to the constraint *true*. The algorithm sets $\tau(p) := \{2\}$, $\delta(p) := \langle p_\tau(X_2) \mid true \rangle$ and $\Delta_{\text{proj}} := (\tau, \delta)$. The formula DNlog1($\Delta_{\text{proj}}, r$) is equivalent to $c \rightarrow \forall X_2(true \rightarrow \exists Y_2 c)$, i.e. $c \rightarrow \forall X_2 \exists Y_2 c$. We have $\mathscr{D} \models_v c$; if we change the value assigned to X_2 to -1, then $X_1 \leqslant X_2$ (a subformula of c) does not hold anymore, and one cannot find any value for Y_2 such that $X_1 \leqslant X_2$ holds again; therefore, we have $\mathscr{D} \not\models_v \forall X_2 \exists Y_2 c$, so $\mathscr{D} \not\models_v c \rightarrow \forall X_2 \exists Y_2 c$. Hence, DNlog1($\Delta_{\text{proj}}, r$) does not hold, so Δ_{proj} is not DNlog for r.
- Consider $m := \emptyset$. The projection of c onto $\emptyset$ is equivalent to the constraint *true*. The algorithm sets $\tau(p) := \emptyset$, $\delta(p) := \langle p_\tau \mid true \rangle$ and $\Delta_{\text{proj}} := (\tau, \delta)$. Both DNlog1($\Delta_{\text{proj}}, r$) and DNlog2($\Delta_{\text{proj}}, r$) hold, as they are equivalent to $c \rightarrow (true \rightarrow c)$ and $c \rightarrow true$ respectively. So, Δ_{proj} is DNlog for r. As $\langle p(Y_1, Y_2) \mid c \rangle$ is Δ_{proj}-more general than $\langle p(X_1, X_2) \mid c \rangle$, by Theorem 4.19 $\langle p(X_1, X_2) \mid c \rangle$ loops w.r.t. $\{r\}$. This query allows us to conclude that the class described by the set of positions $\{\}$ for p is non-terminating.

Consequently, we get no information about the classes described by the sets of positions $\{1, 2\}$, $\{1\}$ and $\{2\}$. Actually, the class described by $\{1, 2\}$ is terminating, i.e. *every* query in this class does not loop; indeed, intuitively, when the arguments of p in a query Q are fixed to some values in $\mathbb{Q}$, we have a finite derivation of $\{r\} \cup \{Q\}$ because in r the first argument of p strictly increases until it becomes greater than the second argument. Hence, the class described by $\{1, 2\}$ will not be inferred by our approach. On the other hand, the query $\langle p(1, X) \mid true \rangle$ loops w.r.t. $\{r\}$, which implies that the class described by $\{1\}$ is non-terminating. Our approach fails to infer this result, as X_1 and X_2 interact in c via $X_1 \leqslant X_2$, so there is no DNlog filter for r that distinguishes position 1 and not position 2 of p. Hence, as DN and DNlog match in this example, the DN approach fails[3] to infer the non-termination of $\{1\}$. So, a limitation of the DN approach when DN and DNlog match is the following: when two arguments interact, if there is no DNlog filter that distinguishes both their positions, then it is not possible to infer non-termination of a class of queries described by a set containing one of these positions and not the other. Notice that non-interaction of arguments is expressed by DNlog and not necessarily by DN; when DNlog and DN do not match (see Theorem 4.29), there are situations in which DN arguments can interact with non-DN arguments. In Example 4.28,

[3] Note that the situation of this example is different from that of Example 4.32. Here, we cannot infer the non-termination of the class described by $\{1\}$ from the non-termination of the class described by $\{\}$. Indeed, every element in the class described by $\{1\}$ has the form $\langle p(t_1, t_2) \mid d \rangle$, where d constrains t_1 to a ground term; on the other hand, every element in the class described by $\{\}$ has the form $\langle p(t'_1, t'_2) \mid d' \rangle$, where t'_1 and t'_2 are not constrained to some ground terms; hence $\langle p(t_1, t_2) \mid d \rangle$ is not more general than $\langle p(t'_1, t'_2) \mid d' \rangle$.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

the arguments of p at positions 1 and 2 interact via $X_2 \geqslant X_1$; the filter that we give in this example distinguishes position 1 but not position 2 of p, and it is DN for r.

5 An implementation

We have implemented the analysis in SWI-Prolog (Wielemaker 2003) for CLP($\mathcal{Q}_{\text{lin}}$). The prototype[4] takes a recursive binary rule $p(\tilde{X}) \leftarrow c \diamond p(\tilde{Y})$ as input and tries to find a filter with the projection of the constraint c of the considered rule onto its head variables $\tilde{X}$. For each possible set of positions, it computes the four logical formulas corresponding to Definition 4.14 and Definition 4.24. As the number of such sets is exponential w.r.t. the arity of the predicate p, our analysis is at least exponential. These formulas are evaluated by a decision procedure for arbitrary logical formulas over $\langle \mathbb{Q}; \{0,1\}; \{+\}; \{=, <\}\rangle$. If they are true (note that Lemma 4.31 shows that some tests are redundant), the analyzer prints the corresponding filter and computes a concrete looping query.

So the analyzer implements Theorem 4.19 with the help of Theorem 4.27. We point out that the analysis can be automated for any constraint domain the theory of which is decidable, e.g. logic programming with finite trees and logic programming with rational trees (Maher 1988).

Table 1 summarizes the result of the analysis of a set of handcrafted binary rules. The symbol ✓ indicates those examples that the analysis presented in Payet and Mesnard (2004) could not prove non-terminating.

6 Conclusion

In Payet and Mesnard (2004) we have presented a technique to complement termination analysis with non-termination inside the logic programming paradigm. Our aim was to detect optimal termination conditions expressed in a language describing classes of queries. The approach was syntactic and linked to some basic logic programming machinery such as the unification algorithm. In Payet and Mesnard (2004) we have presented the first step at generalizing the work of Payet and Mesnard (2006) to the CLP setting. The logical criterion we gave only considers those filters the function δ of which does not filter anything, i.e. δ maps any predicate symbol p to $\langle p_\tau(\tilde{X}) \mid true\rangle$.

This paper describes a generalization of Payet and Mesnard (2006) to the CLP setting. It presents a criterion, both in an operational and a logical form, to infer non-terminating atomic queries with respect to a binary CLP clause. This criterion is generic in the constraint domain; its logical form strictly generalizes that of Payet and Mesnard (2004), and it has been fully implemented for CLP($\mathcal{Q}_{\text{lin}}$).

[4] Available at `http://personnel.univ-reunion.fr/fred/dev/DNlog4Q.zip`.

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Table 1. *Running the analyzer on a set of examples*

Binary clause	τ	δ	Looping query	
$p(A) \leftarrow true \diamond p(B)$	$\{1\}$	$\langle p(X) \mid true \rangle$	$\langle p(0) \mid true \rangle$	
$p(A) \leftarrow A = B \diamond p(B)$	$\{1\}$	$\langle p(X) \mid true \rangle$	$\langle p(0) \mid true \rangle$	
$p(A) \leftarrow A = 0 \diamond p(B)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A) \mid A = 0 \rangle$	
$p(A) \leftarrow A = 0 \wedge B = 0 \diamond p(B)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A) \mid A = 0 \rangle$	
$p(A) \leftarrow A = 0 \wedge B = 1 \diamond p(B)$			None found	
$p(A) \leftarrow A \geqslant 0 \wedge B = 1 \diamond p(B)$	$\{1\}$	$\langle p(X) \mid X \geqslant 0 \rangle$	$\langle p(0) \mid true \rangle$	✓
$p(A) \leftarrow A \geqslant 0 \wedge B \geqslant 1 \diamond p(B)$	$\{1\}$	$\langle p(X) \mid X \geqslant 0 \rangle$	$\langle p(0) \mid true \rangle$	✓
$p(A) \leftarrow A \geqslant 0 \wedge B \geqslant -1 \diamond p(B)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A) \mid A \geqslant 0 \rangle$	
$p(A) \leftarrow A \geqslant 1 \wedge B \leqslant 0 \diamond p(B)$			None found	
$p(A) \leftarrow A = B + 1 \wedge B \geqslant 0 \diamond p(B)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A) \mid A \geqslant 1 \rangle$	
$p(A,B) \leftarrow A = C + 1 \wedge C \geqslant 0 \diamond p(C,D)$	$\{2\}$	$\langle p(Y) \mid true \rangle$	$\langle p(A,0) \mid A \geqslant 1 \rangle$	
$p(A,B) \leftarrow A = C + 1 \wedge C \geqslant 0 \wedge B = D \diamond p(C,D)$	$\{2\}$	$\langle p(Y) \mid true \rangle$	$\langle p(A,0) \mid A \geqslant 1 \rangle$	
$p(A,B) \leftarrow A = C + 1 \wedge C \geqslant 0 \wedge B + 1 = D \diamond p(C,D)$	$\{2\}$	$\langle p(Y) \mid true \rangle$	$\langle p(A,0) \mid A \geqslant 1 \rangle$	
$p(A,B) \leftarrow A = C + 1 \wedge C \geqslant 0 \wedge B + 1 = D \wedge D \geqslant 0 \diamond p(C,D)$	$\{2\}$	$\langle p(Y) \mid Y \geqslant -1 \rangle$	$\langle p(A,-1) \mid A \geqslant 1 \rangle$	✓
$p(A,B) \leftarrow A = C + 1 \wedge C \geqslant 0 \wedge B = D + 1 \wedge D \geqslant 0 \diamond p(C,D)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A,B) \mid A \geqslant 1 \wedge B \geqslant 1 \rangle$	
$p(A,B) \leftarrow A \geqslant B \wedge C = A + 1 \wedge D = B \diamond p(C,D)$	$\{1,2\}$	$\langle p(X,Y) \mid X \geqslant Y \rangle$	$\langle p(0,0) \mid true \rangle$	✓
$p(A,B) \leftarrow A \leqslant B \wedge C = A + 1 \wedge D = B \diamond p(C,D)$	$\emptyset$	$\langle p \mid true \rangle$	$\langle p(A,B) \mid A \leqslant B \rangle$	
$pow2(A,B,C) \leftarrow A = D + 1 \wedge D \geqslant 0 \wedge E = 2 * B \wedge B \geqslant 1 \wedge F = C \wedge C \geqslant 2 \diamond pow2(D,E,F)$	$\{2,3\}$	$\langle pow2(Y,Z) \mid Y \geqslant 1 \wedge Z \geqslant 2 \rangle$	$\langle pow2(A,1,2) \mid A \geqslant 1 \rangle$	✓

https://doi.org/10.1017/S1471068409003652 Published online by Cambridge University Press

Acknowledgements

The authors thank the anonymous reviewers for helpful comments on the previous versions of this paper.

References

Bol, R. N., Apt, K. R. and Klop, J. W. 1991. An analysis of loop checking mechanisms for logic programs. *Theoretical Computer Science 86*, 35–79.

Codish, M. and Taboch, C. 1999. A semantics basis for termination analysis of logic programs. *Journal of Logic Programming 41*, 1, 103–123.

De Schreye, D., Bruynooghe, M. and Verschaetse, K. 1989. On the existence of nonterminating queries for a restricted class of Prolog-clauses. *Artificial Intelligence 41*, 237–248.

De Schreye, D. and Decorte, S. 1994. Termination of logic programs: The never-ending story. *Journal of Logic Programming 19–20*, 199–260.

Gabbrielli, M. and Giacobazzi, R. 1994. Goal independency and call patterns in the analysis of logic programs. In *Proc. of the ACM Symposium on Applied Computing (SAC'94)*. Hal Berghel, Terry Hlengl and Joseph Urban, ACM Press, New York, 394–399.

Giesl, J., Thiemann, R. and Schneider-Kamp, P. 2005. Proving and disproving termination of higher-order functions. In *Proc. of the 5th International Workshop on Frontiers of Combining Systems (FroCoS'05)*, B. Gramlich, Ed. Lecture Notes in Artificial Intelligence, vol. 3717. Springer-Verlag, Berlin, 216–231.

Godefroid, P., Klarlund, N. and Sen, K. 2005. DART: Directed Automated Random Testing. In *Proc. of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI'05)*, V. Sarkar and M. W. Hall, Eds. ACM, New York, 213–223.

Gupta, A., Henzinger, T. A., Majumdar, R., Rybalchenko, A. and Xu, R.-G. 2008. Proving non-termination. In *Proc. of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'08)*, G. Necula and P. Wadler, Eds. ACM, New York, 147–158.

Jaffar, J., Maher, M. J., Marriott, K. and Stuckey, P. J. 1998. The semantics of constraint logic programs. *Journal of Logic Programming 37*, 1–3, 1–46.

Maher, M. 1988. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proc. of the 3rd Annual Symposium on Logic in Computer Science (LICS'88)*. IEEE Computer Society, Los Alamitos, CA, 348–357.

Mesnard, F. and Ruggieri, S. 2003. On proving left termination of constraint logic programs. *ACM Transactions on Computational Logic 4*, 2, 207–259.

Payet, E. 2008. Loop detection in term rewriting using the eliminating unfoldings. *Theoretical Computer Science 403*, 307–327.

Payet, E. and Mesnard, F. 2004. Non-termination inference for constraint logic programs. In *Proc. of the 11th International Symposium on Static Analysis (SAS'04)*, R. Giacobazzi, Ed. Lecture Notes in Computer Science, vol. 3148. Springer-Verlag, Berlin, 377–392.

Payet, E. and Mesnard, F. 2006. Non-termination inference of logic programs. *ACM Transactions on Programming Languages and Systems 28*, 2, 256–289.

Refalo, P. and Hentenryck, P. V. 1996. CLP($\mathscr{R}_{\text{lin}}$) revised. In *Proc. of the Joint International Conf. and Symposium on Logic Programming*, M. Maher, Ed. The MIT Press, Cambridge, MA, 22–36.

Sen, K., Marinov, D. and Agha, G. 2005. CUTE: A Concolic Unit Testing Engine for C. In *Proc. of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, M. Wermelinger and H. Gall, Eds. ACM, New York, 263–272.

Shen, Y.-D., Yuan, L.-Y. and You, J.-H. 2001. Loops checks for logic programs with functions. *Theoretical Computer Science 266*, 1–2, 441–461.

Speirs, C., Somogyi, Z. and Søndergaard, H. 1997. Termination analysis for Mercury. In *Proc. of the 1997 Intl. Symp. on Static Analysis*, P. van Hentenrick, Ed. Lecture Notes in Computer Science, vol. 1302. Springer-Verlag, Berlin, 160–171.

Waldmann, J. 2004. Matchbox: A tool for match-bounded string rewriting. In *Proc. of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, V. van Oostrom, Ed. Lecture Notes in Computer Science, vol. 3091. Springer-Verlag, Berlin, 85–94.

Waldmann, J. 2007. Compressed loops (draft).

`http://dfa.imn.htwk-leipzig.de/matchbox/methods/`.

Wielemaker, J. 2003. An overview of the SWI-Prolog programming environment. In *Proceedings of the 13th International Workshop on Logic Programming Environments (WLPE'03)*, F. Mesnard and A. Serebenik, Eds. Vol. CW371. Department of Computer Science, Katholieke Universiteit Leuven, Heverlee, Belgium, 1–16.

Zankl, H. and Middeldorp, A. 2007. Nontermination of string rewriting using SAT. *Presented at the 9th International Workshop on Termination (WST'07)*.

Zantema, H. 2005. Termination of string rewriting proved automatically. *Journal of Automated Reasoning 34*, 2, 105–139.