

# Termination analysis for abductive general logic programs

*Sofie Verbaeten*

*Report CW 296, July 2000*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Termination analysis for abductive general logic programs

*Sofie Verbaeten*

*Report CW296, July 2000*

Department of Computer Science, K.U.Leuven

## **Abstract**

We present an extension of the methods of Apt and Bezem for proving termination of general logic programs, to the case of abductive general logic programs. We consider programs executed under SLDNFA resolution, an abductive extension of SLDNF proposed by Denecker and De Schreye, w.r.t. an arbitrary safe selection rule. We show that the termination conditions for SLDNF of Apt and Bezem, namely acyclicity of the program and boundedness of the query, are not sufficient for ensuring termination under SLDNFA. A third syntactic condition, namely abductive nonrecursivity of the program and query, is proposed which prevents the abducting of an infinite number of abducible atoms. Acyclicity of the program, boundedness of the query and abductive nonrecursivity of the program and query form our sufficient termination condition for SLDNFA. By the best of our knowledge, this is the first work on termination of an abductive procedure for general logic programs.

**Keywords :** Logic Programming, Termination Analysis.

**CR Subject Classification :** I.2.3, I.2.2

# 1 Introduction

The role of abduction as a reasoning paradigm in AI is widely accepted. Abduction is a form of reasoning which, given a knowledge base and an observation  $Q$ , finds possible explanations of  $Q$  in terms of a particular set of predicates, called the abducible predicates. In the context of logic programming, abductive procedures have been used for a.o. planning [11, 17, 18], knowledge assimilation and belief revision [14], database updating [15], and reasoning in the context of temporal domains with uncertainty [9]. In [8], Denecker and De Schreye present an abductive extension of SLDNF [1], called SLDNFA. We investigate the termination behaviour of general logic programs executed under SLDNFA w.r.t. an arbitrary safe selection rule. We show that the termination conditions for SLDNF of Apt and Bezem [3], namely acyclicity of the program and boundedness of the query w.r.t. a level mapping, are not sufficient for ensuring termination under SLDNFA. In particular, these conditions do not prevent an SLDNFA-derivation from abducting an infinite number of abducible atoms. We propose a third, syntactical condition, namely abductive nonrecursivity of the program and query. This condition together with the acyclicity of the program and boundedness of the query are sufficient for proving termination of an SLDNFA-derivation using a safe selection rule. For definite programs and queries as well as for programs and queries without abducible predicates, the condition of abductive nonrecursivity is trivially satisfied. So, in these cases, termination of SLDNFA comes down to acyclicity of the program and boundedness of the query. By the best of our knowledge, this is the first work on termination of an abductive procedure for general logic programs.

Besides SLDNFA, a number of other abductive extensions of SLDNF resolution have been proposed, including [20, 17, 15, 19, 21, 5]. However, as discussed in [8], either these procedures have not been formalized and proved correct, or they can be proved correct only for a restricted class of abductive logic programs, or they do not provide a way of checking the consistency of the abductive answers, or they do not provide a treatment for floundering abduction (which is an analog problem of the negation floundering problem and arises when a nonground abductive atom is selected). An exception is the iff procedure of [13]. SLDNFA and iff are in many respects complementary. However, while SLDNFA is formalized in a logic programming style, iff is formalized as a rewrite procedure using completion. We refer to [8] for a formal discussion on these works and a detailed comparison with SLDNFA. Our choice for SLDNFA is also motivated by the following reason. Similar to the definition of SLDNF [1], SLDNFA is based on a schema which models the computation process. This allows us to reason on and prove properties of “run time behaviour”, like termination, of derivations.

The paper is structured as follows. After a short section of preliminaries, we recall in Section 3 the SLDNFA proof procedure as defined in [8]. In Section 4, we show that the termination conditions for SLDNF, as proposed in [3], are not sufficient for proving termination of SLDNFA. We present a sufficient termination condition, extending the condition of [3], for SLDNFA executed under an arbitrary safe selection rule. We conclude in Section 5. In the appendix, one can find some more examples illustrating concepts introduced in this paper.

## 2 Preliminaries

We assume familiarity with the basic concepts of logic programming [16, 2]. Throughout the paper,  $P$  will denote a general logic program based on an alphabet  $\Sigma$ . Let  $\Sigma^p \subset \Sigma$  denote the set of predicate symbols of  $\Sigma$ . For a set  $A \subset \Sigma^p$  of predicate symbols, we denote with  $P^A$  the abductive general logic program  $P$  with set of abducible predicates  $A$ . In Section 3, the SLDNFA proof procedure for abductive programs [8] is defined.

With  $U_P$ , resp.  $B_P$ , we denote the *Herbrand Universe*, resp. *Herbrand Base*, associated with ( $\Sigma$  underlying)  $P$ . With  $\neg B_P$  we denote the set of negative ground literals  $\{\neg A \mid A \in B_P\}$ . The *extended Herbrand Universe*,  $U_P^E$ , and the *extended Herbrand Base*,  $B_P^E$ , associated with  $P$ , were introduced in [12]. They are defined as follows. Let  $Term_P$  and  $Atom_P$  denote the set of respectively all terms and atoms that can be constructed from the alphabet underlying  $P$  (including variables). The variant relation, denoted  $\approx$ , defines an equivalence.  $U_P^E$  and  $B_P^E$  are respectively the quotient sets  $Term_P / \approx$  and  $Atom_P / \approx$ . For any term  $t$  (or atom  $A$ ), we denote its class in  $U_P^E$  ( $B_P^E$ ) as  $\tilde{t}$  ( $\tilde{A}$ ). However, when no confusion is possible, we omit the tildes. Let  $\neg B_P^E = \{\neg A \mid A \in B_P^E\}$ .

With  $ground_{\Sigma}(P)$  we denote the set of  $\Sigma$ -ground instances of clauses of  $P$ . For a clause  $C$  of  $P$ , we denote with  $Head(C)$  the atom occurring in the head of  $C$  and with  $Body(C)$  the set of literals occurring in the body of  $C$ .

Let  $p, q$  be two predicate symbols of  $P$ . We say that  $p$  *refers to*  $q$  in  $P$  iff there is a clause in  $P$  with  $p$  in the head and  $q$  occurring in the body. When necessary, we distinguish between two kinds of “refers to”-arcs: there is a *positive*, resp. *negative*, *arc* from  $p$  to  $q$  iff there is a clause in  $P$  with  $p$  in the head and  $q$  occurring positively, resp. negatively, in the body. We say that  $p$  *depends on*  $q$  in  $P$ , and write  $p \sqsupseteq q$ , iff  $(p, q)$  is in the reflexive, transitive closure of the relation refers to. We write  $p \simeq q$  iff  $p \sqsupseteq q$  and  $q \sqsupseteq p$  ( $p$  and  $q$  are mutually recursive or  $p = q$ ), and  $p \sqsubset q$  iff  $p \sqsupseteq q$  and  $q \not\sqsupseteq p$ . If  $L$  is a literal  $p(t_1, \dots, t_n)$  or  $\neg p(t_1, \dots, t_n)$ , then we define  $Rel(L) = p$ . With a *signed predicate* of a set  $P$  of predicate symbols, we mean  $p$  or  $\neg p$ , where  $p \in P$ . An atom of the form  $p(\bar{t})$  will be called a *p-atom*.

### 3 SLDNFA

An abductive logic program  $P^A$  based on  $\Sigma$  is a general program  $P$  based on  $\Sigma$  together with a subset  $A$  of predicates of  $\Sigma^p$ , called *abducible*. The other predicates in  $\Sigma^p \setminus A$  are called *nonabducible*. Without loss of generality, we assume that for any nonpropositional abductive logic program  $P^A$ , equality  $=$  is defined by one unit clause,  $X = X \leftarrow$ , in  $P$  (without explicitly stating this in the examples).

In [8], the *3-valued completion semantics* for abductive logic programs and the notion of *abductive solution* for a query w.r.t. a program are defined. By the lack of space, we will not go into detail here and we will only give the proof procedure, called SLDNFA. This section is based on [8] and we refer to [8] for the soundness and completeness results and for more details.

An SLDNFA computation can be understood as a process of deriving formulas  $\forall(Q_0 \leftarrow \Phi)$ , with  $\leftarrow Q_0$  the initial query and the conjunction  $\Phi$  composed of two types of unsolved queries:

- For any query  $\leftarrow Q$  for which a derivation still needs to be computed (in the sequel, a *positive query*),  $\Phi$  contains the open formula  $Q$  (denoting the open conjunction of literals in  $Q$ ).
- For any query  $\leftarrow Q$  for which a failure tree still needs to be constructed (in the sequel, a *negative query*),  $\Phi$  contains the open formula  $\forall \bar{X}. \leftarrow Q$  with  $\bar{X}$  a subset of the variables of  $Q$ .

SLDNFA selects *unsolved* positive or negative queries  $\leftarrow Q$  and literals in  $\leftarrow Q$  and rewrites these queries depending on the sort of selection. These rewrite operations can be interpreted as theorem proving steps on  $\Phi$ . Resolution is applied on nonabducible atoms selected in positive and negative queries, as in SLDNF. Negative literals are deleted from positive queries and added as negative queries and vice versa, just as in SLDNF. Abducible atoms in positive queries are never selected: they are treated as residual atoms. They are called *abduced atoms*. The process of computing an increasing set of residual abducible atoms can be understood as the incremental construction of a definition for the abducible predicates. Abducible atoms selected in negative queries are resolved with the residual atoms in the positive queries.

Note that  $\Phi$  contains two types of variables: free variables (universally quantified in front of  $\forall(Q_0 \leftarrow \Phi)$ ) and variables universally quantified in a conjunct of  $\Phi$ . This distinction plays a crucial role in SLDNFA. The variables that appear free in  $\Phi$  will be called *positive* and the variables that appear universally quantified in  $\Phi$  will be called *negative*. Whether variables in queries in a derivation are positive or negative depends on the way they are introduced in the derivation. The positive variables are either variables of the initial query  $Q_0$  or are the variables of input program clauses used for resolution with positive queries. The negative variables are the variables of program clauses used for resolution with negative queries. What follows is a precise description of how the basic operations of unification and resolution are modified to take the difference between positive and negative variables into account.

**Definition 3.1 (marking)** *A marking  $\alpha$  is a partial function of the set of variables of an alphabet  $\Sigma$  to the set  $\{+, -\}$ . Given a marking  $\alpha$ , a variable  $X$  is marked iff  $\alpha(X)$  is defined. A marked atom, equality set, query, program clause is one in which all variables are marked.*

Given a marking  $\alpha$ , we denote with  $X^+$ , resp.  $Y^-$ , that  $\alpha(X) = +$ , resp.  $\alpha(Y) = -$ . A marking can be seen as a memo to inform unification and resolution in SLDNFA about the logical nature of the variables.

**Definition 3.2 (positive solved form; solvable equality set)**

An equality set is in solved form iff it is a set of equality atoms of the form  $X = t$  such that  $X$  is a variable different from the term  $t$  and  $X$  occurs only once at the left and not at the right.

Given a marking  $\alpha$ , a marked equality set is in positive solved form iff it is in solved form and it contains no atoms of the form  $X^+ = Y^-$ .

An equality set  $E_s$  is a (positive) solved form of an equality set  $E$  iff  $E_s$  is in (positive) solved form and  $E_s$  is a mgu of  $E$ . An equality set with a solved form is called solvable; an equality set without solved form is called unsolvable. Two atoms  $p(\bar{t}), p(\bar{s})$  are said to be unifiable iff  $\{\bar{t} = \bar{s}\}$  is solvable.

It is straightforward that (given some marking  $\alpha$ ) a marked equality set has a solved form iff it has a positive solved form. Also, a correct unification algorithm can be extended to an algorithm computing a positive solved form.

**Definition 3.3 (positive resolution)** Given is a marking  $\alpha$ , a marked query  $\leftarrow Q = \leftarrow B_1, p(\bar{t}), B_2$  and  $C \equiv p(\bar{s}) \leftarrow B'$  a marked clause.

$\leftarrow Q'$  is derived from  $\leftarrow Q$  and  $C$  by positive resolution on  $p(\bar{t})$  using  $\theta$  if the following holds: (1)  $\theta$  is a solved form of  $\bar{t} = \bar{s}$  and (2)  $\leftarrow Q'$  is the query  $\theta(\leftarrow B_1, B', B_2)$ . We call  $\leftarrow Q'$  the positive resolvent.

In SLDNFA, positive resolution is applied to positive queries.

If  $E$  is a marked equality set in positive solved form, define  $E_+$ , resp.  $E_-$ , as the subset of  $E$  having a positive variable, resp. negative variable, at the left.

**Definition 3.4 (negative resolution)** Given is a marking  $\alpha$ , a marked query  $\leftarrow Q = \leftarrow B_1, p(\bar{t}), B_2$  and  $C \equiv p(\bar{s}) \leftarrow B'$  a marked program clause or a marked abducible atom ( $B' = \{ \}$ ).

$\leftarrow Q'$  is derived from  $\leftarrow Q$  and  $C$  by negative resolution on  $p(\bar{t})$  if the following holds: (1)  $\bar{t} = \bar{s}$  has a positive solved form  $E$  and (2)  $\leftarrow Q'$  is the query  $E_-(\leftarrow B_1, E_+, B', B_2)$ . We call  $\leftarrow Q'$  the negative resolvent.

**Definition 3.5 (irreducible equality atom)** We call  $s = t$  irreducible when  $s$  is a positive variable and  $t$  is either a nonvariable term or another positive variable.

Negative resolution is applied to negative queries. In such a negative query, an irreducible atom formulates a disequality constraint on a positive variable. Negative resolution will never bind positive variables; instead it generates disequality constraints on them.

Note that, for queries and program clauses that contain only positive (resp. negative) variables, positive (resp. negative) resolution and classical resolution coincide.

The following notion of prederivation serves as a kind of skeleton for the notion of an SLDNFA-derivation.

**Definition 3.6 (prederivation)** Given is an abductive logic program  $P^A$  based on an alphabet  $\Sigma$ . A prederivation  $K$  is a tuple  $((\theta_1, \dots, \theta_n), T, \alpha)$  with  $(\theta_1, \dots, \theta_n)$  a sequence of substitutions,  $T$  a tree of labeled queries  $\leftarrow Q$  and labeled arcs, and  $\alpha$  a marking of the variables of  $\Sigma$ . Each query is labeled positive or negative. A query  $\leftarrow Q$  in the tree may or may not be labeled by a literal in  $\leftarrow Q$ , called the selected literal.

An arc from a negative query  $\leftarrow Q_1, L, Q_2$  with a selected atom  $L$  arrives in a negative query  $\leftarrow Q'$  and is labeled with a program clause  $H \leftarrow B$  (or an abducible atom  $H$ ), called the applied resolvendus.  $\leftarrow Q'$  is derived from  $\leftarrow Q_1, L, Q_2$  and  $H \leftarrow B$  (resp.  $H$ ) by negative resolution on  $L$ .

The sequence of substitutions in a prederivation will be the sequence of substitutions computed at positive resolution steps.

Given an abductive logic program  $P^A$  and a prederivation  $K$ , a resolvendus of a node  $N$  in  $K$  with a nonabducible selected atom  $L$  is any program clause  $H \leftarrow B$  of  $P^A$  such that  $L$  and a variant of  $H$  are unifiable. If the selected atom  $L$  of  $N$  is abducible, then a resolvendus of  $N$  is any abducible atom  $L'$  in a positive query of  $K$  such that  $L$  and  $L'$  are unifiable. For a negative query  $\leftarrow Q$  in  $K$ , we distinguish between applied resolventi (those appearing as applied resolventi of arcs leaving  $\leftarrow Q$  in  $K$ ) and the other, nonapplied resolventi.

**Definition 3.7 (selection)** Given is a prederivation  $K$ .

A first SLDNFA-selection in  $K$  is a tuple  $(N, L)$ , where  $N$  is a positive or negative query in  $K$  without selected literal, and  $L$  is a literal in  $N$ . If  $N$  is a positive query, then  $L$  is not an abducible atom.

An SLDNFA-reselection in  $K$  is a tuple  $(N, C)$ , where  $N$  is a negative query in  $K$  labeled with a selected atom  $L$  and  $C$  is a nonapplied resolvent of  $N$  (w.r.t.  $K$  and  $P^A$ ).

An SLDNFA-selection in  $K$  is a first selection or reselection in  $K$ .

Note that an abducible atom in a positive query cannot be selected. The only queries that can be selected more than once are negative queries with a selected atom for which different branches of the failure tree are to be explored.

In [8] it is shown that the safety condition on the selection rule in SLDNF (namely that only *ground* negative literals can be selected) can be weakened in the SLDNFA procedure: positive variables may appear in selected negative literals. However, SLDNFA does not offer a solution for the treatment of negative variables in positive queries. Therefore the following (weak) safety condition is imposed on the selection rule in SLDNFA.

**Definition 3.8 (safe selection)** Given is a prederivation  $K$ . A selection is safe iff it is a reselection or if it is a first selection  $(N, L)$  such that  $L$  is not a negative literal containing negative variables.

A program clause  $C'$  is called a *standardized apart* variant of a program clause  $C$  w.r.t. a prederivation  $K = ((\theta_1, \dots, \theta_n), T, \alpha)$  iff  $C'$  is a variant of  $C$  and the variables in  $C'$  appear neither in  $C$ , nor in  $\theta_1, \dots, \theta_n$  nor in  $T$ .

**Definition 3.9 (SLDNFA-derivation)** Let  $P^A$  be an abductive logic program and  $\leftarrow Q_0$  be a query. An SLDNFA-derivation is defined by induction:

- The tuple  $((\ ), T_0, \alpha_0)$ , with  $T_0$  a tree consisting of a single positive query  $\leftarrow Q_0$  and  $\alpha_0$  the marking that marks the variables of  $Q_0$  positive, is an SLDNFA-derivation.
- Given an SLDNFA-derivation  $K$ , an SLDNFA-extension of  $K$  using some safe selection in  $K$  is an SLDNFA-derivation.

An SLDNFA-extension of a pre-derivation  $K = ((\theta_1, \dots, \theta_n), T, \alpha)$  is defined as follows.

Let  $(N, L)$  be a first selection in  $K$  with  $N \equiv \leftarrow Q$ .

An SLDNFA-extension of  $K$  using first selection  $(N, L)$  is a prederivation  $K' = ((\theta_1, \dots, \theta_n, \theta), T', \alpha')$  such that  $T'$  is obtained from  $T$  by adding a set  $S$  with zero, one, or two descendants to  $N$ , marking  $N$  with selected literal  $L$ , and applying  $\theta$  on all queries and labels of  $T$ .  $\alpha'$ ,  $\theta$  and the set of descendants  $S$  satisfy one of the following conditions:

- Let  $N$  be a positive query and  $L$  be an atom  $p(\bar{t})$  with  $p$  nonabducible.  
 $\alpha'$  extends  $\alpha$  by marking all variables of a standardized apart variant  $C'$  of a program clause  $C \in P$  positive.  $S$  is a singleton containing a positive query  $\leftarrow Q'$ , which is derived by positive resolution from  $\leftarrow Q$  and  $C'$  on  $p(\bar{t})$  using  $\theta$ .  
In all other cases  $\theta = \epsilon$  (the empty substitution) and  $\alpha' = \alpha$ . Depending on the type of selection,  $S$  satisfies the following conditions:
- Let  $N$  be a positive query and  $L = \neg A$ .  
 $S$  is a pair consisting of a negative query  $\leftarrow A$  and a positive query  $\leftarrow Q'$  obtained by deleting  $\neg A$  in  $\leftarrow Q$ .
- Let  $N$  be a negative query and  $L = \neg A$ .  
Either  $S$  is the singleton containing one positive query  $\leftarrow A$ , or  $S$  is the singleton consisting of a negative query  $\leftarrow Q'$  obtained by deleting  $\neg A$  in  $\leftarrow Q$ .

- Let  $N$  be a negative query and  $L = p(\bar{t})$ .  
 $S$  is empty<sup>1</sup>.

Let  $(N, C)$  be a reselection in  $K$  where  $N \equiv \leftarrow Q$ .

An SLDNFA-extension of  $K$  using reselection  $(N, C)$  is the prederivation  $K' = ((\theta_1, \dots, \theta_n, \epsilon), T', \alpha')$ , such that  $T'$  is obtained from  $T$  by adding one new descendant  $N'$  to  $N$  and labeling the arc from  $N$  to  $N'$  with  $C$  as applied resolventus.  $N'$  is a negative query  $\leftarrow Q'$ , which is derived as follows. Recall that by definition of reselection,  $N$  has a selected atom  $A$  appearing in  $\leftarrow Q$ .

- If  $A$  is nonabducible, then  $\alpha'$  extends  $\alpha$  by marking the variables of a standardized apart variant  $C'$  of  $C$  negatively.  $\leftarrow Q'$  is derived from  $\leftarrow Q$  and  $C'$  by negative resolution on  $A$ .
- If  $A$  is abducible, then  $\alpha' = \alpha$  and  $\leftarrow Q'$  is derived from  $\leftarrow Q$  and  $C$  by negative resolution on  $A$ .

In [8], the following is proven: two different queries of an SLDNFA-derivation  $K$  that do not occur in the same branch may share positive but no negative variables; positive queries contain only positive variables; the substitutions  $\theta_1, \dots, \theta_n$  of  $K$  contain only positive variables.

**Definition 3.10 (finitely failed derivation)** *An SLDNFA-derivation  $K$  is finitely failed if  $K$  contains a positive query that contains a nonabducible atom without resolventi w.r.t.  $P^A$  or if  $K$  contains the empty negative query.*

A negative query  $N$  in an SLDNFA-derivation  $K$  is called *completed* iff  $N$  has a selected literal  $L$  and either  $L$  is a negative literal or  $L$  is an atom such that each resolventus of  $L$  w.r.t.  $P^A$  and  $K$  is an applied resolventus of  $N$ .

**Definition 3.11 (SLDNFA-refutation)** *An SLDNFA-refutation  $K$  for a query  $\leftarrow Q$  is an SLDNFA-derivation  $K$  for  $\leftarrow Q$  such that all positive leaves contain only abducible atoms and all negative queries are completed or they have no selected literal and contain an irreducible equality atom.*

In [8], it is shown in detail how to extract abductive solutions from an SLDNFA-refutation. Roughly, an abductive solution is extracted from an SLDNFA-refutation by completing the abduced atoms and adding the irreducible equality atoms as constraints (without going into detail, we mention that more general answers can be derived, as shown in [8]).

In the following section, we study termination under SLDNFA-execution w.r.t. an arbitrary selection rule. However, we put some natural conditions on the selection in SLDNFA, namely:

- no abducible atom is selected in a positive query (by Definition 3.7 of selection),
- no negative literal containing negative variables is selected in a negative query (by Definition 3.9 of SLDNFA-derivation: the selection is *safe*).
- no irreducible equality atom is selected in a negative query (this is a reasonable condition, since the descendant in that case would be identical to the negative query).

For the rest of this paper, we consider SLDNFA-derivations using a selection rule satisfying the above conditions.

We illustrate the above concepts with a small fault diagnosis problem of [8].

**Example 3.1** *A faulty lamp problem is caused by a broken lamp or by a power failure of a circuit without backup, that is, a loaded battery. The only circuit with battery is c1; its battery is b1. A battery is unloaded iff one of its energy cells is dry. This is formalized in  $P^{\{broken/1, power\_failure/1, dry\_cell/1\}}$ .*

$lamp(l1)$	$\leftarrow$	
$battery(c1, b1)$	$\leftarrow$	
$faulty\_Lamp$	$\leftarrow$	$lamp(X), broken(X)$
$faulty\_Lamp$	$\leftarrow$	$power\_failure(X), \neg backup(X)$
$backup(X)$	$\leftarrow$	$battery(X, Y), \neg unloaded(Y)$
$unloaded(X)$	$\leftarrow$	$dry\_cell(X)$

---

<sup>1</sup> $K$  and  $K'$  differ by the fact that  $N$  in  $K$  has no selected literal, whereas  $N$  in  $K'$  has the selected literal  $L$ . Branches of the failure tree below  $N$  are added in later stages, when  $N$  is reselected.



As noted in [7] in the case of definite programs (note that in that case acyclicity is also called recurrency), the condition that the level mapping decreases from clause heads to clause bodies, is used for two different purposes: (1) in (mutually) recursive calls, to ensure termination of (mutually) recursive procedures and (2) in non (mutually) recursive calls, to ensure that non (mutually) recursive procedures are called with terminating queries. Although a decreasing of the level mapping is apparently essential for the first purpose, this is not the case for the second purpose, since a weaker condition can be adopted to ensure that nonrecursive procedures are properly called. For the case of definite programs (hence recurrency), this weaker condition is given in [4]; namely, semi-recurrency. It is shown in [4] that semi-recurrency is equivalent to recurrency. We follow the same approach as [4] for the case of acyclicity.

**Definition 4.5 (semi-acyclic program)** *Let  $|\cdot|$  be a level mapping for  $P^A$ .  $P^A$  is semi-acyclic w.r.t.  $|\cdot|$  iff  $\forall H \leftarrow L_1, \dots, L_n \in \text{Ground}_\Sigma(P^A)$  and  $\forall i \in \{1, \dots, n\}$ :*

$$\begin{aligned} |H| > |L_i| & \text{ if } \text{Rel}(H) \simeq \text{Rel}(L_i), \\ |H| \geq |L_i| & \text{ if } \text{Rel}(H) \sqsupseteq \text{Rel}(L_i). \end{aligned}$$

$P^A$  is semi-acyclic iff there exists a level mapping  $|\cdot|$  such that  $P^A$  is semi-acyclic w.r.t.  $|\cdot|$ .

Note that abducible predicates do not occur in the head of a clause and cannot be recursive. So, in the search for a level mapping such that  $P^A$  is semi-acyclic, it can be safely assumed that abducible literals are assigned the value 0.

**Proposition 4.1** *Let  $P^A$  be a program.  $P^A$  is acyclic iff  $P^A$  is semi-acyclic. In particular, if  $P^A$  is semi-acyclic w.r.t. a level mapping  $|\cdot|$ , there is a level mapping  $|\cdot|^*$  such that  $P^A$  is acyclic w.r.t.  $|\cdot|^*$  and all queries  $\leftarrow Q$  that are bounded w.r.t.  $|\cdot|$  are also bounded w.r.t.  $|\cdot|^*$ .*

**Proof** The proof is similar to the proof of [4, Lemma 4.4] (for the equivalence of recurrency and semi-recurrency).

$\Rightarrow$ : Trivial.

$\Leftarrow$ : Let  $|\cdot|$  be a level mapping such that  $P^A$  is semi-acyclic w.r.t.  $|\cdot|$ . Let  $l$  be a mapping from  $\Sigma^p$  to  $\mathbb{N}$  such that  $l(p) > l(q)$  if  $p \sqsupseteq q$  and  $l(p) = l(q)$  if  $p \simeq q$ . Note that such a mapping exists since  $\Sigma^p$  is finite and  $\sqsupseteq$  is an equivalence relation. Define  $|\cdot|^* : B_P \cup \neg B_P \rightarrow \mathbb{N}$  as follows:

$$L \mapsto |L|^* = |L| + l(\text{Rel}(L)).$$

Then it is easy to verify that  $P^A$  is acyclic w.r.t.  $|\cdot|^*$ .

From the above proof it also follows that, for all  $L \in B_P^E \cup \neg B_P^E$  if  $L$  is bounded by  $k$  w.r.t.  $|\cdot|$ , then  $L$  is bounded by  $k + l(\text{Rel}(L))$  w.r.t.  $|\cdot|^*$ .  $\square$

In [3] it was proven that for an acyclic general program  $P$  and a bounded query  $\leftarrow Q$ , all SLDNF-derivations of  $\leftarrow Q$  in  $P$  are finite. In the following example we show that this is not the case for SLDNFA.

**Example 4.1** *Consider the following program:*

$$P^{\{p/1\}} : \begin{cases} r & \leftarrow p(X), \neg q(X) \\ q(X) & \leftarrow p(f(X)) \end{cases}$$

*and the query  $\leftarrow \neg r, p(X)$ . Consider the following level mapping  $|\cdot|$  for  $P^{\{p/1\}}$ :  $|r| = 2$ ,  $|q(t)| = 1$  and  $|p(t)| = 0$  for all  $t \in U_P$  (we assume that  $\Sigma$  contains at least one constant symbol). Then,  $P^{\{p/1\}}$  is acyclic w.r.t.  $|\cdot|$  and the query  $\leftarrow \neg r, p(X)$  is bounded by 2 w.r.t.  $|\cdot|$ .*

*An SLDNFA-derivation for  $\leftarrow \neg r, p(X)$  in  $P^{\{p/1\}}$  is shown in Figure 2. Note that the derivation is not finite, since there is an infinitely branching (negative) node.*



where  $C, C', C''$  are clauses of  $P^A$ , is called a path from  $\leftarrow Q$  in  $P^A$ .

If there is a subpath (possibly of length 0) in a path from  $\leftarrow Q$  in  $P^A$  starting in  $*sp$  and ending in  $*sq$ , where  $*, *' \in \{+, -\}$  and  $sp, sq$  are signed predicates of  $\Sigma^p \cup \{start\}$ , then we denote this with  $*sp \rightarrow^{Q,P,s} *'sq$ . If in this subpath from  $*sp$  to  $*'sq$ , no reduction of the sort  $\rightarrow_s$  occurs, then we denote this with  $*sp \rightarrow^{Q,P} *'sq$  (note that in this case  $* = *'$ ).

We introduce some notation. Let  $K = ((\theta_1, \dots, \theta_n), T, \alpha)$  be an SLDNFA-derivation. A positive, resp. negative, query  $N$  in  $T$  will be prefixed with  $+$ , resp.  $-$ ; e.g.  $+N$ , resp.  $-N$ . If a query  $N$  (positive or negative) in  $T$  contains a literal  $p(\bar{t})$ , resp.  $\neg p(\bar{t})$ , then this will be denoted as  $N(p)$ , resp.  $N(\neg p)$ . If a node  $N$  is an ancestor of a node  $M$  in  $T$ , then this will be denoted with  $N \rightsquigarrow M$ .

**Proposition 4.2** *Let  $P^A$  be a program and  $\leftarrow Q$  be a query. Let  $K = ((\theta_1, \dots, \theta_n), T, \alpha)$  be an SLDNFA-derivation of  $\leftarrow Q$  in  $P^A$ . Let  $*M_1(sp_1)$  and  $*M_2(sp_2)$  be nodes in  $T$ , with  $*, *' \in \{+, -\}$  and  $sp_1, sp_2$  signed predicates of  $\Sigma^p \setminus \{=\}$ .*

*If  $*M_1(sp_1) \rightsquigarrow *'M_2(sp_2)$ , then there exists a predicate  $q \in (\Sigma^p \setminus \{=\}) \cup \{start\}$  such that  $*q \rightarrow^{Q,P} *sp_1$  and  $*q \rightarrow^{Q,P,s} *'sp_2$ .*

**Proof** Let  $T'$  be the tree obtained from  $T$  by adding the positive query  $\leftarrow start$  as parent of  $\leftarrow Q$  in  $T$  (so  $\leftarrow start$  is the root of  $T'$ ). Then,  $K' = ((\theta_1, \dots, \theta_n), T', \alpha)$  is an SLDNFA-derivation of  $\leftarrow start$  in  $P^A \cup \{\leftarrow Q\}$ . Note that the root of  $T'$  is a query consisting of one atom, namely  $start$ . Also, each positive, resp. negative, query which is a direct descendant of a negative, resp. positive, query consists of one atom (see Definition 3.9 of SLDNFA-derivation).

Given is that  $*M_1(sp_1) \rightsquigarrow *'M_2(sp_2)$  in  $T$ , hence also in  $T'$ . Define  $\bar{+} = -$  and  $\bar{-} = +$ . Let  $*N$  be the highest ancestor of  $*M_1(sp_1)$  in  $T'$  such that there are no queries  $\bar{*}W$  on the branch from  $*N$  to  $*M_1(sp_1)$ . So, the parent of  $*N$  is of the form  $\bar{*}V$ . Note that, by the remark above,  $*N$  consists of one atom, say  $q(\bar{t})$ . By Definition 3.9 of SLDNFA-derivation, Definition 4.6 and the fact that  $sp_1$  is a signed predicate of  $\Sigma^p \setminus \{=\}$ , it is easy to see that  $*q \rightarrow^{Q,P} *sp_1$ . We prove that  $*q \rightarrow^{Q,P,s} *'sp_2$ . Let  $*N_2$  be the highest ancestor of  $*M_2(sp_2)$  in  $T'$  such that there are no queries  $\bar{*}'W$  on the branch from  $*N_2$  to  $*M_2(sp_2)$ . So, the parent of  $*N_2$  is of the form  $\bar{*}'V$ . If  $*N_2 = *N$ , then  $*' = *$  and  $*q \rightarrow^{Q,P} *sp_2$ , hence  $*q \rightarrow^{Q,P,s} *'sp_2$ . Suppose that  $*N_2 \neq *N$ . Again, by the remark above,  $*N_2$  consists of one atom, say  $q_2(\bar{t})$  and  $*'q_2 \rightarrow^{Q,P} *'sp_2$ . Note that  $q_2 \in \Sigma^p \setminus \{=\}$ . The parent of  $*N_2(q_2)$  is of the form  $\bar{*}'N_3(\neg q_2)$ . Since  $\bar{*}'\neg q_2 \rightarrow_s *'q_2$ , we have that  $\bar{*}'\neg q_2 \rightarrow^{Q,P,s} *'sp_2$ . By inductively exploiting this argument (on  $\bar{*}'N_3(\neg q_2)$ ), we obtain that  $*q \rightarrow^{Q,P,s} \bar{*}'\neg q_2$ , hence  $*q \rightarrow^{Q,P,s} *'sp_2$ .  $\square$

Note that Proposition 4.2 does not hold if  $sp_1$  or  $sp_2$  are equal to  $=$ . This is because the selection of an atom in a negative query, might introduce an equality atom (by negative resolution) and this is not taken into account in the Definition 4.6 of reductions. But this is not a problem, since we will only be interested in the case that  $sp_1$  and  $sp_2$  are signed predicates of  $A$ , the set of abducible predicate symbols. The next definition introduces the syntactical notion of abductive recursive program and query. The concept of abductive *nonrecursive* program and query will be the additional requirement in our termination condition (Theorem 4.1).

**Definition 4.7 (abductive recursive program and query)** *Let  $P^A$  be a program and  $\leftarrow Q$  be a query. We call  $P^A \cup \{\leftarrow Q\}$  abductive recursive iff there are abducible predicates  $p_1, \dots, p_n \in A$  and predicates  $q_1, \dots, q_n \in \Sigma^p \setminus (A \cup \{=\})$ ,  $n \geq 1$ , such that*

$$\begin{array}{cccccccc} -q_1 & \rightarrow^{Q,P} & -p_1 & -q_2 & \rightarrow^{Q,P} & -p_2 & \dots & -q_n & \rightarrow^{Q,P} & -p_n \\ -q_1 & \rightarrow^{Q,P,s} & +p_2 & -q_2 & \rightarrow^{Q,P,s} & +p_3 & \dots & -q_n & \rightarrow^{Q,P,s} & +p_1 \end{array}$$

*We call  $P^A \cup \{\leftarrow Q\}$  abductive nonrecursive iff  $P^A \cup \{\leftarrow Q\}$  is not abductive recursive.*

**Example 4.2** *Recall the program and query of Example 4.1.  $P^{\{p/1\}} \cup \{\leftarrow \neg r, p(X)\}$  is abductive recursive since  $\neg r \rightarrow^{Q,P} \neg p$  and  $\neg r \rightarrow^{Q,P,s} +p$ .*

For two more (slightly more complicated) examples of abductive programs and queries with infinite SLDNFA-derivations, we refer to the appendix (Examples A.1 and A.2).

The following is a corollary to Proposition 4.2. It gives the meaning of the notion of abductive recursive program and query in terms of SLDNFA-derivations.

**Corollary 4.1** *Let  $P^A$  be a program and  $\leftarrow Q$  be a query. Suppose there is an SLDNFA-derivation  $K = ((\theta_1, \dots, \theta_n), T, \alpha)$  of  $\leftarrow Q$  in  $P^A$  with nodes  $-N_1(p_1), +M_1(p_2), -N_2(p_2), +M_2(p_3), \dots, -N_n(p_n), +M_n(p_1)$  in  $T$  with  $n \geq 1$  and  $p_1, p_2, \dots, p_n \in A$ , such that*

$$-N_1(p_1) \rightsquigarrow +M_1(p_2) \quad -N_2(p_2) \rightsquigarrow +M_2(p_3) \quad \dots \quad -N_n(p_n) \rightsquigarrow +M_n(p_1)$$

Then,  $P \cup \{\leftarrow Q\}$  is abductive recursive.

**Proof** Note that  $=$  is not an abducible predicate, so by applying Proposition 4.2, there are predicates  $q_1, \dots, q_n \in (\Sigma^p \setminus \{=\}) \cup \{start\}$  such that

$$\begin{array}{ccccccc} -q_1 & \rightarrow^{Q,P} & -p_1 & -q_2 & \rightarrow^{Q,P} & -p_2 & \dots & -q_n & \rightarrow^{Q,P} & -p_n \\ -q_1 & \rightarrow^{Q,P,s} & +p_2 & -q_2 & \rightarrow^{Q,P,s} & +p_3 & \dots & -q_n & \rightarrow^{Q,P,s} & +p_1 \end{array}$$

Note then that  $-start$  does not occur in a path from  $P$  in  $\leftarrow Q$ , so  $q_i \neq start$  for all  $i \in \{1, \dots, n\}$ . Also, for  $p$  an abducible predicate,  $-p$  cannot be reduced further, so  $q_i \notin A$  for all  $i \in \{1, \dots, n\}$ .  $\square$

We are now able to formulate and prove the main theorem of this paper.

**Theorem 4.1** *Let  $P^A$  be a semi-acyclic program w.r.t. a level mapping  $|\cdot|$ , and let  $\leftarrow Q$  be a bounded query w.r.t.  $|\cdot|$ . Suppose that  $P^A \cup \{\leftarrow Q\}$  is abductive nonrecursive. Then,  $P^A$  is SLDNFA-terminating w.r.t.  $\leftarrow Q$ .*

**Proof** We may assume that  $P^A$  is acyclic w.r.t.  $|\cdot|$  (if  $P^A$  is not acyclic w.r.t.  $|\cdot|$ , then we take the level mapping  $|\cdot|^*$  defined in the proof of Proposition 4.1 and  $P^A$  is acyclic w.r.t.  $|\cdot|^*$ ; note that  $\leftarrow Q$  will also be bounded w.r.t.  $|\cdot|^*$ ). Let  $k \in \mathbb{N}$  such that  $\leftarrow Q$  is bounded by  $k$ .

Let  $|\cdot|_1$  be the mapping from  $B_P \cup \neg B_P$  to  $\mathbb{N} \cup \{-1\}$  obtained from  $|\cdot|$  as follows:  $|t = s|_1 = |t \neq s|_1 = -1$  for all  $t, s \in U_P$ , and  $|L|_1 = |L|$  for all  $L \in B_P \cup \neg B_P$  such that  $Rel(L) \neq =$ . Bounded literals and queries w.r.t.  $|\cdot|_1$  and their level under  $|\cdot|_1$  are defined like in Definition 4.3 for level mappings.

Note then that, since  $P^A$  is acyclic w.r.t.  $|\cdot|$ ,  $P^A$  is acyclic w.r.t.  $|\cdot|_1$  (note that acyclicity is only defined w.r.t. level mappings of the form of Definition 4.2, but Definition 4.4 of acyclicity can be extended in the obvious way for mappings like  $|\cdot|_1$ ). This is because equality is defined by one unit clause. Note also that, because  $\leftarrow Q$  is bounded by  $k$  w.r.t.  $|\cdot|$ ,  $\leftarrow Q$  is also bounded by  $k$  w.r.t.  $|\cdot|_1$ .

Consider the mapping  $|\cdot|_2$  from  $B_P \cup \neg B_P$  to  $(\mathbb{N} \cup \{-1\}) \times \{0, 1\}$  obtained from  $|\cdot|_1$  as follows:  $|B|_2 = (|B|_1, 0)$ ,  $|\neg B|_2 = (|\neg B|_1, 1)$  for all  $B \in B_P$ . We consider the lexicographical order  $<_2$  on  $(\mathbb{N} \cup \{-1\}) \times \{0, 1\}$ :  $(z, b) <_2 (z', b')$  iff  $z < z'$  or  $z = z'$  and  $b < b'$ . Note that  $<_2$  is well-founded. Bounded literals and queries w.r.t.  $|\cdot|_2$  and their level under  $|\cdot|_2$  are defined like in Definition 4.3 for level mappings.

Because  $P^A$  is acyclic w.r.t.  $|\cdot|_1$ ,  $P^A$  is acyclic w.r.t.  $|\cdot|_2$ . Also, because  $\leftarrow Q$  is bounded by  $k$  w.r.t.  $|\cdot|_1$ ,  $\leftarrow Q$  is bounded by  $(k, 1)$  w.r.t.  $|\cdot|_2$ .

Let  $\prec_2$  be the multiset ordering on finite multisets of elements of  $(\mathbb{N} \cup \{-1\}) \times \{0, 1\}$  obtained from  $((\mathbb{N} \cup \{-1\}) \times \{0, 1\}, <_2)$ ; i.e.  $\prec_2$  is the transitive closure of  $\prec'_2$  with

$$X \prec'_2 Y \text{ iff } X = Y - \{(z, b)\} \cup Z \text{ for some } (z, b) \in Y \text{ and } Z \text{ such that } (z', b') <_2 (z, b) \text{ for } (z', b') \in Z,$$

where  $X, Y, Z$  are finite multisets of elements of  $(\mathbb{N} \cup \{-1\}) \times \{0, 1\}$ . Because  $<_2$  is well-founded, the multiset ordering  $\prec_2$  is well-founded (see [10]).

Let  $K$  be an SLDNFA-derivation with all queries bounded w.r.t.  $|\cdot|_2$  by  $(k, 1)$ . Let  $K'$  be an SLDNFA-extension of  $K$ . Let  $N_s$  be the selected node in  $K$ . For a node  $N$  in  $K$ , denote with  $N'$  its corresponding node in  $K'$ . So, in particular,  $N'_s$  is the node in  $K'$  corresponding to the selected node  $N_s$  in  $K$ . In the following, we prove that all queries in  $K'$  are also bounded by  $(k, 1)$ , and that  $|N'_d|_2 \prec_2 |N|_2$  or  $|N'_d|_2 = |N|_2$  for all nodes  $N$  in  $K$ . We also show the relationship between  $|N'_s|_2$  and  $|N'_d|_2$ , where  $N'_d$  is a descendant of  $N'_s$  in  $K'$  added in the SLDNFA-extension step. Let  $L_s$  be the selected literal in  $N_s$ . By definition of SLDNFA-extension,  $K'$  is obtained from  $K$  in one of the following ways:

1.  $N_s$  is a positive query and  $L_s$  is an atom  $p(\bar{t})$  with  $p$  nonabducible.

Suppose  $N_s = \leftarrow \mathbf{L}, L_s, \mathbf{K}$  (note that possibly  $\mathbf{L}$  or  $\mathbf{K}$  are empty). Let  $C = H \leftarrow L_1, \dots, L_n$  be a standardized apart variant of a program clause of  $P^A$  and let  $\theta$  be the mgu of  $H$  and  $L_s$  (note that we use positive resolution). Then  $N'_s = \leftarrow \mathbf{L}\theta, L_s\theta, \mathbf{K}\theta$  and the descendant of  $N'_s$  in  $K'$  is  $N'_d = \leftarrow \mathbf{L}\theta, L_1\theta, \dots, L_n\theta, \mathbf{K}\theta$ . For every node  $N$  in  $K$ , the corresponding node  $N'$  in  $K'$  is obtained from  $N$  by applying  $\theta$  on it:  $N' = N\theta$ .

It is easy to see that for every node  $N$  in  $K$ ,  $|N'_d|_2 \prec_2 |N|_2$  or  $|N'_d|_2 = |N|_2$ , since for every literal  $L$ ,  $|L\theta|_2 \leq_2 |L|_2$ .

We prove that  $|N'_d|_2 \prec_2 |N'_s|_2$ . Since  $P^A$  is acyclic w.r.t.  $|\cdot|_2$  and since  $L_s$  and hence also  $L_s\theta$  are bounded w.r.t.  $|\cdot|_2$ ,  $L_i\theta$  is bounded for all  $i \in \{1, \dots, n\}$  and  $|L_i\theta|_2 <_2 |L_s\theta|_2$ . Also, for all literals  $F$  in  $\mathbf{L}$  or  $\mathbf{K}$ ,  $|F\theta|_2 \leq_2 |F|_2$ . By definition of  $\prec_2$ ,  $|N'_d|_2 \prec_2 |N'_s|_2$ .

From the above it follows that, if all queries in  $K$  are bounded by  $(k, 1)$ , then also all queries in  $K'$  are bounded by  $(k, 1)$ .

2.  $N_s$  is a positive query and  $L_s$  is a negative literal,  $L_s = \neg A$ .

In this case, we have for all nodes  $N$  in  $K$  that the corresponding node  $N'$  is equal to  $N$ . So obviously,  $|N'_d|_2 = |N|_2$ .

$N'_s = N_s$  has one or two descendants: a negative query  $N'_{d1} = \leftarrow A$  and (possibly, if  $N_s \neq \leftarrow \neg A$ ) a positive query  $N'_{d2}$  obtained from  $N'_s$  by deleting  $\neg A$ . We prove that  $|N'_{d1}|_2 \prec_2 |N'_s|_2$  and  $|N'_{d2}|_2 \prec_2 |N'_s|_2$ . Suppose  $N_s = \leftarrow \mathbf{L}, \neg A, \mathbf{K}$  (note that possibly  $\mathbf{L}$  or  $\mathbf{K}$  are empty). By definition of  $|\cdot|_2$ ,  $|\neg A|_2 = (|A|_1, 1)$  and  $|A|_2 = (|A|_1, 0)$ . So,  $|\neg A|_2 >_2 |A|_2$  and hence,  $|N'_{d1}|_2 = \{( |A|_1, 0)\} \prec_2 |N'_s|_2$ . Also, it is easy to see that  $N'_{d2} = \leftarrow \mathbf{L}, \mathbf{K}$ , hence  $|N'_{d2}|_2 \prec_2 |N'_s|_2$ .

Again, it follows from above that, if all queries in  $K$  are bounded by  $(k, 1)$ , then also all queries in  $K'$  are bounded by  $(k, 1)$ .

3.  $N_s$  is a negative query and  $L_s$  is a negative literal,  $L_s = \neg A$ .

There are two possible extensions in this case (see below). In both cases we have for all nodes  $N$  in  $K$  that the corresponding node  $N'$  is equal to  $N$ . So obviously,  $|N'_d|_2 = |N|_2$ . Let us consider now the two possibilities for the descendant  $N'_d$  of  $N'_s$ :

- $N'_d$  is the positive query  $\leftarrow A$ .

We can apply the same reasoning as in the previous case for the descendant  $N'_{d1}$ .

- $N'_d$  is the negative query obtained from  $N_s = N'_s$  by deleting  $\neg A$ .

Again, the same reasoning as in the previous case for the descendant  $N'_{d2}$  can be applied.

Again, it follows from above that, if all queries in  $K$  are bounded by  $(k, 1)$ , then also all queries in  $K'$  are bounded by  $(k, 1)$ .

4.  $N_s$  is a negative query and  $L_s$  is an atom  $p(\bar{t})$  with  $p$  nonabducible and  $p \neq =$ .

Again, we have for all nodes  $N$  in  $K$  that the corresponding node  $N'$  is equal to  $N$ . So obviously,  $|N'_d|_2 = |N|_2$ .

Suppose  $N_s = \leftarrow \mathbf{L}, p(\bar{t}), \mathbf{K}$  (note that possibly  $\mathbf{L}$  or  $\mathbf{K}$  are empty). Let  $C = p(\bar{s}) \leftarrow L_1, \dots, L_n$  be a standardized apart variant of a program clause of  $P^A$ . Let  $E = E_+ \cup E_-$  be

the positive solved form of  $\bar{t} = \bar{s}$ . Then, the descendant  $N'_d$  of  $N_s = N'_s$  in  $K'$  is the negative query  $\leftarrow E_-(\mathbf{L}), E_+, E_-(L_1), \dots, E_-(L_n), E_-(\mathbf{K})$  (note that we use negative resolution). First, note that for all literals  $F$  from  $\mathbf{L}$  or  $\mathbf{K}$ ,  $|E_-(F)|_2 \leq_2 |F|_2$ . Also, note that, since all variables in  $C$  are marked as negative  $E(p(\bar{s})) = E_-(p(\bar{s}))$  and  $E(L_i) = E_-(L_i)$  for all  $i \in \{1, \dots, n\}$ . Hence, because  $P^A$  is acyclic,  $|p(\bar{t})|_2 \geq_2 |E(p(\bar{s}))|_2 = |E_-(p(\bar{s}))|_2 >_2 |E_-(L_i)|_2$  for all  $i \in \{1, \dots, n\}$ . Since we assumed that  $p \neq$ ,  $|p(\bar{t})|_2 \geq_2 (0, 0)$ . Because for all equality atoms  $\bar{t}' = \bar{s}'$  in  $E_+$ ,  $|\bar{t}' = \bar{s}'|_2 = (-1, 0)$ , we have that  $|p(\bar{t})|_2 >_2 |\bar{t}' = \bar{s}'|_2$ . So, putting all things together, we can conclude that  $|N'_d|_2 \prec_2 |N'_s|_2$ .

Again, it follows from above that, if all queries in  $K$  are bounded by  $(k, 1)$ , then also all queries in  $K'$  are bounded by  $(k, 1)$ .

5.  $N_s$  is a negative query and  $L_s$  is an atom  $p(\bar{t})$  with  $p$  abducible.

The same reasoning can be applied as in the previous case (note that  $p \neq$ , so again  $|p(\bar{t})|_2 \geq_2 (0, 0)$ ).

6.  $N_s$  is a negative query and  $L_s$  is a reducible equality atom  $\bar{t} = \bar{s}$ .

Again, we have for all nodes  $N$  in  $K$  that the corresponding node  $N'$  is equal to  $N$ . So obviously,  $|N'|_2 = |N|_2$ .

Suppose  $N_s = \leftarrow \mathbf{L}, \bar{t} = \bar{s}, \mathbf{K}$  (note that possibly  $\mathbf{L}$  or  $\mathbf{K}$  are empty). Note that there is only one clause, namely  $X = X \leftarrow$ , for equality in  $P^A$ . Let  $E = E_+ \cup E_-$  be the positive solved form of  $\bar{t} = \bar{s}$ . Then, the descendant  $N'_d$  of  $N_s = N'_s$  in  $K'$  is the negative query  $\leftarrow E_-(\mathbf{L}), E_+, E_-(\mathbf{K})$  (note that we use negative resolution). First, notice that for all literals  $F$  from  $\mathbf{L}$  or  $\mathbf{K}$ ,  $|E_-(F)|_2 \leq_2 |F|_2$ . So,  $|E_-(\mathbf{L})|_2 \prec_2 |\mathbf{L}|_2$  or  $|E_-(\mathbf{L})|_2 = |\mathbf{L}|_2$ , and the same holds for  $\mathbf{K}$  and  $E_-(\mathbf{K})$ . Concerning the selected equality atom  $\bar{t} = \bar{s}$ , we have that  $|\bar{t} = \bar{s}|_2 = (-1, 0)$ . And for every equality  $\bar{t}' = \bar{s}'$  in  $E_+$ , we also have that  $|\bar{t}' = \bar{s}'|_2 = (-1, 0)$ . So we can not conclude that  $|N'_d|_2 \prec_2 |N'_s|_2$  in this case, since it is possible that  $|E_-(\mathbf{L})|_2 = |\mathbf{L}|_2$  and  $|E_-(\mathbf{K})|_2 = |\mathbf{K}|_2$  and  $E_+$  consists of 1 or more equality atoms.

Note that we can conclude however, that, if all queries in  $K$  are bounded by  $(k, 1)$ , then also all queries in  $K'$  are bounded by  $(k, 1)$ .

We will now prove that there is no infinite sequence of SLDNFA-extensions of  $\leftarrow Q$  in  $P^A$ , or equivalently, the limit of a sequence of SLDNFA-extensions consists of a finite tree.

Let  $K_0 = (( ), T_0, \alpha_0)$  be the SLDNFA-derivation of length 0 of  $\leftarrow Q$  in  $P^A$ . Let  $K_0, K_1, \dots, K_n, \dots$  be a sequence of SLDNFA-extensions starting with  $K_0$ . In the sequel, we will sometimes identify an SLDNFA-derivation  $K_i = ((\theta_1, \dots, \theta_i), T_i, \alpha_i)$  with its tree  $T_i$ . Note that the only query  $\leftarrow Q$  in  $K_0$  is bounded w.r.t.  $|\cdot|_2$  by  $(k, 1)$ . So, by the previous, for all  $n \geq 0$ , all queries in  $K_n$  are bounded by  $(k, 1)$ . We prove the following:

- (a) In the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinite branch with its tail consisting only of positive queries.

Suppose that, in the limit, there is an infinite branch with its tail consisting of only positive queries. Then a contradiction follows from point 1 above, the fact that all queries are bounded by  $(k, 1)$ , and the fact that  $\prec_2$  is well-founded.

- (b) In the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinite branch consisting of an infinite number of positive and an infinite number of negative queries.

Suppose that, in the limit, there is an infinite branch consisting of an infinite number of positive and an infinite number of negative queries. This means that in this branch, after a finite number of positive queries, there is a negative query, and after a finite number of negative queries, there is a positive query. In point 1, we proved that by adding a positive query as descendant of a positive query in a SLDNFA-extension step, the level mapping  $|\cdot|_2$  decreases w.r.t.  $\prec_2$ . Also, in point 2, resp. point 3, we proved that the level mapping decreases when we add a negative, resp. positive, query as descendant of a positive, resp. negative, query. Also, the selection of a nonequality atom in a negative query resulting in

a negative descendant, gives a decrease of the level mapping  $|\cdot|_2$  w.r.t.  $\prec_2$  (points 4 and 5). The only problem occurs when an equality atom is selected in a negative query (point 6): in that case, the level  $|\cdot|_2$  of the literals in the query will not increase, but a finite number of equality atoms (with level  $(-1, 0)$ ) might be added to the query. But, this can only occur a finite number of times before a negative literal  $\neg A$  in the negative query is selected (which results in a decrease since  $|A|_2 <_2 |\neg A|_2$ , see point 3). Hence, since all queries are bounded by  $(k, 1)$  and  $\prec_2$  is well-founded, we can conclude that no branch with an infinite number of positive and an infinite number of negative queries exists.

- (c) In the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinitely branching node. Suppose that in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is an infinitely branching node. We prove a contradiction with the fact that  $P^A \cup \{\leftarrow Q\}$  is abductive nonrecursive. Note first that the tree can only be infinitely branching in a negative query with a selected abducible atom  $p_1(\bar{t}_1)$ , and there are infinitely many positive queries with abducible  $p_1$ -atoms. Because of the previous two proven facts (points (a) and (b) above), there can only be an infinite number of positive queries as the result of infinitely branching nodes. So, in the limit there must be a negative query  $-N_2(p_2)$  with a selected abducible atom  $p_2(\bar{t}_2)$  which is infinitely branching and from which a positive query  $+M_2(p_1)$  with an abducible  $p_1$ -atom descends; i.e.  $-N_2(p_2) \rightsquigarrow +M_2(p_1)$  (note that there must be a tree  $T_n$  which already contains the nodes (corresponding to)  $-N_2(p_2)$  and  $+M_2(p_1)$  with  $-N_2(p_2) \rightsquigarrow +M_2(p_1)$ ). There are two possibilities: either  $p_2 = p_1$  and by Corollary 4.1 we obtain that  $P^A \cup \{\leftarrow Q\}$  is abductive recursive which gives a contradiction, or  $p_2 \neq p_1$  and we apply the same reasoning on  $p_2$ . That is, in the limit there is a negative query  $-N_3(p_3)$  with a selected abducible atom  $p_3(\bar{t}_3)$  which is infinitely branching and from which a positive query  $+M_3(p_2)$  with an abducible  $p_2$ -atom descends; i.e.  $-N_3(p_3) \rightsquigarrow +M_3(p_2)$  (note that there must be a tree  $T_m$  which already contains the nodes (corresponding to)  $-N_3(p_3)$  and  $+M_3(p_2)$  with  $-N_3(p_3) \rightsquigarrow +M_3(p_2)$ ; we continue with the tree  $T_{max(n,m)}$ ). We have three possibilities now: either  $p_3 = p_1$  or  $p_3 = p_2$ , and by Corollary 4.1 we obtain that  $P^A \cup \{\leftarrow Q\}$  is abductive recursive which gives a contradiction, or  $p_3 \neq p_1$  and  $p_3 \neq p_2$  and we apply the same reasoning on  $p_3$ . We continue this reasoning. Since there are finitely many predicates, and in particular finitely many abducible predicates, eventually, we obtain a contradiction with the fact that  $P^A \cup \{\leftarrow Q\}$  is abductive nonrecursive. So, we can conclude that in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinitely branching node.

- (d) In the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinite branch with its tail consisting only of negative queries.

Suppose that in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is an infinite branch with its tail consisting only of negative queries. We prove a contradiction. Note that, in the tail of this infinite branch, always positive literals are selected. Because of points 4 and 5, if a nonequality atom is selected in a negative query, there is a decrease between the level  $|\cdot|_2$  of the selected negative query and the added descendant. If a reducible equality atom is selected (see point 6), possibly some equality-atoms are added and a substitution is applied on the remaining literals in the negative query. So, if we have in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  an infinite branch with its tail consisting of only negative queries, then in this tail, there are a finite number of selections of nonequality atoms, and an infinite number of selections of reducible equality atoms. That is, this tail has itself a tail where only reducible equality atoms are selected. Note that, after resolving a reducible equality atom, the added equality atoms (those from  $E_+$ ) are all irreducible. Note also that the application of a unifier of the form  $E_-$  on an irreducible equality atoms will not turn it into a reducible equality atom (only unifiers which instantiate positive variables, like positive unifiers, can make irreducible equality atoms reducible). So, there has to be an infinite number of positive unifiers, obtained by positive resolution in positive queries. This means that there has to be an infinite number of positive queries in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$ . Because of the previous three proven facts (points (a), (b) and (c)) this cannot be the case, so a contradiction follows and we can conclude that in the limit of the sequence  $K_0, K_1, \dots, K_n, \dots$  there is no infinite branch with its tail consisting only of

negative queries.

From points (a), (b), (c) and (d) above, it follows that in the limit of a sequence  $K_0, K_1, \dots, K_n, \dots$  of SLDNFA-extensions, the tree is finite. Or, a sequence of SLDNFA-extensions of  $\leftarrow Q$  in  $P^A$  is finite. □

**Example 4.3** Recall the program and query of Example 3.1. Let  $|\cdot|$  be the following level mapping:  $|broken(t)| = |power\_failure(t)| = |dry\_cell(t)| = 0$ ,  $|lamp(t)| = |battery(t, s)| = |unloaded(t)| = 1$ ,  $|backup(t)| = 2$  and  $|faulty\_lamp| = 3$ , with  $t, s \in U_P$ .  $P$  is acyclic w.r.t.  $|\cdot|$  and the query  $\leftarrow faulty\_lamp$  is bounded by 3 w.r.t.  $|\cdot|$ . One can easily verify that  $P \cup \{\leftarrow faulty\_lamp\}$  is abductive nonrecursive, so,  $P$  SLDNFA-terminates w.r.t.  $\leftarrow faulty\_lamp$ . In Example 4.5, we prove that  $P$  SLDNFA-terminates w.r.t. all queries.

**Example 4.4** Recall the program  $P^{\{p/1\}}$  of Example 4.1. As we already noted in Example 4.2,  $P^{\{p/1\}} \cup \{\leftarrow \neg r, p(X)\}$  is abductive recursive. Consider next the query  $\leftarrow r$ . Note that  $P^{\{p/1\}} \cup \{\leftarrow r\}$  is abductive nonrecursive. By the fact that  $P^{\{p/1\}}$  is acyclic and  $\leftarrow r$  is bounded by 2 w.r.t. the level mapping proposed in Example 4.1, we have that  $P^{\{p/1\}}$  SLDNFA-terminates w.r.t.  $\leftarrow r$ .

Note that, in case  $A = \emptyset$ , a program  $P^{\{ \}}$  and query  $\leftarrow Q$  are trivially abductive nonrecursive. So, the termination condition of Theorem 4.1 boils down to the termination condition of Apt and Bezem [3] for SLDNF in that case.

Note also that a definite program and query are trivially abductive nonrecursive. So, termination in that case is ensured by the (semi-)acyclicity of the program (note that for definite programs, the notion of (semi-)acyclicity is called (semi-)recurrency in [4]) and boundedness of the query.

The notion of abductive recursivity of Definition 4.7 depends on the program *and* the query in question. We refer to Example 4.4 where a program and two queries are given such that the program and first query are abductive recursive (and do not SLDNFA-terminate) whereas the program and second query are abductive recursive (and do SLDNFA-terminate). However, for some programs, a condition can be stated on the program only, which we will also call abductive nonrecursivity (but now with no reference to a particular query; see Definition 4.9), such that the program  $P^A$  is abductive nonrecursive (Definition 4.9) iff for all queries  $\leftarrow Q$ ,  $P^A \cup \{\leftarrow Q\}$  is abductive nonrecursive (Definition 4.7).

**Definition 4.8 (paths II)** A reduction sequence (see Definition 4.6) of the form

$$\cdot(\rightarrow_s). \rightarrow_C \cdot(\rightarrow_s). \rightarrow_{C'} \dots (\rightarrow_s). \rightarrow_{C''} \dots$$

where  $C, C', C''$  are clauses of  $P^A$ , is called a path in  $P^A$ .

If there is a subpath (possibly of length 0) in a path in  $P^A$  starting in  $*sp$  and ending in  $*'sq$ , where  $*, *' \in \{+, -\}$  and  $sp, sq$  are signed predicates of  $\Sigma^p$ , then we denote this with  $*sp \rightarrow^{P,s} *'sq$ . If in this subpath from  $*sp$  to  $*'sq$ , no reduction of the sort  $\rightarrow_s$  occurs, then we denote this with  $*sp \rightarrow^P *'sq$  (note that in this case  $* = *'$ ).

**Definition 4.9 (abductive recursive program)** We call the program  $P^A$  abductive recursive iff there are abducible predicates  $p_1, \dots, p_n \in A$  and predicates  $q_1, \dots, q_n \in \Sigma^p \setminus (A \cup \{=\})$ ,  $n \geq 1$ , such that

$$\begin{array}{ccccccc} -q_1 & \rightarrow^P & -p_1 & -q_2 & \rightarrow^P & -p_2 & \dots & -q_n & \rightarrow^P & -p_n \\ -q_1 & \rightarrow^{P,s} & +p_2 & -q_2 & \rightarrow^{P,s} & +p_3 & \dots & -q_n & \rightarrow^{P,s} & +p_1 \end{array}$$

We call  $P^A$  abductive nonrecursive iff  $P^A$  is not abductive recursive.

The above definition can be understood in terms of the dependency relation as follows. Note that, for  $*, *' \in \{+, -\}$  with  $\overline{+} = -$  and  $\overline{-} = +$ , and for all predicates  $p, q$  we have:  $*q \rightarrow^P *'p$  iff  $\overline{*}q \rightarrow^P \overline{*}'p$  and  $*q \rightarrow^{P,s} *'p$  iff  $\overline{*}q \rightarrow^{P,s} \overline{*}'p$ . So, a statement of the form “ $-q \rightarrow^P -p$ ” in the above definition can be replaced by “ $q$  depends on  $p$  in  $P$  and there is a dependency of  $q$  on  $p$  which consists of positive arcs only”. A statement of the form “ $-q \rightarrow^{P,s} +p$ ” in the above definition can be replaced by “ $q$  depends on  $p$  in  $P$  and there is a dependency of  $q$  on  $p$  which consists of an odd number of negative arcs”.

**Proposition 4.3** *The program  $P^A$  is abductive nonrecursive iff for all queries  $\leftarrow Q$ ,  $P^A \cup \{\leftarrow Q\}$  is abductive nonrecursive. Or,  $P^A$  is abductive recursive iff there is a query  $\leftarrow Q$  such that  $P^A \cup \{\leftarrow Q\}$  is abductive recursive.*

**Proof** Suppose that there is a query  $\leftarrow Q$  such that  $P^A \cup \{\leftarrow Q\}$  is abductive recursive. Then we prove that  $P^A$  is abductive recursive. This follows from the fact that for every path  $Path$  from  $\leftarrow Q$  in  $P^A$ , the reduction sequence obtained from  $Path$  by deleting the first element and reduction, namely  $+start \rightarrow_Q$ , is a path in  $P^A$  and from the Definitions 4.7 and 4.9.

Suppose next that  $P^A$  is abductive recursive. We prove that there is a query  $\leftarrow Q$  such that  $P^A \cup \{\leftarrow Q\}$  is abductive recursive. There are abducible predicates  $p_1, \dots, p_n \in A$  and predicates  $q_1, \dots, q_n \in \Sigma^p \setminus (A \cup \{=\})$ ,  $n \geq 1$ , such that

$$\begin{array}{ccccccc} -q_1 & \xrightarrow{P} & -p_1 & -q_2 & \xrightarrow{P} & -p_2 & \dots & -q_n & \xrightarrow{P} & -p_n \\ -q_1 & \xrightarrow{P,s} & +p_2 & -q_2 & \xrightarrow{P,s} & +p_3 & \dots & -q_n & \xrightarrow{P,s} & +p_1 \end{array}$$

Let  $\leftarrow Q$  be the following query  $\leftarrow \neg q_1(\bar{t}_1), \dots, \neg q_n(\bar{t}_n)$  (with  $\bar{t}_i$ ,  $i \in \{1, \dots, n\}$ , an arbitrary sequence of terms with length equal to the arity of  $q_i$ ). Then, for all  $i \in \{1, \dots, n\}$ ,  $+start \rightarrow_Q +\neg q_i \rightarrow_s -q_i$  is a path from  $\leftarrow Q$  in  $P^A$ . Hence, together with the above, we have that

$$\begin{array}{ccccccc} -q_1 & \xrightarrow{Q,P} & -p_1 & -q_2 & \xrightarrow{Q,P} & -p_2 & \dots & -q_n & \xrightarrow{Q,P} & -p_n \\ -q_1 & \xrightarrow{Q,P,s} & +p_2 & -q_2 & \xrightarrow{Q,P,s} & +p_3 & \dots & -q_n & \xrightarrow{Q,P,s} & +p_1 \end{array}$$

and  $P^A \cup \{\leftarrow Q\}$  is abductive recursive. □

Hence, as a corollary to the main theorem (Theorem 4.1) and Proposition 4.3, we can state the following termination condition.

**Theorem 4.2** *Let  $P^A$  be a semi-acyclic program w.r.t. a level mapping  $|\cdot|$ . Suppose that  $P^A$  is abductive nonrecursive. Then, for all bounded queries  $\leftarrow Q$  w.r.t.  $|\cdot|$ ,  $P^A$  is SLDNFA-terminating w.r.t.  $\leftarrow Q$ .*

**Example 4.5** *Recall the program of Example 3.1. Note that the program is abductive nonrecursive. As we showed in Example 4.3, the program is acyclic w.r.t. the level mapping  $|\cdot|$  proposed in Example 4.3. Note also that all queries are bounded (by 3) w.r.t.  $|\cdot|$ . So, by Theorem 4.2, the program is SLDNFA-terminating w.r.t. all queries.*

## 5 Conclusions and future work

We presented an extension of the methods of Apt and Bezem [3] for proving termination of general logic programs executed under SLDNF, to the case of abductive general logic programs executed under SLDNFA [8]. Whereas the conditions of Apt and Bezem (i.e. acyclicity of the program and boundedness of the query) are also sufficient for proving termination of SLDNFA in the case of definite programs and queries, in the propositional case, as well as in the case of the empty set of abducible predicates, their conditions are not sufficient in general for proving SLDNFA-termination. We proposed a third, syntactical condition, namely abductive nonrecursivity of the program and query, which together with the acyclicity of the program and boundedness of the query imply termination of SLDNFA. By the best of our knowledge, this is the first work on termination of an abductive procedure for general logic programs.

Our termination condition (Theorem 4.1) is sufficient but not necessary. In particular, there are (semi-)acyclic programs and bounded queries which SLDNFA-terminate, but which are abductive recursive. Consider for instance the following program:

$$P^{\{p/1\}} : \begin{cases} r & \leftarrow p(X), \neg q(X) \\ q(X) & \leftarrow p(X) \end{cases}$$

and the query  $\leftarrow \neg r, p(X)$ . Notice the similarity with the program and query of Example 4.1 (which does not SLDNFA-terminate).  $P^{\{p/1\}}$  is acyclic and  $\leftarrow \neg r, p(X)$  is bounded w.r.t. the same level mapping as

in Example 4.1. Also,  $P^{\{p/1\}} \cup \{\leftarrow \neg r, p(X)\}$  is abductive recursive. But,  $P^{\{p/1\}}$  SLDNFA-terminates w.r.t.  $\leftarrow \neg r, p(X)$ . It remains a topic for future work to refine the notion of abductive recursivity and to find a necessary and sufficient condition for SLDNFA-termination. Another topic for future research is to study termination of SLDNFA w.r.t. a particular selection rule. Finally, we want to note that, besides abductive nonrecursivity, other conditions can be found which, together with acyclicity of the program and boundedness of the query, imply SLDNFA-termination. We refer to the discussion after Example 4.1 on page 9. There, we noted that the additional condition should ensure that, for every derivation of  $\leftarrow Q$  in  $P^A$ , the set of abduced atoms in the derivation is finite. This is, as we noted, trivially true in the propositional case. In the predicate case, this is for instance true if the set of abduced atoms in each derivation is a finite set of ground atoms (which could e.g. be derived from mode or type information). Further research in this direction and in combining the different conditions should be done.

## References

- [1] K. R. Apt and K. Doets. A new definition of SLDNF-resolution. *Journal of Logic Programming*, 18:177–190, 1994.
- [2] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of theoretical computer science, Vol. B*. Elsevier Science Publishers, 1990.
- [3] K.R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.
- [4] K.R. Apt and D. Pedreschi. Modular termination proofs for logic and pure Prolog programs. In *Advances in Logic Programming Theory*, pages 183–229. Oxford University Press, 1994.
- [5] L. Console, D. Theseider Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.
- [6] D. De Schreye and S. Decorte. Termination of logic programs: the never-ending story. *Journal of Logic Programming*, 19 & 20:199–260, may/july 1994.
- [7] D. De Schreye, K. Verschaetse, and M. Bruynooghe. A framework for analysing the termination of definite logic programs with respect to call patterns. In *Proc. FGCS'92*, pages 481–488, ICOT Tokyo, 1992. ICOT.
- [8] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.
- [9] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proc. of the European Conference on Artificial Intelligence*, 1992.
- [10] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 & 2):69–116, 1987.
- [11] K. Eshghi. Abductive planning with Event Calculus. In R.A. Kowalski and K.A. Bowen, editors, *Proc. of the International Conference on Logic Programming*, pages 562–579. MIT-press, 1988.
- [12] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. Declarative modelling of the operational behaviour of logic languages. *Theoretical Computer Science*, 69(3):289–318, 1989.
- [13] T.H. Fung and R. Kowalski. The iff proof procedure for abductive logic programming. *Journal of Logic Programming*, 33(2):151–165, 1997.
- [14] A. C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
- [15] A.C. Kakas and P. Mancarella. Database updates through abduction. In D. McLeod, R. Sacks-Davis, and H. Schek, editors, *Proc. of the 16th Very large Database Conference*, pages 650–661. Morgan Kaufmann, 1990.
- [16] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1987.
- [17] Lode R. Missiaen, Marc Denecker, and Maurice Bruynooghe. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, September 1995.
- [18] L.R. Missiaen. Localized abductive planning for robot assembly. In *Proceedings 1991 IEEE Conference on Robotics and Automation*, pages 605–610. IEEE Robotics and Automation Society, 1991.
- [19] K. Satoh and N. Iwayama. A Query Evaluation method for Abductive Logic Programming. In K.R. Apt, editor, *Proc. of the International Joint Conference and Symposium on Logic Programming*, 1992.
- [20] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of the IJCAI89*, page 1055, 1989.
- [21] F. Teusink. Using SLDFEA-resolution with abduction. In *Proc. of the ILPS'93 workshop on Logic Programming with Incomplete Information*, pages 35–47, 1993.

## A Some more examples

In this appendix two examples of abductive recursive programs and queries with infinite SLDNFA-derivations are given. Note that for both examples, a level mapping can be found such that the program is acyclic and the query is bounded w.r.t. that level mapping. Hence, both examples also illustrate the fact that the conditions for termination of SLDNF of Apt and Bezem [3] (namely acyclicity of the program and boundedness of the query) are not sufficient for proving termination of SLDNFA.

**Example A.1** *The following program and query are abductive recursive with a cycle consisting of two abducible predicates ( $n = 2$  in Definition 4.7). Notice the similarity with Example 4.1.*

$$P\{p_1/1, p_2/1\} : \begin{cases} r_1 & \leftarrow p_1(X), \neg q_1(X) \\ r_2 & \leftarrow p_2(X), \neg q_2(X) \\ q_1(X) & \leftarrow p_2(f(X)) \\ q_2(X) & \leftarrow p_1(f(X)) \end{cases}$$

*Consider the query  $\leftarrow \neg r_1, \neg r_2, p_1(X)$ . Then  $P\{p_1/1, p_2/1\} \cup \{\leftarrow \neg r_1, \neg r_2, p_1(X)\}$  is abductive recursive since  $\neg r_1 \rightarrow^{Q,P} \neg p_1$ ,  $\neg r_1 \rightarrow^{Q,P,s} \neg p_2$  and  $\neg r_2 \rightarrow^{Q,P} \neg p_2$ ,  $\neg r_2 \rightarrow^{Q,P,s} \neg p_1$ . Note that the SLDNFA-derivation of  $\leftarrow \neg r_1, \neg r_2, p_1(X)$  in  $P\{p_1/1, p_2/1\}$  (see Figure 3) is infinite.*

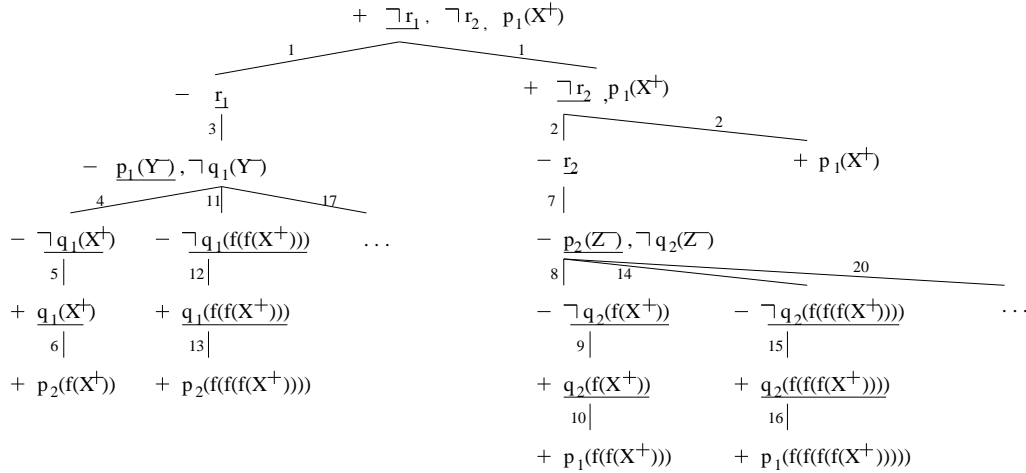


Figure 3: SLDNFA-derivation for  $\leftarrow \neg r_1, \neg r_2, p_1(X)$  in  $P\{p_1/1, p_2/1\}$ .

**Example A.2** *In this example we illustrate how an infinitely branching node in an SLDNFA-derivation can give rise to an infinite branch.*

$$P\{p/2\} : \begin{cases} r & \leftarrow p(X, Y), \neg q(X, Y) \\ q(f(X), f(Y)) & \leftarrow p(X, Y) \end{cases}$$

*Consider the query  $\leftarrow p(X, Y), \neg X = Y, \neg r$ . Then  $P\{p/2\} \cup \{\leftarrow p(X, Y), \neg X = Y, \neg r\}$  is abductive recursive since  $\neg r \rightarrow^{Q,P} \neg p$ ,  $\neg r \rightarrow^{Q,P,s} \neg p$ . The SLDNFA-derivation of  $\leftarrow p(X, Y), \neg X = Y, \neg r$  in  $P\{p/2\}$ , shown in Figure 4, contains an infinitely branching node and an infinite branch. Concerning the notation used in this figure: note that a mgu obtained in a positive resolution step is applied to all the nodes in the SLDNFA-derivation. When an application of such a positive unifier instantiates a leaf in the tree and enables the selection of an atom in that leaf, we write this instance together with the number of the corresponding positive resolution step under that leaf.*

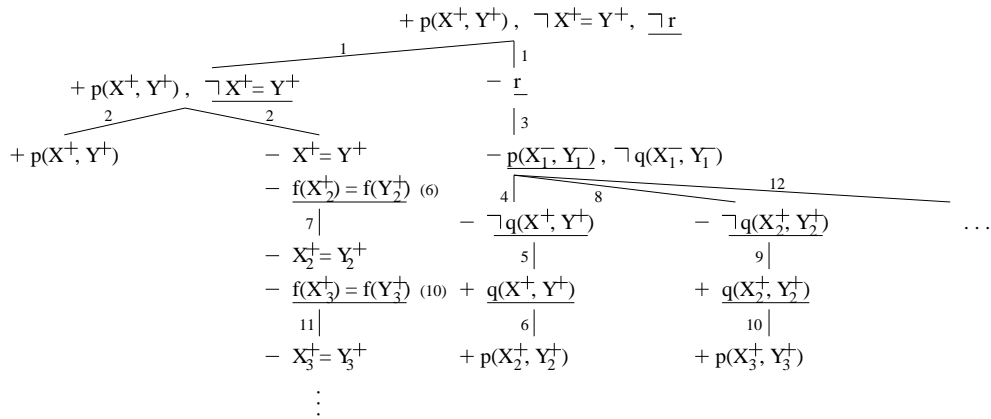


Figure 4: SLDNFA-derivation for  $\leftarrow p(X, Y), \neg X = Y, \neg r$  in  $P^{\{p/2\}}$ .