

Modular Termination Proofs for Prolog with Tabling

Sofie Verbaeten Konstantinos Sagonas Danny De Schreye

Report CW 279, September 1999



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

Modular Termination Proofs for Prolog with Tabling

Sofie Verbaeten Konstantinos Sagonas Danny De Schreye

Report CW 279, September 1999

Department of Computer Science, K.U.Leuven

Abstract

Tabling avoids many of the shortcomings of SLD(NF) execution and provides a more flexible and efficient execution mechanism for logic programs. In particular, tabled execution of logic programs terminates more often than execution based on SLD-resolution. One of the few works studying termination under a tabled execution mechanism is that of Decorte *et al.* They introduce and characterise two notions of universal termination of logic programs w.r.t. sets of queries executed under SLG-resolution, using the left-to-right selection rule; namely the notion of quasi-termination and the (stronger) notion of LG-termination.

This paper extends the results of Decorte *et al.* in two ways: (1) we consider a mix of tabled and Prolog execution, and (2) besides a characterisation of the two notions of universal termination under such a mixed execution, we also give modular termination conditions. From both practical and efficiency considerations, it is important to allow tabled and non-tabled predicates to be freely intermixed. This motivates the first extension. Concerning the second extension, it was already noted in the literature in the context of termination under SLD-resolution (by e.g. Apt and Pedreschi), that it is important for programming in the large to have modular termination proofs, i.e. proofs that are capable of combining termination proofs of separate programs to obtain termination proofs of combined programs.

Keywords : Logic Programming, Termination Analysis

CR Subject Classification : I.2.3, I.2.2, F.3.1

1 Introduction

The extension of SLD-resolution with a tabling mechanism [6, 18, 20], avoids many of the shortcomings of SLD(NF) execution and provides a more flexible and often considerably more efficient execution mechanism for logic programs. In particular, tabled execution terminates more often than execution based on SLD. So, if a program and query can be proven terminating under SLD-resolution (by one of the existing techniques surveyed in [7]), then they will also trivially terminate under SLG-resolution, the resolution principle of tabulation [6]. However, since there are programs and queries which terminate under SLG-resolution and not under SLD-resolution, more effective proof techniques can be found. This paper is one of the few works studying termination of tabled logic programs.

The ideas underlying tabling are very simple. Essentially, under tabled execution mechanism, answers for selected tabled atoms are stored in a table. When a variant of such an atom is recursively called, the selected atom is not resolved against program clauses, instead, all corresponding answers computed so far are looked up in the table and the corresponding answer substitutions are applied to the atom. This process is repeated for all subsequent computed answer substitutions that correspond to the atom.

We base our approach on the work of Decorte *et al* [10]. There, two notions of universal termination of a tabled logic program w.r.t. a set of queries executed under SLG-resolution using the left-to-right selection rule (called LG-resolution in the sequel) are introduced and characterised. Namely, the notion of quasi-termination and the (stronger) notion of LG-termination. We extend the results of [10] in two ways: (1) we consider a mix of LG-resolution and LD-resolution, i.e. a mix of tabled and Prolog execution, and (2) besides a characterisation of the two notions of universal termination under such a mixed execution schema, we also give modular termination conditions, i.e. conditions on two programs P and R , where P extends R , ensuring termination of the union $P \cup R$. The motivation for extension (1) will be given in the next section. There, examples from context-free grammar recognition and parsing are given, which show that, from the point of view of efficiency, it is important to allow tabled and non-tabled predicates to be freely intermixed. Extension (2) was already motivated in the literature in the context of termination under SLD-resolution (see for instance [4]). Indeed, it is important for programming in the large, to have modular termination proofs, i.e. proofs that are capable of combining termination proofs of separate programs to obtain termination proofs of combined programs.

The rest of the paper is structured as follows: In the next section we present examples which motivate the need to freely mix tabled and Prolog execution. In Section 3, we introduce some necessary concepts and present a definition of SLG-resolution. Section 4 introduces a first notion of termination under tabled evaluation: *quasi-termination*. In addition, a characterisation for quasi-termination is given which generalises the characterisation given in [10] to the case where Prolog and tabled execution are intermixed. Then, a modular termination condition is given for the quasi-termination of the union $P \cup R$ of two programs P and R , where P extends R . In Section 5, the stronger notion of *LG-termination* is defined and characterised, and a method for obtaining modular proofs for LG-termination is presented. We conclude with discussing related and future work. In appendix A, a modular proof for quasi-termination of $P \cup R$ is given which explicitly constructs an appropriate level mapping for $P \cup R$ from level mappings for P and R .

$$\begin{array}{ll}
\text{expr}(Si, So) & \leftarrow \text{expr}(Si, S1), S1 = ['+'|S2], \text{term}(S2, So) \\
\text{expr}(Si, So) & \leftarrow \text{term}(Si, So) \\
\text{term}(Si, So) & \leftarrow \text{term}(Si, S1), S1 = ['*'|S2], \text{primary}(S2, So) \\
\text{term}(Si, So) & \leftarrow \text{primary}(Si, So) \\
\text{primary}(Si, So) & \leftarrow Si = ['(|S1], \text{expr}(S1, S2), S2 = [')|So] \\
\text{primary}(Si, So) & \leftarrow Si = [I|So], \text{integer}(I)
\end{array}$$

Figure 1: A tabled program recognizing simple arithmetic expressions.

2 Mixing Tabled and Prolog Execution: Motivating Examples

It has long been noted in the literature that tabled evaluation can be used for context-free grammar recognition and parsing: tabling eliminates redundancy and handles grammars that would otherwise infinitely loop under Prolog-style execution (e.g. left recursive ones). The program of Fig. 1 where all predicates are tabled, provides such an example. This grammar, recognizing arithmetic expressions containing additions and multiplications over the integers, is left recursive — left recursion is used to give the arithmetic operators their proper associativity — and would be non-terminating for Prolog-style execution. Under tabled execution, left recursion is handled correctly. In fact, one only needs to table predicates *expr/2* and *term/2* to get the desired termination behaviour; we can and will safely drop the tabling of *primary/2* in the sequel.

To see why a mix of tabled with Prolog execution is desirable in practice, suppose that we want to extend the above recognition grammar to handle exponentiation. The most natural way to do so is to introduce a new nonterminal, named *factor*, for handling exponentiation and make it right recursive, since the exponentiation operator is right associative. The resulting grammar is as below where only the predicates *expr/2* and *term/2* are tabled. Note that, at

$$\begin{array}{ll}
\text{expr}(Si, So) & \leftarrow \text{expr}(Si, S1), S1 = ['+'|S2], \text{term}(S2, So) \\
\text{expr}(Si, So) & \leftarrow \text{term}(Si, So) \\
\text{term}(Si, So) & \leftarrow \text{term}(Si, S1), S1 = ['*'|S2], \text{factor}(S2, So) \\
\text{term}(Si, So) & \leftarrow \text{factor}(Si, So) \\
\text{factor}(Si, So) & \leftarrow \text{primary}(Si, S1), S1 = ['^'|S2], \text{factor}(S2, So) \\
\text{factor}(Si, So) & \leftarrow \text{primary}(Si, So) \\
\text{primary}(Si, So) & \leftarrow Si = ['(|S1], \text{expr}(S1, S2), S2 = [')|So] \\
\text{primary}(Si, So) & \leftarrow Si = [I|So], \text{integer}(I)
\end{array}$$

least as far as termination is concerned, there is no need to table the new nonterminal. Indeed, Prolog's evaluation strategy handles right recursion in grammars finitely. In fact, Prolog-style evaluation of right recursion is more efficient than its tabled-based evaluation: Prolog has linear complexity for a simple right recursive grammar, but with tabling implemented as in XSB the evaluation could be quadratic as calls need to be recorded in the tables using explicit copying. Thus, it is important to allow tabled and non-tabled predicates to be freely intermixed, and be able to choose the strategy that is most efficient for the situation at hand.

By using tabling in context-free grammars, one gets a recognition algorithm that is a variant of Early's algorithm (also known as active chart recognition algorithm) whose complexity is polynomial in the size of the input expression/string [11]. However, often one wants to construct the parse tree(s) for a given input string. The usual approach is to introduce an extra argument

$$\begin{array}{l}
R : \left\{ \begin{array}{l}
s(Si, So) \leftarrow a(Si, S), S = [b|So] \\
a(Si, So) \leftarrow a(Si, S), a(S, So) \\
a(Si, So) \leftarrow Si = [a|So]
\end{array} \right. \\
\\
P : \left\{ \begin{array}{l}
s(Si, So, PT) \leftarrow a(Si, S), S = [b|So], PT = spt(Pa, b), \\
\qquad\qquad\qquad a(Si, S, Pa) \\
a(Si, So, PT) \leftarrow a(Si, S), a(S, So), PT = apt(P1, P2), \\
\qquad\qquad\qquad a(Si, S, P1), a(S, So, P2) \\
a(Si, So, PT) \leftarrow Si = [a|So], PT = a
\end{array} \right.
\end{array}$$

Figure 2: A tabled program recognizing and parsing the language $a^n b$.

to the nonterminals of the input grammar — representing the portion of the parse tree that each rule generates — and naturally to also add the necessary code that constructs the parse tree. This approach is straightforward, but as noticed in [21], using the same program for recognition as well as parsing may be extremely unsatisfactory from a complexity standpoint: in context-free grammars recognition is polynomial while parsing is exponential since there can be exponentially many parse trees for a given input string. The obvious solution is to use two interleaved versions of the grammar as in the example program of Fig. 2. Note that only $a/2$, i.e. the recursive predicate of the ‘recognition’ part, R , of the program (consisting of predicates $s/2$ and $a/2$), needs to be tabled. This action allows recognition to terminate and to have polynomial complexity. Furthermore, the recognizer can now be used as a filter for the parsing process in the following way: only after knowing that a particular part of the input belongs to the grammar and having computed the exact substring that each nonterminal spans, do we invoke the parsing routine on the nonterminal to construct its (possibly exponentially many) parse trees. Doing so, avoids e.g. cases where it may take exponential time to fail on an input string that does not belong in the given language: an example for the grammar under consideration is the input string a^n . On the other hand, tabling the ‘parsing’ part of the program (consisting of predicates $s/3$ and $a/3$) does not affect the efficiency of the process complexity-wise and incurs a small performance overhead due to the recording of calls and their results in the tables. Finally, note that the construction is modular in the sense that the ‘parsing’ part of the program, P , depends on the ‘recognition’ part, R , but not vice versa — we say that P *extends* R .

3 Preliminaries

We assume familiarity with the basic concepts of logic programming [16, 1]. Throughout the paper, P will denote a definite logic program and we restrict ourselves in this class. By $Pred_P$, we denote the set of predicates occurring in P , and by Def_P we denote the set of predicates defined in P (i.e. predicates occurring in the head of a clause of P). By Rec_P , resp. $NRec_P$, we denote the set of (directly or indirectly) recursive, resp. non-recursive, predicates of the program P (so $NRec_P = Pred_P \setminus Rec_P$). If $A = p(t_1, \dots, t_n)$, then we denote by $Rel(A)$ the predicate symbol p of A ; i.e. $Rel(A) = p$.

The *extended Herbrand Universe*, U_P^E , and the *extended Herbrand Base*, B_P^E , associated with a program 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 . 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. If $\Pi \subseteq Pred_P$, we denote with B_Π^E the subset of B_P^E consisting of (equivalence classes of) atoms based on the predicate symbols of Π . So B_P^E can be seen as an abbreviation of $B_{Pred_P}^E$.

Let P be a program and p, q 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. 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. Note that, by definition, each predicate depends on itself. We write $p \simeq q$ iff $p \sqsupseteq q$, $q \sqsupseteq p$ (p and q are mutually recursive or $p = q$). The *dependency graph* G of a program P is a graph where the nodes are labeled with the predicates of $Pred_P$. There is a *directed arc* from p to q in G iff p refers to q . Finally, we will say that a program P *extends* a program R iff no predicate defined in P occurs in R .

In analogy with [2], we will refer to SLD-derivations (see [16]) following the left-to-right selection rule as LD-derivations. Other concepts will adopt this naming accordingly.

Definition 3.1 (call set associated to S) *Let P be a program and $S \subseteq B_P^E$. By $Call(P, S)$ we denote the subset of B_P^E such that $B \in Call(P, S)$ whenever a representant of B is a selected atom in an LD-derivation for some $P \cup \{\leftarrow A\}$, with $\tilde{A} \in S$.*

In the sequel, with abuse of notation, we will write $Call(P, S)$ for a set $S \subseteq Atom_P$, when we mean the call set $Call(P, \tilde{S})$ associated to $\tilde{S} = \{\tilde{A} \mid A \in S\} \subseteq B_P^E$. Throughout the paper we assume that in any derivation of a query w.r.t. a program, representants of equivalence classes are systematically provided with fresh variables, to avoid the necessity of renaming apart. In the sequel, we abbreviate computed answer substitution with c.a.s.

The concepts defined in the following Definitions 3.2, 3.3 and 3.4, will be used in the proofs of theorems and propositions of this paper.

Definition 3.2 (direct descendant) *Let P be a program and $\tilde{A}, \tilde{B} \in B_P^E$. We call \tilde{B} a direct descendant of \tilde{A} iff*

- *there exists a clause $H \leftarrow B_1, \dots, B_n$ in P such that $mgu(A, H) = \theta$ exists,*
- *there is an $i \in [1, n]$ such that $\leftarrow (B_1, \dots, B_{i-1})\theta$ has an LD-refutation with c.a.s. θ_{i-1} and $B \approx B_i\theta\theta_{i-1}$.*

Definition 3.3 (directed subsequence of an LD-derivation) *Let P be a program and $\tilde{A} \in B_P^E$. Let $\leftarrow A = G_0, G_1, \dots$ be an LD-derivation of $\leftarrow A$ in P . A subsequence G_{i_0}, G_{i_1}, \dots , with $G_{i_j} = \leftarrow A_{i_j}, \mathbf{B}_{i_j}$, is called a directed subsequence iff for all $j \geq 0$, $\tilde{A}_{i_{j+1}}$ is a direct descendant of \tilde{A}_{i_j} in the LD-derivation.*

Definition 3.4 (call graph associated to S) *Let P be a program and $S \subseteq B_P^E$. The call graph $Call-Gr(P, S)$ associated to P and S is a graph such that:*

- *its set of nodes is $Call(P, S)$,*
- *there exists a directed arc from \tilde{A} to \tilde{B} iff \tilde{B} is a direct descendant of \tilde{A} .*

In the next definition we define the notion of level mapping, useful in the context of termination proofs (see for instance the survey [7]). We also define the concept of finitely partitioning level mapping, which was shown to be crucial in the context of a termination condition for tabled logic programs (see [10]).

Definition 3.5 (level mapping) Let P be a program and $L \subseteq B_P^E$.
A level mapping on L is a function $|\cdot| : L \rightarrow \mathbb{N}$.

With abuse of notation, we will write $|A|$, where A is a representant of $\tilde{A} \in B_P^E$, instead of $|\tilde{A}|$.

Definition 3.6 (finitely partitioning level mapping) Let P be a program and $C \subseteq L \subseteq B_P^E$.

A level mapping $|\cdot|$ on L is finitely partitioning on C iff for all $n \in \mathbb{N} : \sharp(|\cdot|^{-1}(n) \cap C) < \infty$, where \sharp is the cardinality function.

3.1 SLG-Resolution

We present a non-constructive definition of SLG-resolution that is sufficient for our purposes, and refer to [5, 6, 18, 20] for more constructive formulations of (variants) of tabled resolution. We refer to [10], where the same non-constructive definition of SLG-resolution is given in case all predicates occurring in a program are tabled.

By fixing a *tabling* for a program P , we mean choosing a set of predicates of P which are tabled. We denote this tabling as Tab_P . The complement of this set (i.e. the set of predicates which are not tabled w.r.t. that tabling) is denoted with $NTab_P = Pred_P \setminus Tab_P$.

Definition 3.7 (pseudo SLG-tree, pseudo LG-tree) Let P be a definite program, $Tab_P \subseteq Pred_P$, \mathcal{R} a selection rule and A an atom. A pseudo SLG-tree w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ under \mathcal{R} is a tree τ_A such that:

1. the nodes of τ_A are labeled with goals along with an indication of the selected atom according to \mathcal{R} ,
2. the arcs are labeled with substitutions,
3. the root of τ_A is $\leftarrow A$,
4. the children of the root $\leftarrow A$ are obtained by resolution against all matching program clauses in P , the arcs are labeled with the corresponding mgu used in the resolution step,
5. the children of a non-root node labeled with the goal \mathbf{Q} where $\mathcal{R}(\mathbf{Q}) = B$ are obtained as follows:
 - (a) if $Rel(B) \in Tab_P$, then the (possibly infinitely many) children of the node can only be obtained by resolving the selected atom B of the node with clauses of the form $B\theta \leftarrow$ (not necessarily in P), the arcs are labeled with the corresponding mgu used in the resolution step (i.e. θ),
 - (b) if $Rel(B) \in NTab_P$, then the children of the node are obtained by resolution of B against all matching program clauses in P , and the arcs are labeled with the corresponding mgu used in the resolution step.

If \mathcal{R} is the leftmost selection rule, τ_A is called a pseudo LG-tree w.r.t. Tab_P for $P \cup \{\leftarrow A\}$. We say that a pseudo SLG-tree τ_A w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ is smaller than another pseudo SLG-tree τ'_A w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ iff τ'_A can be obtained from τ_A by attaching new sub-branches to nodes in τ_A .

A (computed) answer clause of a pseudo SLG-tree τ_A w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ is a clause of the form $A\theta \leftarrow$ where θ is the composition of the substitutions found on a branch of τ_A whose leaf is labeled with the empty query.

Intuitively, a pseudo SLG-tree (in an SLG-forest, see Definition 3.8 below) represents the tabled computation (w.r.t. Tab_P) of all answers for a given subgoal labeling the root node of the tree. The trees in the above definition are called *pseudo* SLG-trees because there is no condition yet on which clauses $B\theta \leftarrow$ exactly are to be used for resolution in point 5a. These clauses represent the answers found (possibly in another tree of the forest) for the selected tabled atom. This interaction between the trees in an SLG-forest is captured in the following definition.

Definition 3.8 (SLG-forest, LG-forest) *Let P be a definite program, $Tab_P \subseteq Pred_P$, \mathcal{R} be a selection rule and T be a (possibly infinite) set of atoms such that no two different atoms in T are variants of each other. \mathcal{F} is an SLG-forest w.r.t. Tab_P for P and T under \mathcal{R} iff \mathcal{F} is a set of minimal pseudo SLG-trees $\{\tau_A \mid A \in T\}$ w.r.t. Tab_P where*

1. τ_A is a pseudo SLG-tree w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ under \mathcal{R} ,
2. every selected tabled atom B of each node in every $\tau_A \in \mathcal{F}$ is a variant of an element B' of T , such that every clause resolved with B is a variant of an answer clause of $\tau_{B'}$ and vice versa, for every answer clause of $\tau_{B'}$ there is a variant of this answer clause which is resolved with B .

Let S be a set of atoms. An SLG-forest for P and S w.r.t. Tab_P under \mathcal{R} is an SLG-forest w.r.t. Tab_P for a minimal set T with $S \subseteq T$. If $S = \{A\}$, then we also talk about the SLG-forest for $P \cup \{\leftarrow A\}$.

An LG-forest is an SLG-forest containing only pseudo LG-trees.

Point 2 of Definition 3.8, together with the imposed minimality of trees in a forest, now uniquely determines these trees. So we can drop the designation “pseudo” and refer to (S)LG-trees in an (S)LG-forest.

Note that, selected atoms which are not tabled (i.e. of which the predicate belongs to $NTab_P$) are resolved against program clauses, as in (S)LD-resolution. So, if $Tab_P = \emptyset$, the (S)LG-forest of $P \cup \{\leftarrow A\}$ consists of one tree: the (S)LD-tree of $P \cup \{\leftarrow A\}$. As illustrated in Section 2, it is important to allow tabled and non-tabled predicates to be freely intermixed, and thus to choose the strategy that is most efficient for the situation at hand. We use the following artificial tabled program to illustrate the concepts that we introduce.

Example 3.1 *Let P be the following program, with $Tab_P = \{member/2\}$.*

$$\begin{cases} intersection(Xs, Ys, Z) & \leftarrow member(Xs, Z), member(Ys, Z) \\ member([Z|Zs], Z) & \leftarrow \\ member([X|Zs], Z) & \leftarrow member(Zs, Z) \end{cases}$$

Let $S = \{intersection(Xs, Ys, a)\}$. Then, $Call(P, S) = S \cup \{member(Zs, a)\}$ and the LG-forest for P and S is shown in Fig. 3. Note that there is a finite number of LG-trees, all with finite branches, but the trees have infinitely branching nodes.

Note that we can use the notions of LD-derivation and LD-computation (as they appear in Definition 3.1 of the call set $Call(P, S)$) even in the context of SLG-resolution, as the set of call patterns and the set of computed answer substitutions are not influenced by tabling; see e.g. [14, Theorem 2.1].

In [10, Proposition A.1], it was noted that the notion of a call graph (Definition 3.4) has a particularly interesting property, which is useful in the study of termination under tabled execution. We recall this proposition together with its proof.

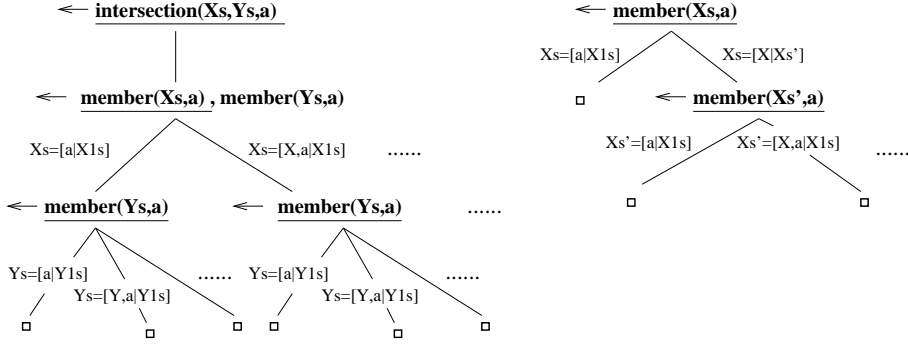


Figure 3: The LG-forest for $P \cup \{\leftarrow \text{intersection}(Xs, Ys, a)\}$.

Proposition 3.1 (call graph: paths and selected atoms) *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. Let p be any directed path in $Call-Gr(P, S)$. Then there exists an LG-derivation for some element of $Call(P, S)$, such that all the nodes in p occur as selected atoms in the derivation.*

Proof By definition of $Call-Gr(P, S)$, for every arc from \tilde{A} to \tilde{B} in $Call-Gr(P, S)$, there exists a sequence of consecutive LG-derivation steps, starting from $\leftarrow A$ and having a variant of B as its selected atom at the end. Because (a variant of) B is selected at the end-point, any two such derivation-step sequences, corresponding to consecutive arcs in $Call-Gr(P, S)$, can be composed to form a new sequence of LG-derivation steps. In this sequence, all 3 nodes of the consecutive arcs remain selected atoms in the new sequence of derivation steps. Transitively exploiting the above argument yields the result. \square

Note that by definition of $Call-Gr(P, S)$ and $Call(P, S)$, this also implies that there is a sequence of derivation steps starting from $P \cup \{\leftarrow A\}$, with $\tilde{A} \in S$, such that all nodes in the given path p are selected atoms in the derivation sequence. We will use Proposition 3.1 in the proofs of several termination conditions in this paper.

4 Quasi-Termination

A first basic notion of universal termination under tabled execution mechanism is quasi-termination. It is defined as follows (see [10, Definition 3.1] for the case $Tab_P = Pred_P$).

Definition 4.1 (quasi-termination) *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. P quasi-terminates w.r.t. Tab_P and S iff for all A such that $\tilde{A} \in S$, the LG-forest w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ consists of a finite number of LG-trees without infinite branches. Also, P quasi-terminates w.r.t. S iff P quasi-terminates w.r.t. $Pred_P$ and S .*

Note that it is not required that the LG-trees are finitely branching in their nodes. In the next section, we introduce and provide conditions for the stronger notion of LG-termination, which requires that the LG-forest consists of a finite number of finite trees (i.e. trees with finite branches and which are finitely branching).

Example 4.1 *Recall P and S of Example 3.1. P quasi-terminates w.r.t. $\{\text{member}/2\}$ and S . Note that P does not LG-terminate w.r.t. $\{\text{member}/2\}$ and S .*

Many works (see [7] for a survey) address the problem of proving LD-termination: A program P is said to be *LD-terminating* w.r.t. a set $S \subseteq B_P^E$ iff for all A such that $\tilde{A} \in S$, the LD-tree of $P \cup \{\leftarrow A\}$ is finite (see for instance [8]). In the next Lemma 4.1, we show that the notion of LD-termination is stronger than the notion of quasi-termination. As shown in Example 3.1, the notion of LD-termination is “strictly” stronger than the notion of quasi-termination.

Lemma 4.1 *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. If P LD-terminates w.r.t. S , then P quasi-terminates w.r.t. Tab_P and S .*

Proof Let A be an atom such that $\tilde{A} \in S$. Let \mathcal{F} be the LG-forest w.r.t. Tab_P for $P \cup \{\leftarrow A\}$. If P LD-terminates w.r.t. S , it is easy to see that $Call(P, \{A\})$ is finite. Hence, $Call(P, \{A\}) \cap B_{Tab_P}^E$ is finite, and \mathcal{F} consists of a finite number of LG-trees. Now we prove that no tree in \mathcal{F} has an infinite branch. Suppose this is not the case and there is a tree in \mathcal{F} with an infinite branch. Let H be the leftmost atom of a query labeling a node in this infinite branch. Then, H has an infinite LD-derivation (just plug in, for each tabled atom G in the infinite branch which is resolved with an answer, the branch of the tree with root G which leads to this answer). This gives a contradiction. \square

Note that by definition, P quasi-terminates w.r.t. $Tab_P = \emptyset$ and S iff P LD-terminates w.r.t. S .

In [10] the special case where $Tab_P = Pred_P$, i.e. all predicates occurring in P are tabled, is considered. If $Tab_P = Pred_P$, an LG-tree cannot have infinite branches. So, P quasi-terminates w.r.t. a set S iff for all A such that $\tilde{A} \in S$, the LG-forest for $P \cup \{\leftarrow A\}$ consists of a finite number of LG-trees. The following equivalence was proven in [10].

Lemma 4.2 [10, Lemma 3.1] *Let P be a program and $S \subseteq B_P^E$. P quasi-terminates w.r.t. S iff for every $A \in S$, $Call(P, \{A\})$ is finite.*

Note that, in case the Herbrand Universe U_P^E associated to a program P is finite, P quasi-terminates w.r.t. any set of queries S .

Lemma 4.2 is not true in case that the tabled predicates of a program are a strict subset of the set of predicates occurring in the program. A counterexample for the if-direction is given by the program $P = \{p \leftarrow q, q \leftarrow p\}$, the set $S = \{p\}$ and the empty set of tabled predicates, $Tab_P = \emptyset$. The LG-forest consists of one tree, namely the LD-tree of $P \cup \{\leftarrow p\}$, and thus quasi-termination is the same as LD-termination. P doesn't quasi-terminate w.r.t. Tab_P and S , whereas $Call(P, \{p\}) = \{p, q\}$ is a finite set. Also the only if-direction of Lemma 4.2 does not hold in case $Tab_P \subset Pred_P$. A counterexample is given in the following example.

Example 4.2 *Consider the following program P :*

$$\left\{ \begin{array}{l} p(a) \\ p(f(X)) \leftarrow p(X), q(X) \\ q(X) \end{array} \right.$$

with set of tabled predicates $Tab_P = \{p/1\}$ and the set $S = \{p(X)\}$. The LG-forest is shown in Figure 4. P quasi-terminates w.r.t. Tab_P and S . There is only one LG-tree in the LG-forest for $P \cup \{\leftarrow p(X)\}$ with no infinite branches (note that the LG-tree has an infinitely branching node). But the set $Call(P, \{p(X)\}) = \{p(X), q(a), \dots, q(f^n(a)), \dots\}$ is infinite.

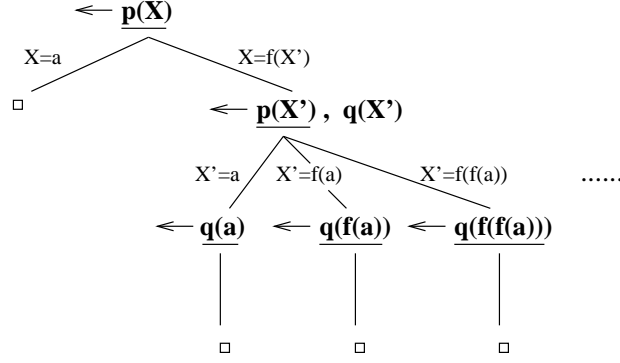


Figure 4: The LG-forest for $P \cup \{\leftarrow p(X)\}$.

Note however that, since quasi-termination requires that there are only finitely many LG-trees in the LG-forest of a query, there can only be a finite number of tabled atoms in the call set of that query. Hence, in case $Tab_P \subseteq Pred_P$, we have the following result.

Lemma 4.3 *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. If P quasi-terminates w.r.t. Tab_P and S , then, for every $A \in S$, $Call(P, \{A\}) \cap B_{Tab_P}^E$ is finite.*

Proof Let A be an atom such that $\tilde{A} \in S$. If P quasi-terminates w.r.t. Tab_P and S , we know that the LG-forest \mathcal{F} w.r.t. Tab_P of $P \cup \{\leftarrow A\}$ consists of a finite number of LG-trees without infinite branches. Consider the set $Call(P, \{A\}) \cap B_{Tab_P}^E$. Because \mathcal{F} consists of a finite number of trees and the root of the trees are exactly of the form $\leftarrow B$ where $\tilde{B} \in Call(P, \{A\}) \cap B_{Tab_P}^E$, this set is finite. \square

Example 4.3 *Recall the program P and the set S of Example 4.2. We already know that P quasi-terminates w.r.t. Tab_P and S . And indeed, $Call(P, \{p(X)\}) \cap B_{Tab_P}^E = \{p(X)\}$ is finite.*

4.1 Characterisation of Quasi-Termination

In this subsection, we will give a condition for quasi-termination, called quasi-acceptability. In case the tabling satisfies some property, quasi-acceptability is a necessary and sufficient condition for quasi-termination; otherwise, it is a sufficient condition. We want to note that the condition is adapted from the acceptability notion for LD-termination defined in [8], and not from the more “standard” definition of acceptability by Apt and Pedreschi in [3]. The reason for this choice is that the quasi-termination property of a tabled program and (set of) goal(s) is *not* invariant under substitution. Consider the following example from [15].

Example 4.4 *Let $p/2$ be a tabled predicate defined by the following clause.*

$$p(f(X), Y) \leftarrow p(X, Y)$$

Then, the query $\leftarrow p(X, Y)$ terminates while $\leftarrow p(X, X)$ does not.

The acceptability notion in [3] is expressed in terms of ground instances of clauses and its associated notion of left-termination is expressed in terms of the set of all goals that are bounded under the given level mapping. Such sets are closed under substitution. Because quasi-termination lacks invariance under substitution, we need a stronger notion of acceptability, capable of treating *any* set of interest S .

In order to state a necessary and sufficient condition for quasi-termination, we need to make an assumption on the set Tab_P of tabled predicates of the program P . If the assumption is satisfied, we are able to give a necessary and sufficient condition for quasi-termination. If the assumption is not satisfied, the condition is still sufficient.

We first need the following notations. Let P be a program and let G_P be the dependency graph of the predicates of P . For a tabling Tab_P for P and predicates $p, q \in NTab_P$ with $p \simeq q$, let $C_1(p, q)$, $C_2(p, q)$ and $C_3(p, q)$ denote the following cases:

$C_1(p, q)$: no cycle of directed arcs in G_P containing p and q contains a predicate from Tab_P .

$C_2(p, q)$: all cycles of directed arcs in G_P containing p and q contain at least one predicate from Tab_P .

$C_3(p, q)$: there is a cycle of directed arcs in G_P containing p and q which contains no predicate from Tab_P and there is a cycle of directed arcs in G_P containing p and q which contains a predicate from Tab_P .

Note that $C_1(p, q)$, $C_2(p, q)$ and $C_3(p, q)$ depend on the program P (more precisely on the dependency graph G_P of P) and on the tabling Tab_P for P . When referring to one of these three cases, it will always be clear from the context which program and tabling are under consideration. Note that, given a program P and tabling Tab_P , for all predicates $p, q \in NTab_P$ with $p \simeq q$, exactly one of the cases $C_1(p, q)$, $C_2(p, q)$ or $C_3(p, q)$ hold.

We next define the notion of well-chosen tabling w.r.t. a program P . This is a tabling for P such that the third case C_3 does never occur. When the tabling is well-chosen, we are able to give a necessary and sufficient condition for quasi-termination.

Definition 4.2 (well-chosen tabling (w.r.t. a program)) *Let P be a program. The tabling Tab_P is called well-chosen w.r.t. the program P iff for every $p, q \in NTab_P$ such that $p \simeq q$, either $C_1(p, q)$ or $C_2(p, q)$ holds.*

Note that, in case Tab_P is well-chosen w.r.t. P , we have that, if $p, q, r \in NTab_P$ and $p \simeq q \simeq r$ and $C_1(p, q)$ (resp. $C_2(p, q)$) holds, then $C_1(q, r)$ (resp. $C_2(q, r)$) holds.

In particular, if $NTab_P \subseteq \{p \in Pred_P \mid p \text{ is a non-recursive or only directly recursive predicate}\}$ or if $NTab_P = \emptyset$ (i.e. $Tab_P = Pred_P$), then the tabling Tab_P is well-chosen w.r.t. P .

We next give a simple example of a program and tabling, which is not well-chosen w.r.t. the program.

Example 4.5 *Let P be the following propositional program:*

$$\left\{ \begin{array}{l} p \leftarrow q \\ q \leftarrow r \\ r \leftarrow p \\ r \leftarrow q \end{array} \right.$$

with $Tab_P = \{p\}$. This tabling is not well-chosen w.r.t. P .

We next introduce the notion of quasi-acceptability, which is a necessary and sufficient condition for quasi-termination in case the tabling is well-chosen and which is a sufficient condition for quasi-termination in general.

Definition 4.3 (quasi-acceptability) Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. P is quasi-acceptable w.r.t. Tab_P and S iff there is a level mapping $|\cdot|$ on B_P^E such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $Call(P, \{A\}) \cap B_{Tab_P}^E$ and such that

- for every atom A such that $\tilde{A} \in Call(P, S)$,
- for every clause $H \leftarrow B_1, \dots, B_n$ in P , such that $mgu(A, H) = \theta$ exists,
- for every $1 \leq i \leq n$ and for every LD-c.a.s. θ_{i-1} for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$\begin{aligned} |A| &\geq |B_i\theta\theta_{i-1}| \\ \text{and} \\ |A| &> |B_i\theta\theta_{i-1}| \quad \text{if } Rel(A) \simeq Rel(B_i) \in NTab_P \text{ and} \\ &\quad C_2(Rel(A), Rel(B_i)) \text{ does not hold.} \end{aligned}$$

Theorem 4.1 (characterisation of quasi-termination) Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$.

If P is quasi-acceptable w.r.t. Tab_P and S , then P quasi-terminates w.r.t. Tab_P and S .

If the tabling Tab_P is well-chosen w.r.t. P , then also the converse holds, i.e. P is quasi-acceptable w.r.t. Tab_P and S iff P quasi-terminates w.r.t. Tab_P and S .

Proof \Rightarrow : Suppose that P is quasi-acceptable w.r.t. Tab_P and S . We prove that P quasi-terminates w.r.t. Tab_P and S . Let A be an atom such that $\tilde{A} \in S$, let \mathcal{F} be the LG-forest of $P \cup \{\leftarrow A\}$.

- \mathcal{F} consists of a finite number of LG-trees, i.e. $\sharp(Call(P, \{A\}) \cap B_{Tab_P}^E) < \infty$.
Due to the quasi-acceptability condition, any call in $Call(P, \{A\})$ directly descending from A , say B , is such that $|A| \geq |B|$. The same holds recursively for the atoms descending from B . Thus, the level mapping of any call, recursively descending from A , is smaller than or equal to $|A| \in \mathbb{N}$. Since $|\cdot|$ is finitely partitioning on $Call(P, \{A\}) \cap B_{Tab_P}^E$, we have that: $\sharp(\bigcup_{n \leq |A|} l^{-1}(n) \cap Call(P, \{A\}) \cap B_{Tab_P}^E) < \infty$. Hence, $\sharp(Call(P, \{A\}) \cap B_{Tab_P}^E) < \infty$, i.e. \mathcal{F} consists of a finite number of trees.
- The LG-trees in \mathcal{F} have finite branches.

Suppose there is a tree in \mathcal{F} with an infinite branch. This infinite branch contains an infinite directed subsequence G_0, G_1, \dots . It is easy to see that the leftmost atoms in the nodes of this infinite directed subsequence all are $NTab_P$ -atoms (because Tab_P -atoms are resolved using answers). There is a $n \in \mathbb{N}$, such that each $G_i, i \geq n$, has as leftmost atom A_i and for all $i, j \geq n$, $Rel(A_i) \simeq Rel(A_j)$ and $C_2(Rel(A_i), Rel(A_j))$ doesn't hold. Because of the quasi-acceptability condition, $|A_i| > |A_{i+1}|$, for all $i \geq n$. This gives a contradiction.

\Leftarrow : Suppose that the tabling Tab_P is well-chosen w.r.t. P and suppose that P quasi-terminates w.r.t. S . We have to construct a level mapping $|\cdot|$ such that P is quasi-acceptable w.r.t. Tab_P, S and this level mapping $|\cdot|$. We will only define $|\cdot|$ on elements of $Call(P, S)$. On elements of the complement of $Call(P, S)$ in B_P^E , $|\cdot|$ can be assigned any value, as these elements don't turn up in the quasi-acceptability condition.

In order to define $|\cdot|$ on $Call(P, S)$, consider the $Call-Gr(P, S)$ -graph (Definition 3.4).

Consider a strongly connected component C in $Call-Gr(P, S)$.

Then, there is at least one Tab_P -atom in C . To see this, suppose this is not the case. Consider a cyclic path p in C . This consists only of $NTab_P$ -atoms. But then, because of Proposition 3.1, there is an infinite branch in a tree of the LG-forest of an element of S . This gives a contradiction.

Also, there is only a finite number of Tab_P -atoms in C . To see this, suppose this is not the case. Then there is an infinitely long path p through infinitely many Tab_P -atoms of C . Because of Proposition 3.1, there is an infinite number of Tab_P -atoms selected in a derivation of an element of S , i.e. there are infinitely many trees in the LG-forest of that element of S . This gives a contradiction.

Note also that, for every two atoms in C with predicates $p, q \in NTab_P$, $p \simeq q$ and the condition $C_1(p, q)$ does not hold (since there is at least one Tab_P -atom in C). Or, since the tabling is well-chosen, the condition $C_2(p, q)$ holds.

Define \overline{CG} as the graph obtained from $Call-Gr(P, S)$ by replacing any strongly connected component by a single contracting node and replacing any arc from $Call-Gr(P, S)$ pointing to (resp. from) any node in that strongly connected component by an arc to (resp. from) that contracting node. \overline{CG} does not have any (non-trivial) strongly connected components. Moreover, any strongly connected component from $Call-Gr(P, S)$ that was collapsed into a contracting node of \overline{CG} necessarily contains at least one and at most a finite number of Tab_P -atoms.

Note now that each path in \overline{CG} which is not cyclic (not that there are only trivial cycles in \overline{CG}) is finite. This also follows directly from Proposition 3.1.

Note that it is possible that \overline{CG} has an infinitely branching (possibly contracting) node. Let A be an atom in that infinitely branching node. It follows from Lemma 4.3 that, because P quasi-terminates w.r.t. S , $\sharp(\{B \mid B \text{ is a descendant of } A \text{ in } \overline{CG}\} \cap B_{Tab_P}^E) < \infty$.

We now construct $\overline{\overline{CG}}$ from \overline{CG} starting from the top nodes N_1 downwards as follows:

- replace all direct descendants of N_1 in \overline{CG} , by a single contracting node N_2 ,
- replace any arc from \overline{CG} pointing to (resp. from) any node in that (possibly infinite) set of direct descendants by an arc to (resp. from) that contracting node N_2 ,
- repeat this for the nodes N_2 .

This process stops because, as we already noted, each path in \overline{CG} which is not cyclic is finite. It is easy to see that $\overline{\overline{CG}}$ is a graph in which each node has at most one direct descendant. Also, each node in $\overline{\overline{CG}}$ consists of a (possibly infinite) set of nodes of $Call-Graph(P, S)$ which contains only finitely many Tab_P -atoms.

We define the level mapping $|\cdot|$ as follows. Consider the layers of $\overline{\overline{CG}}$ (there are only a finite number of layers). Let layer-0 be the set of leaves in $\overline{\overline{CG}}$. We assign to these nodes a number in \mathbb{N} , such that all nodes get a different number. Then, we move up to the next layer in $\overline{\overline{CG}}$. This layer, layer-1, consists of all nodes N such that the path starting from N has length 1. We assign to each such node N a natural number, such that the number assigned to N is strictly larger than the number assigned to its descendant (in the previous step). We continue this process layer by layer. The value of the level mapping $|\cdot|$ on elements of $Call(P, S)$ is defined as follows: all calls contained in the node N receive the number assigned to the node N .

We prove that, P is quasi-acceptable w.r.t. Tab_P, S and this level mapping $|\cdot|$.

- for every $A \in S$, $|\cdot|$ is finitely partitioning on $B_{Tab_P}^E \cap Call(P, \{A\})$.
Note that $|\cdot|$ is even finitely partitioning on $B_{Tab_P}^E \cap Call(P, S)$. This is because each (contracting) node of \overline{CG} contains only a finite number of Tab_P -atoms and because of the construction of $|\cdot|$.
- Let A be an atom such that $\tilde{A} \in Call(P, S)$, let $H \leftarrow B_1, \dots, B_n$ be a clause in P , such that $mgu(A, H) = \theta$ exists, let θ_{i-1} be a LD-c.a.s. for $\leftarrow (B_1, \dots, B_{i-1})\theta$:
 - then $|A| \geq |B_i\theta\theta_{i-1}|$.
This is because there is a directed arc from A to $B_i\theta\theta_{i-1}$ in $Call-Graph(P, S)$ and because of the construction of $|\cdot|$.
 - then $|A| > |B_i\theta\theta_{i-1}|$ if $Rel(A) \simeq Rel(B_i) \in NTab_P$ and $C_2(Rel(A), Rel(B_i))$ does not hold (i.e. $C_1(Rel(A), Rel(B_i))$ holds).
There is a directed arc in $Call-Graph(P, S)$ from A to $B_i\theta\theta_{i-1}$. Note that A and $B_i\theta\theta_{i-1}$ don't belong to the same strongly connected component of $Call-Graph(P, S)$. This is because $C_1(Rel(A), Rel(B_i))$ holds. So, A and $B_i\theta\theta_{i-1}$ belong to a different layer and $B_i\theta\theta_{i-1}$ is a direct descendant of A . Hence, because of the construction of $|\cdot|$, $|A| > |B_i\theta\theta_{i-1}|$.

□

We illustrate the intuition behind Theorem 4.1 in the following example.

Example 4.6 Consider the following two propositional programs P and P' :

$$P : \begin{cases} p \leftarrow q \\ q \leftarrow r \\ r \leftarrow q \end{cases} \quad P' : \begin{cases} p \leftarrow q \\ q \leftarrow r \\ r \leftarrow p \end{cases}$$

with $Tab_P = Tab_{P'} = \{p\}$ and $S = \{p\}$. The LG-forests for $P \cup \{\leftarrow p\}$ and $P' \cup \{\leftarrow p\}$ are shown in Figure 5. P does not quasi-terminate w.r.t. $\{p\}$, whereas P' does quasi-terminate

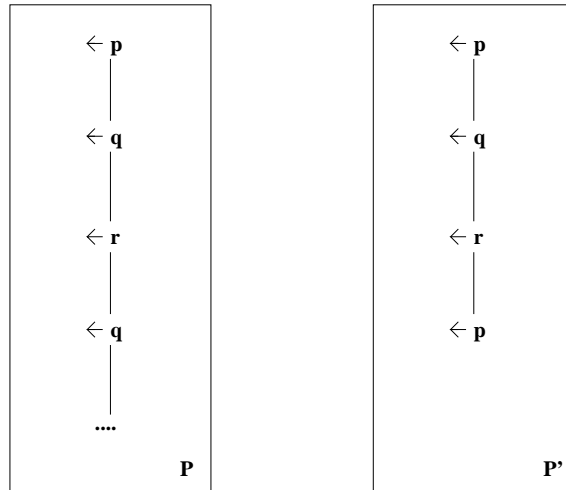


Figure 5: The LG-forests for $P \cup \{\leftarrow p\}$ and for $P' \cup \{\leftarrow p\}$

w.r.t. $\{p\}$.

This can be proven by Theorem 4.1. Note that for both programs, the tablings are well-chosen. Also note that, because the programs are propositional, every level mapping is finitely partitioning on the whole Herbrand base (assuming that there are only finitely many proposition symbols).

Let's first consider program P . Notice that for this program and tabling $\{p\}$ the condition $C_1(q, r)$ holds. Also note that there is no level mapping $|\cdot|$ such that $|q| > |r|$ and $|r| > |q|$ holds. Hence, the condition in Theorem 4.1 can not be satisfied and P does not quasi-terminate w.r.t. $\{p\}$.

Consider next program P' . Notice that for this program and tabling $\{p\}$ the condition $C_2(q, r)$ holds. Let $|\cdot|$ be the following level mapping $|p| = |q| = |r| = 0$. With this level mapping, P' satisfies the condition of Theorem 4.1 and hence, P' quasi-terminates w.r.t. $\{p\}$.

4.2 Modular Proofs for Quasi-Termination

In this subsection, we present modular proofs for quasi-termination of the union $P \cup R$ of two programs P and R , where P extends R (recall that a program P extends a program R iff no predicate defined in P occurs in R). In order to fix a notation, for $Pred_{P \cup R} = Tab_{P \cup R} \sqcup NTab_{P \cup R}$, let

$$\begin{aligned} Tab_P &= Tab_{P \cup R} \cap Pred_P & , & & NTab_P &= NTab_{P \cup R} \cap Pred_P, \\ Tab_R &= Tab_{P \cup R} \cap Pred_R & , & & NTab_R &= NTab_{P \cup R} \cap Pred_R. \end{aligned}$$

So the tabling of the union $P \cup R$ determines the tabling of the components P and R . Note that Tab_P also contains predicates which are tabled in $P \cup R$ but defined in R .

The first Proposition 4.1 proves the quasi-termination of the program $P \cup R$, where P extends R , by imposing conditions on the two parts P and R . The following two propositions consider special cases of this first proposition. In Proposition 4.2 we consider the case in which all defined predicates in P are tabled and in Proposition 4.3 we consider the case in which both programs extend each other. That is, no predicate defined in one program depends on a predicate defined in the other program. The case where no defined predicate symbol in P is tabled, does not lead to a simpler modular termination condition, than the condition of Proposition 4.1.

Proposition 4.1 *Suppose P and R are two programs, such that P extends R . Let $S \subseteq B_{P \cup R}^E$. If*

- R quasi-terminates w.r.t. Tab_R and $Call(P \cup R, S)$,
- there is a level mapping $|\cdot|$ on B_P^E such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $Call(P \cup R, \{A\}) \cap B_{Tab_P}^E$, and such that
 - for every atom A such that $\tilde{A} \in Call(P \cup R, S)$,
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P such that $mgu(A, H) = \theta$ exists,
 - for every $1 \leq i \leq n$ and for every LD-c.a.s. θ_{i-1} in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$\begin{aligned} |A| &\geq |B_i \theta \theta_{i-1}| \\ \text{and} \\ |A| &> |B_i \theta \theta_{i-1}| \quad \text{if } Rel(A) \simeq Rel(B_i) \in NTab_P \text{ and} \\ &\quad C_2(Rel(A), Rel(B_i)) \text{ does not hold} \end{aligned}$$

then, $P \cup R$ quasi-terminates w.r.t. $Tab_{P \cup R}$ and S .

Proof Let A be an atom such that $\tilde{A} \in S$. Let \mathcal{F} be the LG-forest w.r.t. $Tab_{P \cup R}$ of $P \cup R \cup \{\leftarrow A\}$. We prove that \mathcal{F} consists of a finite number of LG-trees without infinite branches. If A is defined in R , this follows directly from the fact that P extends R and that R quasi-terminates w.r.t. Tab_R and $Call(P \cup R, S)$.

So, suppose A is defined in P . Because of the second condition in the proposition statement, every call directly descending from A , say B , is such that $|B| \leq |A|$. This holds recursively for atoms descending from A using clauses of P . Because $|\cdot|$ is finitely partitioning on $B_{Tab_P}^E \cap Call(P \cup R, \{A\})$, the set of tabled atoms, descending from A , using clauses of P is finite. For atoms C , defined in R and descending from A using clauses of P , we know that $\sharp(Call(P \cup R, \{C\}) \cap B_{Tab_R}^E) < \infty$. So, $\sharp(Call(P \cup R, \{A\}) \cap B_{Tab_{P \cup R}}^E) < \infty$.

We now prove that there is no tree in \mathcal{F} with an infinite branch. Suppose this is not the case, and there is a tree in \mathcal{F} with an infinite branch. Because, R quasi-terminates w.r.t. Tab_R and $Call(P \cup R, S)$, and because P extends R , this infinite branch contains an infinite directed subsequence G_0, G_1, \dots , with leftmost atoms A_0, A_1, \dots , belonging to $B_{NTab_P \cap Def_P}^E$. This infinite directed subsequence has a tail, such that for all i such that G_i belongs to this tail, $Rel(A_i) \simeq Rel(A_{i+1})$ and $C_2(Rel(A_i), Rel(A_{i+1}))$ does not hold. But because of the condition in the proposition statement, $|A_i| > |A_{i+1}|$ and this gives a contradiction. \square

Example 4.7 Recall the program of Example 3.1. We write this program as the union of two programs $P \cup R$, where P extends R :

$$P : \begin{cases} intersection(Xs, Ys, Z) \leftarrow member(Xs, Z), member(Ys, Z) \\ member([Z|Zs], Z) \leftarrow \\ member([X|Zs], Z) \leftarrow member(Zs, Z) \end{cases}$$

with $Tab_{P \cup R} = \{member/2\}$ and $S = \{intersection(Xs, Ys, a)\}$. Then, $Call(P \cup R, S) = S \cup \{member(Xs, a)\}$. As we already noted in Example 4.1, $P \cup R$ quasi-terminates w.r.t. $Tab_{P \cup R}$ and S (the LG-forest for $P \cup R$ and S is shown in Figure 3). We can prove this in a modular way, by using Proposition 4.1:

- R quasi-terminates w.r.t. $\{member/2\}$ and $Call(P \cup R, S)$.

To prove that R quasi-terminates w.r.t. $\{member/2\}$ and $\{member(Xs, a)\}$, we can apply Theorem 4.1. It can be easily verified that R is quasi-acceptable w.r.t. $\{member/2\}$, $\{member(Xs, a)\}$ and the trivial level mapping (mapping everything to 0). (Note that all level mappings are finitely partitioning on the set $Call(R, \{member(Xs, a)\}) \cap B_{Tab_R}^E = \{member(Xs, a)\}$, as this is a finite set.)

- Let $|\cdot|$ be the trivial level mapping (mapping everything to 0). As we already noted above, $|\cdot|$ is finitely partitioning on $Call(P \cup R, S) \cap B_{member/2}^E = \{member(Xs, a)\}$. We consider the clause of P . It is easy to see that $|intersection(Xs, Ys, a)| = 0 \geq 0 = |member(Xs, a)|$, and $|intersection(Xs, Ys, a)| = 0 \geq 0 = |member(Ys, a)|$ (note that the computed answer substitutions of the first body atom do not instantiate the second body atom). So, the second condition of Proposition 4.1 is also satisfied.

Hence we can conclude that $P \cup R$ quasi-terminates w.r.t. S .

The next proposition considers the special case of two programs P and R , such that P extends R and such that all the defined predicates in P are tabled.

Proposition 4.2 *Suppose P and R are two programs, such that P extends R , and such that $Def_P \subseteq Tab_P$. Let $S \subseteq B_{P \cup R}^E$. If*

- R quasi-terminates w.r.t. Tab_R and $Call(P \cup R, S)$,
- there is a level mapping $|\cdot|$ on B_P^E such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $Call(P \cup R, \{A\}) \cap B_{Tab_P}^E$, and such that
 - for every atom A such that $\tilde{A} \in Call(P \cup R, S)$
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P such that $mgu(A, H) = \theta$ exists,
 - for every $1 \leq i \leq n$ and for every LD-c.a.s. θ_{i-1} in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$|A| \geq |B\theta\theta_{i-1}|$$

then, $P \cup R$ quasi-terminates w.r.t. $Tab_{P \cup R}$ and S .

Proof This is a direct corollary of Proposition 4.1 (every recursive predicate in P is defined in P and hence tabled). \square

Finally, we consider the case of two programs P_1 and P_2 extending each other.

Proposition 4.3 *Let P_1, P_2 be two programs such that P_1 extends P_2 and P_2 extends P_1 . Let $S \subseteq B_{P_1 \cup P_2}^E$. Suppose that*

- P_1 quasi-terminates w.r.t. Tab_{P_1} and $S \cap B_{P_1}^E$,
- P_2 quasi-terminates w.r.t. Tab_{P_2} and $S \cap B_{P_2}^E$,

then $P_1 \cup P_2$ quasi-terminates w.r.t. $Tab_{P_1 \cup P_2}$ and S .

Proof Because P_1 extends P_2 and P_2 extends P_1 , $Call(P_1 \cup P_2, S) \cap B_{P_i}^E = Call(P_i, S \cap B_{P_i}^E)$ for $i = 1, 2$. The proposition follows by definition of quasi-termination. \square

Note that all the above modular termination conditions prove the quasi-termination of $P \cup R$ without constructing a level mapping $|\cdot|$ such that $P \cup R$ is quasi-acceptable w.r.t. this level mapping $|\cdot|$. We refer to the Appendix A where modular termination conditions for quasi-termination are given which construct (from simpler level mappings) a level mapping such that $P \cup R$ is quasi-acceptable w.r.t. this level mapping.

5 LG-Termination

The notion of quasi-termination only partially corresponds to our intuitive notion of a terminating execution of a query against a tabled program. This notion only requires that the LG-forest consists of only a finite number of LG-trees, without infinite branches, yet these trees can have infinitely branching nodes. In order to capture this source of non-termination for a tabled computation, the following stronger notion is introduced (see [10, Definition 4.1] for the special case where $Tab_P = Pred_P$).

Definition 5.1 (LG-termination) Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. P LG-terminates w.r.t. Tab_P and S iff for every atom A such that $\tilde{A} \in S$, the LG-forest w.r.t. Tab_P for $P \cup \{\leftarrow A\}$ consists of a finite number of finite LG-trees. Also, P LG-terminates w.r.t. S iff P LG-terminates w.r.t. $Pred_P$ and S .

Example 5.1 As we already noted before, the program P of Example 3.1 doesn't LG-terminate w.r.t. $\{member/2\}$ and $\{intersection(Xs, Ys, a)\}$.

Obviously, the notion of LG-termination is stronger than the notion of quasi-termination. Also, LD-termination implies LG-termination.

Lemma 5.1 Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. If P LG-terminates w.r.t. Tab_P and S , then P quasi-terminates w.r.t. Tab_P and S . If P LD-terminates w.r.t. S , then P LG-terminates w.r.t. Tab_P and S .

Proof The first statement is trivial by definition. For the second statement, this is a corollary of the following Proposition 5.1 with $Tab_1 = \emptyset$ and $Tab_2 = Tab_P$. \square

Consider two tablings for a program P ; one with set of tabled predicates equal to $Tab_1 \subseteq Pred_P$, the other with set of tabled predicates equal to $Tab_2 \subseteq Pred_P$. Suppose $Tab_1 \subseteq Tab_2$ (hence $NTab_1 \supseteq NTab_2$). The next proposition studies the relationship between the LG-termination of P w.r.t. these two tablings.

Proposition 5.1 Let P be a program. Let $Pred_P = Tab_1 \sqcup NTab_1$ and $Pred_P = Tab_2 \sqcup NTab_2$. Suppose $Tab_1 \subseteq Tab_2$. Let $S \subseteq B_P^E$. If P LG-terminates w.r.t. Tab_1 and S , then P LG-terminates w.r.t. Tab_2 and S .

Proof Let A be an atom such that $\tilde{A} \in S$. Let \mathcal{F}_1 be the LG-forest w.r.t. Tab_1 of $P \cup \{\leftarrow A\}$ and let \mathcal{F}_2 be the LG-forest w.r.t. Tab_2 of $P \cup \{\leftarrow A\}$. We know that \mathcal{F}_1 consists of a finite number of finite LG-trees. So, $\sharp Call(P, \{A\}) < \infty$, hence, $\sharp(Call(P, \{A\}) \cap B_{Tab_2}^E) < \infty$ and \mathcal{F}_2 consists of a finite number of LG-trees. We prove that the LG-trees of \mathcal{F}_2 are finite. Since each LG-tree in \mathcal{F}_2 can be extended to obtain an LG-tree in \mathcal{F}_1 , this follows from the finiteness of the LG-trees in \mathcal{F}_1 . \square

Note that this proposition does not hold for quasi-termination as is shown in the following example.

Example 5.2 Recall from Example 4.2 the following program P :

$$\left\{ \begin{array}{l} p(a) \\ p(f(X)) \leftarrow p(X), q(X) \\ q(X) \end{array} \right.$$

with $S = \{p(X)\}$. Let $Tab_1 = \{p/1\}$ (as in Example 4.2) and $Tab_2 = \{p/1, q/1\}$. Then, P quasi-terminates w.r.t. Tab_1 and S (the LG-forest in this case was shown in Figure 4). But, as is shown in Figure 6, P doesn't quasi-terminate w.r.t. Tab_2 and S .

The following lemma relates the notions of quasi-termination and LG-termination in a more detailed way. By definition, quasi-termination only corresponds to part of the LG-termination notion; it fails to capture non-termination caused by an infinitely branching node in an LG-tree. Note that if an LG-forest contains a tree with an infinitely branching node, then there is

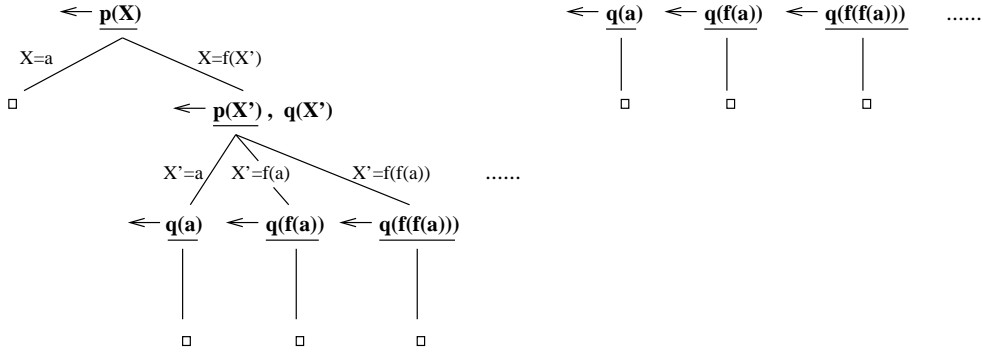


Figure 6: The LG-forest for $P \cup \{\leftarrow p(X)\}$ w.r.t. Tab_2

an LG-tree in the forest which is infinitely branching in a node which contains a goal with a recursive, tabled atom at the leftmost position. This observation leads to the following lemma. Denote the set of tabled, recursive predicates in a program P with TR_P :

$$TR_P = Tab_P \cap Rec_P.$$

Lemma 5.2 *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. P LG-terminates w.r.t. Tab_P and S iff P quasi-terminates w.r.t. Tab_P and S and for all $A \in S$ the set of (LD-)computed answers for atoms in $Call(P, \{A\}) \cap B_{TR_P}^E$ is finite.*

Proof \Rightarrow : Suppose P LG-terminates w.r.t. Tab_P and S . It is trivial to see that P quasi-terminates w.r.t. Tab_P and S . It is also easy to see that, since, for every A such that $\tilde{A} \in S$, the LG-forest for $P \cup \{\leftarrow A\}$ consists of a finite number of finite trees, the set of computed answers for atoms in $Call(P, \{A\})$ is finite.

\Leftarrow : Suppose that P quasi-terminates w.r.t. Tab_P and S and for all $A \in S$ the set of (LD-)computed answers for atoms in $Call(P, \{A\}) \cap B_{TR_P}^E$ is finite. We prove that P LG-terminates w.r.t. Tab_P and S . Let A be an atom such that $\tilde{A} \in S$. We already know that the LG-forest \mathcal{F} of $P \cup \{\leftarrow A\}$ consists of a finite number of LG-trees without infinite branches. We prove by contradiction that these LG-trees are finitely branching. Suppose there is an LG-tree in \mathcal{F} which is infinitely branching. Then, there is an LG-tree in \mathcal{F} with an infinitely branching node, which contains a query which has a tabled, recursive atom on the leftmost position. That is, there is an atom in $Call(P, \{A\}) \cap B_{TR_P}^E$ which has infinitely many computed answers. This gives a contradiction. \square

It follows from the proof that, if P LG-terminates w.r.t. Tab_P and S , the set of computed answers for atoms in $Call(P, \{A\})$ is finite for all $A \in S$.

5.1 Characterisation of LG-Termination

In this subsection, we give a characterisation of LG-termination. First (Theorem 5.1), we will characterise LG-termination of a program P in terms of quasi-termination of the program P^a , which is obtained by applying the answer-transformation (Definition 5.2) on P . However, we will also characterise LG-termination in a more direct way (Theorem 5.2).

Lemma 5.2 of the previous subsection gives the intuition behind the answer-transformation of the following Definition 5.2. The answer-transformation forms the basis of the characterisation of LG-termination in Theorem 5.1; LG-termination of a program P will be shown equiva-

lent with quasi-termination of the program P^a , obtained by applying the answer-transformation to P .

Definition 5.2 (a(nswer)-transformation) *Let P be a program. The a-transformation is defined as follows:*

- For a clause $C = H \leftarrow B_1, \dots, B_n$ in P , we define

$$C^a = H \leftarrow B_1, B_1^*, \dots, B_n, B_n^*$$

with B_i^* defined as follows (suppose $B_i = p(t_1, \dots, t_n)$):

if $p \simeq \text{Rel}(H)$ and $p \in \text{Tab}_P$, then $B_i^* = p^a(t_1, \dots, t_n)$, where p^a/n is a new predicate, else $B_i^* = \emptyset$.

Let $\text{TR}_P^a = \{p^a/n \mid p/n \in \text{TR}_P\}$ (recall that $\text{TR}_P = \text{Tab}_P \cap \text{Rec}_P$).

- For the program P , we define

$$P^a = \{C^a \mid C \in P\} \cup \{p^a(X_1, \dots, X_n) \leftarrow \mid p^a/n \in \text{TR}_P^a\}.$$

- The set of tabled predicates of the program P^a is defined as

$$\text{Tab}_{P^a} = \text{Tab}_P \cup \text{TR}_P^a.$$

It is easy to see that $\text{Call}(P, S) = \text{Call}(P^a, S) \cap B_P^E$. Also, if we denote with $\text{cas}(P, \{p(\bar{t})\})$ the set of computed answer substitutions of $P \cup \{\leftarrow p(\bar{t})\}$, then $\text{cas}(P, \{p(\bar{t})\}) = \text{cas}(P^a, \{p(\bar{t})\})$ for all $p(\bar{t}) \in B_P^E$. It is important to note that, if we have a query $p(\bar{t}) \in B_{\text{TR}_P^a}^E$ to the program P , then $p(\bar{t})\sigma$ is a computed answer if $p^a(\bar{t})\sigma \in \text{Call}(P^a, \{p(\bar{t})\})$. This is in fact the main purpose of the transformation.

We want to mention that a similar transformation, namely the solution-transformation, is introduced in [10, Definition 4.2] in order to relate the concepts of LG-termination and quasi-termination. But, as opposed to the answer-transformation, the solution-transformation introduces much more “overhead” in the sense that a clause $C = H \leftarrow B_1, \dots, B_n$ is transformed into a clause $C_{\text{sol}} = H \leftarrow B_1, \text{sol}(B_1), \dots, B_n, \text{sol}(B_n)$ where $\text{sol}/1$ is a new tabled predicate. Notice that in contrast, the answer-transformation only keeps track of the computed answers of *recursive, tabled* body atoms (and not of all body atoms).

Example 5.3 *Recall the program P of Example 3.1, with $\text{Tab}_P = \{\text{member}/2\}$. The a-transformation of P is the following program P^a :*

$$\begin{cases} \text{intersection}(Xs, Ys, Z) & \leftarrow \text{member}(Xs, Z), \text{member}(Ys, Z) \\ \text{member}([Z|Zs], Z) & \leftarrow \\ \text{member}([X|Zs], Z) & \leftarrow \text{member}(Zs, Z), \text{member}^a(Zs, Z) \\ \text{member}^a(L, Z) & \leftarrow \end{cases}$$

with $\text{Tab}_{P^a} = \{\text{member}/2, \text{member}^a/2\}$.

The following theorem is a generalisation of [10, Theorem 4.1] (there, the previously mentioned solution-transformation of [10, Definition 4.2] is used to relate LG-termination and quasi-termination in case $\text{Tab}_P = \text{Pred}_P$).

Theorem 5.1 (characterisation of LG-termination in terms of quasi-termination)

Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$.

P LG-terminates w.r.t. Tab_P and S iff P^a quasi-terminates w.r.t. Tab_{P^a} and S .

Proof \Leftarrow : Suppose P^a is quasi-terminating w.r.t. Tab_{P^a} and S . Let A be an atom such that $\bar{A} \in S$. Let \mathcal{F} be the LG-forest w.r.t. Tab_P of $P \cup \{\leftarrow A\}$. We prove that \mathcal{F} consists of a finite number of finite LG-trees.

We know that the LG-forest \mathcal{F}^a w.r.t. Tab_{P^a} of $P^a \cup \{\leftarrow A\}$ is a finite set of LG-trees, without infinite branches. It is easy to see that hence, \mathcal{F} consists also of a finite number of trees without infinite branches. We prove that the LG-trees in \mathcal{F} are finitely branching. Suppose this is not the case, i.e. there is an LG-tree in \mathcal{F} which is infinitely branching. Then, there is an LG-tree T in \mathcal{F} which is infinitely branching in a non-root node, which is a goal with leftmost atom $p(t_1, \dots, t_n)$, with $p \in TR_P$, which is directly descending from an atom $q(s_1, \dots, s_m)$, with $p \simeq q$, via a recursive clause $C = q(u_1, \dots, u_m) \leftarrow \dots, p(v_1, \dots, v_n), \dots$. Let T^a be the LG-tree in \mathcal{F}^a corresponding to T . Note that the clause C^a instead of C is used in T^a . Because of this, the atom on the right of $p(t_1, \dots, t_n)$ in the infinitely branching node is $p^a(t_1, \dots, t_n)$. Thus, \mathcal{F}^a consists of a infinite number of LG-trees (there are an infinite number of LG-trees with predicate p^a in the root). But this gives a contradiction.

\Rightarrow : Suppose P is LG-terminating w.r.t. Tab_P and S . Reasoning the other way around, we can prove that P^a is quasi-terminating w.r.t. Tab_{P^a} and S . \square

Example 5.4 Recall Example 5.3. The LG-forest of $P \cup \{\leftarrow intersection(Xs, Ys, a)\}$ w.r.t. Tab_P was shown in Figure 3. Note that the trees are infinitely branching; P doesn't LG-terminate w.r.t. Tab_P and $\{intersection(Xs, Ys, a)\}$.

In Figure 7, the LG-forest of the program P^a and $\{intersection(Xs, Ys, a)\}$ w.r.t. Tab_{P^a} is shown. Note that there are infinitely many LG-trees in the forest; P^a doesn't quasi-terminate w.r.t. Tab_{P^a} and $\{intersection(Xs, Ys, a)\}$.

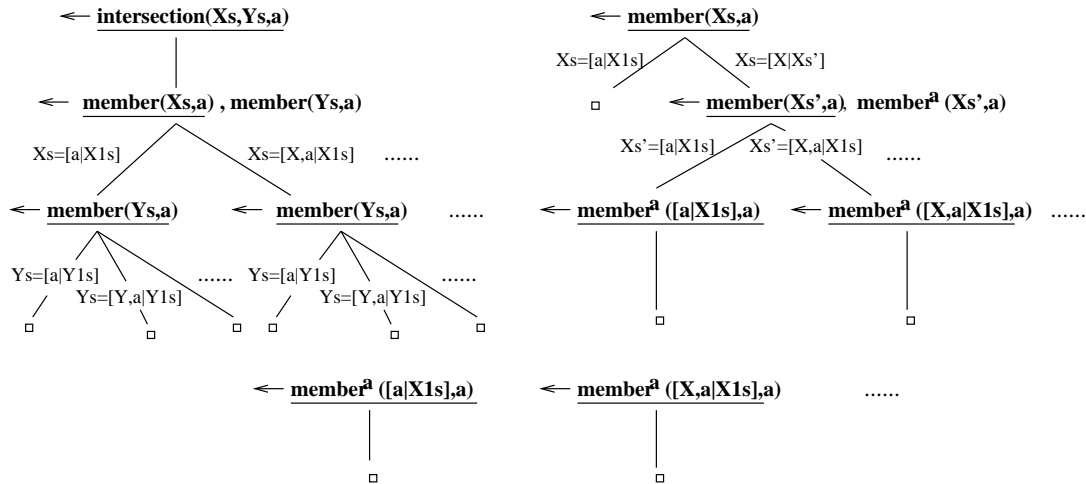


Figure 7: The LG-forest for $P^a \cup \{\leftarrow intersection(Xs, Ys, a)\}$.

Theorem 5.1 provides us with a characterisation of LG-termination of a program in terms of quasi-termination. That is, to prove LG-termination of P w.r.t. Tab_P and S , we have to prove quasi-termination of P^a , the a-transformation of the program P , w.r.t. Tab_{P^a} and S . To

prove quasi-termination, it is sufficient (and also necessary in case the tabling is well-chosen¹) to prove the quasi-acceptability of P^a w.r.t. Tab_{P^a} and S . However, the condition of quasi-acceptability on P^a can be weakened; i.e. some of the decreases “ $|A| \geq |B_i\theta_{i-1}|$ ” need not be checked because they can always be fulfilled. In particular, we only have to require the non-strict decrease for recursive, tabled body atoms B_i (to obtain an LG-forest with only finitely many LG-trees) or for body atoms B_i of the form $p^a(t_1, \dots, t_n)$ (to obtain LG-trees which are finitely branching); the conditions on non-tabled predicates remain the same. The following notion of LG-acceptability gives this optimised condition for LG-termination of a program. So, instead of proving quasi-acceptability of the program P^a , we can prove LG-acceptability of P (and this is a less strong condition).

Definition 5.3 (LG-acceptability) *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$. P is LG-acceptable w.r.t. Tab_P and S iff there is a level mapping $|\cdot|$ on $B_{P^a}^E$ such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $Call(P^a, \{A\}) \cap B_{TR_P \cup TR_P^a}^E$, and such that*

- for every atom A such that $\tilde{A} \in Call(P^a, S)$,
- for every clause $H \leftarrow B_1, \dots, B_n$ in P^a , such that $mgu(A, H) = \theta$ exists,
- for every B_i such that $Rel(B_i) \simeq Rel(H)$ or $Rel(B_i) \in TR_P^a$,
- for every LD-c.a.s. θ_{i-1} in P^a for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$\begin{aligned} &|A| \geq |B_i\theta_{i-1}| \\ &\text{and} \\ &|A| > |B_i\theta_{i-1}| \quad \text{if } Rel(A) \simeq Rel(B_i) \in NTab_P \text{ and} \\ & \quad C_2(Rel(A), Rel(B_i)) \text{ does not hold.} \end{aligned}$$

Theorem 5.2 (characterisation of LG-termination) *Let P be a program, $Tab_P \subseteq Pred_P$ and $S \subseteq B_P^E$.*

If P is LG-acceptable w.r.t. Tab_P and S , then P LG-terminates w.r.t. Tab_P and S .

If the tabling Tab_P is well-chosen w.r.t. P , then also the converse holds, i.e. P is LG-acceptable w.r.t. Tab_P and S iff P LG-terminates w.r.t. Tab_P and S .

Proof \Rightarrow : Suppose that P is LG-acceptable w.r.t. Tab_P and S . We prove that P LG-terminates w.r.t. Tab_P and S .

Let A be an atom such that $\tilde{A} \in S$. Let \mathcal{F} be the LG-forest w.r.t. Tab_P of $P \cup \{\leftarrow A\}$.

We prove that \mathcal{F} consists of a finite number of finite LG-trees.

- The LG-trees in \mathcal{F} are finitely branching.

Suppose this is not the case, i.e. there is an LG-tree in \mathcal{F} which is infinitely branching. Then, there is an LG-tree T in \mathcal{F} which is infinitely branching in a non-root node, which is a query with leftmost atom $p(t_1, \dots, t_n)$, with $p \in TR_P$, which is directly descending from an atom $q(s_1, \dots, s_m)$, with $p \simeq q$, via a recursive clause $C = q(u_1, \dots, u_m) \leftarrow \dots, p(v_1, \dots, v_n), \dots$. Now, consider the LG-forest \mathcal{F}^a of $P^a \cup \{\leftarrow A\}$. Let T^a be the LG-tree in \mathcal{F}^a corresponding to T . Note that the clause C^a instead of C is used in T^a . Because of this, the atom on the right of $p(t_1, \dots, t_n)$

¹note that if Tab_P is well-chosen w.r.t. P , then also Tab_{P^a} is well-chosen w.r.t. P^a .

in the infinitely branching node is $p^a(t_1, \dots, t_n)$. Thus, \mathcal{F}^a consists of a infinite number of LG-trees (there are an infinite number of LG-trees with predicate p^a in the root). But, all these p^a -atoms directly descend from the node $q(s_1, \dots, s_m)$ via the clause C^a in P^a and hence, because of the LG-acceptability condition, their value under the level mapping $|\cdot|$ is smaller or equal to $|q(s_1, \dots, s_m)|$. Because $|\cdot|$ is finitely partitioning on $Call(P^a, \{A\}) \cap B_{TR^a}^E$, this gives a contradiction.

- \mathcal{F} consists of a finite number of LG-trees, i.e. $\sharp(Call(P, \{A\}) \cap B_{Tab_P}^E) < \infty$.
Suppose this is not the case. A first possible reason for an infinite number of LG-trees in \mathcal{F} is an infinitely branching LG-tree in \mathcal{F} . But we already proved that this does not occur. The other possibility is that there exists an infinite LD-derivation of $\leftarrow A$ in P which contains an infinite directed subsequence, such that this infinite directed subsequence has a tail G_n, G_{n+1}, \dots with $G_i = \leftarrow A_i, \mathbf{B}_i, i \geq n$, such that $\{\tilde{A}_i \mid i \geq n\} \subseteq B_{Tab_P}^E$ is an infinite set and $Rel(A_i) \simeq Rel(A_{i+1})$ for all $i \geq n$. So, $\{\tilde{A}_i \mid i \geq n\} \subseteq B_{TR_P}^E$. Since $\tilde{A}_i \in Call(P, \{A\}) \cap B_{TR_P}^E \subseteq Call(P^a, \{A\}) \cap B_{TR_P \cup TR_P^a}^E$, and since $|\cdot|$ is finitely partitioning on this set and $|A_i| \geq |A_{i+1}|$ for all $i \geq n$ (by the LG-acceptability condition), this gives a contradiction.
- The LG-trees in \mathcal{F} have finite branches.
The same argumentation as in the proof of Theorem 4.1 can be applied here.

\Leftarrow : Suppose that the tabling Tab_P is well-chosen w.r.t. P and suppose that P LG-terminates w.r.t. Tab_P and S . We prove that there exists a level mapping $|\cdot|$ such that P is LG-acceptable w.r.t. Tab_P, S and this level mapping $|\cdot|$.

Since P LG-terminates w.r.t. Tab_P and S , we know by Theorem 5.1 that P^a quasi-terminates w.r.t. Tab_{P^a} and S . Note that, since Tab_P is well-chosen w.r.t. P , Tab_{P^a} is well-chosen w.r.t. P^a . By Theorem 4.1, there exists a level mapping $|\cdot|$ such that P^a is quasi-acceptable w.r.t. Tab_{P^a}, S and this level mapping $|\cdot|$. It is straightforward to verify that P is LG-acceptable w.r.t. Tab_P, S and this level mapping $|\cdot|$ restricted to B_P^E . (Note that, as we already discussed in the beginning of this subsection, the level mapping obtained in this way satisfies more conditions than required by the notion of LG-acceptability.) \square

Example 5.5 Recall the part of the grammar program (Section 2) which recognizes the language $a^n b$:

$$R : \begin{cases} s(Si, So) \leftarrow a(Si, S), S = [b|So] \\ a(Si, So) \leftarrow a(Si, S), a(S, So) \\ a(Si, So) \leftarrow Si = [a|So] \end{cases}$$

with $Tab_R = \{a/2\}$. We show that R LG-terminates w.r.t. $\{a/2\}$ and $S = \{s(si, So)\}$ where si is a ground list consisting of atoms and So is a variable. Consider the following a -transformation, R^a , of R ($Tab_{R^a} = \{a/2, a^a/2\}$):

$$R^a : \begin{cases} s(Si, So) \leftarrow a(Si, S), S = [b|So] \\ a(Si, So) \leftarrow a(Si, S), a^a(Si, S), a(S, So), a^a(S, So) \\ a(Si, So) \leftarrow Si = [a|So] \\ a^a(Si, So) \leftarrow \end{cases}$$

When applying Theorem 5.2, we only have to consider the second clause of R^a . Note that, for all $a(t_1, t_2) \in Call(R^a, \{s(si, So)\})$, t_1 is a sublist of si and t_2 is a variable. Also, for

all $a^a(v_1, v_2) \in \text{Call}(R^a, \{s(si, So)\})$, v_1 is a sublist of si and v_2 is a (strict) sublist of v_1 . Consider the trivial level mapping $|\cdot|$ (mapping everything to 0) on $\text{Call}(R^a, \{s(si, So)\}) \cap B_{\{a/2, a^a/2\}}^E$. Since this set is finite, $|\cdot|$ is obviously finitely partitioning on this set. R and S , together with $|\cdot|$, satisfy the conditions of Theorem 5.2. Hence, R LG-terminates w.r.t. $\{a/2\}$ and S .

5.2 Modular Proofs for LG-Termination

Similarly to Section 4.2, we want to be able to obtain modular termination proofs for LG-termination of the union $P \cup R$ of two programs P and R , where P extends R . Note that, because of Theorem 5.1, and because $(P \cup R)^a = P^a \cup R^a$ (if P extends R), we can use the modular proofs for quasi-termination of Section 4.2. However, as we already noted in Section 5.1, we can give optimised conditions which require less checking for decreases between the values under the level mapping of the head and body atoms. Again we put for $\text{Pred}_{P \cup R} = \text{Tab}_{P \cup R} \sqcup \text{NTab}_{P \cup R}$

$$\begin{aligned} \text{Tab}_P &= \text{Tab}_{P \cup R} \cap \text{Pred}_P & , & \quad \text{NTab}_P = \text{NTab}_{P \cup R} \cap \text{Pred}_P, \\ \text{Tab}_R &= \text{Tab}_{P \cup R} \cap \text{Pred}_R & , & \quad \text{NTab}_R = \text{NTab}_{P \cup R} \cap \text{Pred}_R. \end{aligned}$$

The following propositions give modular conditions for LG-termination of $P \cup R$ without using Theorem 5.1.

Proposition 5.2 *Let P and R be two programs, such that P extends R . Let $S \subseteq B_{P \cup R}^E$. If*

- R LG-terminates w.r.t. Tab_R and $\text{Call}(P \cup R, S)$,
- there is a level mapping $|\cdot|$ on $B_{P^a}^E$ such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $\text{Call}(P^a \cup R, \{A\}) \cap B_{TR_P \cup TR_R^a}^E$, and such that
 - for every atom A such that $\tilde{A} \in \text{Call}(P^a \cup R, S)$,
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P^a such that $\text{mgu}(A, H) = \theta$ exists,
 - for every B_i such that $\text{Rel}(B_i) \simeq \text{Rel}(H)$ or $\text{Rel}(B_i) \in TR_P^a$,
 - for every LD-c.a.s. θ_{i-1} in $P^a \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$|A| \geq |B_i \theta \theta_{i-1}|$$

and

$$|A| > |B_i \theta \theta_{i-1}| \quad \text{if } \text{Rel}(A) \simeq \text{Rel}(B_i) \in \text{NTab}_P \text{ and } C_2(\text{Rel}(A), \text{Rel}(B_i)) \text{ does not hold}$$

then $P \cup R$ LG-terminates w.r.t. $\text{Tab}_{P \cup R}$ and S .

Proof Let A be an atom such that $\tilde{A} \in S$. Let \mathcal{F} be the LG-forest w.r.t. $\text{Tab}_{P \cup R}$ of $P \cup R \cup \{\leftarrow A\}$. We prove that \mathcal{F} consists of a finite number of finite LG-trees.

If A is defined in R , this follows from the fact that R LG-terminates w.r.t. Tab_R and $\text{Call}(P \cup R, S)$. So, suppose that A is defined in P . The proof that \mathcal{F} is finitely branching, \mathcal{F} consists of a finite number of LG-trees and \mathcal{F} contains no trees with infinite branches is a simple adaptation of the proof of the if-direction of Theorem 5.2 (the adaptation is similar to the adaptation needed to transform the proof of the if-direction of Theorem 4.1 into a proof of Proposition 4.1). \square

Next, we consider three special cases of Proposition 5.2. In Proposition 5.3, we consider the case in which no defined predicate in P is tabled. In Proposition 5.4, we consider the case in which all defined predicates in P are tabled. Finally, in Proposition 5.5, we consider the case of two programs extending each other.

Proposition 5.3 *Let P and R be two programs, such that P extends R and such that $Def_P \subseteq NTab_P$. Let $S \subseteq B_{P \cup R}^E$. If*

- R LG-terminates w.r.t. Tab_R and $Call(P \cup R, S)$,
- there is a level mapping $|\cdot|$ on B_P^E such that
 - for every atom A such that $\tilde{A} \in Call(P \cup R, S)$,
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P such that $mgu(A, H) = \theta$ exists,
 - for every B_i such that $Rel(B_i) \simeq Rel(A)$,
 - for every LD-c.a.s. θ_{i-1} in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$|A| > |B_i \theta \theta_{i-1}|$$

then $P \cup R$ LG-terminates w.r.t. $Tab_{P \cup R}$ and S .

Proof Because no defined predicate in P is tabled, $P^a = P$. Also, for all $p, q \in NTab_P \cap Def_P$ with $p \simeq q$, $C_1(p, q)$ holds. The proposition follows then from Proposition 5.2. \square

Example 5.6 *Recall program R of Example 5.5. Let P be the following program which parses the language $a^n b$ (see also Section 2):*

$$P : \begin{cases} s(Si, So, P) \leftarrow a(Si, S), S = [b|So], P = spt(Pa, b), a(Si, S, Pa) \\ a(Si, So, P) \leftarrow a(Si, S), a(S, So), P = apt(P1, P2), a(Si, S, P1), \\ \quad \quad \quad a(S, So, P2) \\ a(Si, So, P) \leftarrow Si = [a|So], P = a \end{cases}$$

As already noted, P extends R and the only tabled predicate in $P \cup R$ is $a/2$ – see Section 2 for why this tabling is sufficient.

Let $S = \{s(si, So, P)\}$ where si is a ground list consisting of atoms, and So, P are distinct variables. We show, by using Proposition 5.3, that $P \cup R$ LG-terminates w.r.t. $\{a/2\}$ and S .

- R LG-terminates w.r.t. $\{a/2\}$ and $Call(P \cup R, S)$.

Note that, if $a(t1, t2) \in Call(P \cup R, S)$, then either $t1$ is a sublist of si and $t2$ is a variable, or $t1$ and $t2$ are both sublists of si . In Example 5.5, we proved that R LG-terminates w.r.t. this first kind of queries. To prove that R LG-terminates w.r.t. the second kind of queries, we can again apply Theorem 5.2. Since the proof is similar to the one given in Example 5.5, we omit it here.

- Note first that, if $a(t1, t2, P) \in Call(P \cup R, S)$, then $t2$ is a (strict) sublist of $t1$, $t1$ is a sublist of si and P is a variable. Let $|\cdot|$ be the following level mapping on $Call(P \cup R, S) \cap B_{\{a/3\}}^E$: $|a(t1, t2, P)| = \|t1\|_l - \|t2\|_l$, where $\|\cdot\|_l$ is the list-length norm. Because of the remark above, $|\cdot|$ is well-defined. Note that we only have to consider the recursive clause for $a/3$ in the analysis.

- Consider the fourth body atom in the recursive clause for $a/3$. If this clause is called with $a(ti, to, P)$, with to a (strict) sublist of ti , then the fourth body atom is called as $a(ti, t, P1)$ where to is a (strict) sublist of t and t is a (strict) sublist of ti . Hence, $|a(ti, to, P)| = \| ti \|_l - \| to \|_l > \| ti \|_l - \| t \|_l = |a(ti, t, P1)|$.
- Consider the last body atom. If the recursive clause is called with $a(ti, to, P)$, with to a (strict) sublist of ti , then the last body atom is called as $a(t, to, P2)$ where to is a (strict) sublist of t and t is a (strict) sublist of ti . Hence, $|a(ti, to, P)| = \| ti \|_l - \| to \|_l > \| t \|_l - \| to \|_l = |a(t, to, P2)|$.

We conclude that $P \cup R$ and S satisfy the condition of Proposition 5.3, so $P \cup R$ LG-terminates w.r.t. $\{a/2\}$ and S .

In the next proposition, a modular termination proof for the LG-termination of the union $P \cup R$ is given, where P extends R and all defined predicates in P are tabled.

Proposition 5.4 *Let P and R be two programs, such that P extends R and such that $Def_P \subseteq Tab_P$. Let $S \subseteq B_{P \cup R}^E$. If*

- R LG-terminates w.r.t. Tab_R and $Call(P \cup R, S)$,
- there is a level mapping $|\cdot|$ on $B_{P^a}^E$ such that for all A such that $\tilde{A} \in S$, $|\cdot|$ is finitely partitioning on $Call(P^a \cup R, \{A\}) \cap B_{TR_P \cup TR_P^a}^E$, and such that
 - for every atom A such that $\tilde{A} \in Call(P^a \cup R, S)$,
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P^a such that $mgu(A, H) = \theta$ exists,
 - for every B_i such that $Rel(B_i) \simeq Rel(H)$ or $Rel(B_i) \in TR_P^a$,
 - for every LD-c.a.s. θ_{i-1} in $P^a \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$|A| \geq |B_i \theta \theta_{i-1}|$$

then $P \cup R$ LG-terminates w.r.t. $Tab_{P \cup R}$ and S .

Proof This is a direct corollary of Proposition 5.2 (every recursive predicate in P is defined in P and hence tabled). \square

Finally, we consider the case of two programs P_1 and P_2 extending each other.

Proposition 5.5 *Let P_1, P_2 be two programs such that P_1 extends P_2 and P_2 extends P_1 . Let $S \subseteq B_{P_1 \cup P_2}^E$. If*

- P_1 LG-terminates w.r.t. Tab_{P_1} and $S \cap B_{P_1}^E$,
- P_2 LG-terminates w.r.t. Tab_{P_2} and $S \cap B_{P_2}^E$,

then $P_1 \cup P_2$ LG-terminates w.r.t. $Tab_{P_1 \cup P_2}$ and S .

Proof Because P_1 extends P_2 and P_2 extends P_1 , $Call(P_1 \cup P_2, S) \cap B_{P_i}^E = Call(P_i, S \cap B_{P_i}^E)$, for $i = 1, 2$. The proposition follows then by definition of LG-termination. \square

6 Related Papers and Future Work

Our work is based on, and significantly extends, the results of [10]. In [10], the two notions of (universal) termination under tabled execution, namely quasi-termination and LG-termination, are introduced and characterised. As opposed to [10], where it is assumed that all predicates in the program are tabled, we here consider programs with a mix of tabled and Prolog execution, thereby providing a termination framework for ‘real’ tabled programs. We further extend the applicability of this framework by presenting modular termination conditions: conditions ensuring termination of the union $P \cup R$ of two programs P and R , where P extends R .

Termination proofs for (S)LD-resolution (such as e.g. those surveyed in [7]) are sufficient to prove termination under a tabled execution mechanism, but, since there are quasi-terminating and LG-terminating programs, which are not LD-terminating, more effective proof techniques can and need to be found. Besides [10], there are only relatively few works studying termination under tabling. In the context of well-moded programs, [17] presents a sufficient condition for a program to have the bounded term-size property, which implies LG-termination. [13] provides another sufficient condition for quasi-termination in the context of functional programming. In parallel with the work reported on in this paper, an orthogonal extension of the work of [10] was investigated in [19]. Namely, in [19] the constraint-based approach of [9] for automatically proving LD-termination was extended to the case of quasi-termination and LG-termination. More specifically, in the context of simply moded, well-typed programs and queries, sufficient conditions for quasi-termination and LG-termination (in the case that $Tab_P = Pred_P$) are given. These conditions allow reasoning fully at the clause level, contrary to those in the current paper which are stated for sets of calls. An integration of these two extensions of [10] is straightforward.

A topic for future research is to extend our results to *normal* logic programs executed under a mix of Prolog and tabled execution. Another, with an arguably more practical flavour, is to investigate how the termination conditions presented here can form the basis of a compiler that automatically decides on — or at least guides a programmer in choosing — a tabling (i.e. a set of tabled predicates) for an input program such that quasi-termination of the program is ensured.

Acknowledgements

Sofie Verbaeten is Research Assistant of the Fund for Scientific Research - F.W.O. Flanders, Belgium. Konstantinos Sagonas was supported by GOA ”LP+, a second generation logic programming language”. Danny De Schreye is Senior Research Associate of F.W.O. Flanders.

References

- [1] K.R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of theoretical computer science, Vol. B*. Elsevier Science Publishers, 1990.
- [2] K.R. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, 1993.
- [3] K.R. Apt and D. Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106(1):109–157, 1993.
- [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] R. Bol and L. Degerstedt. The underlying search for magic templates and tabulation. In D. S. Warren, editor, *Proceedings of the Tenth International Conference on Logic Programming*, pages 793–811, Budapest, Hungary, june 1993. The MIT Press.
- [6] W. Chen and D. S. Warren. Tabled Evaluation with Delaying for General Logic Programs. *Journal of the ACM*, 43(1):20–74, january 1996.
- [7] 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.
- [8] 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.
- [9] S. Decorte, D. De Schreye, and H. Vandecasteele. Constraint-based automatic termination analysis for logic programs. *ACM TOPLAS*. to appear.
- [10] Stefaan Decorte, Danny De Schreye, Michael Leuschel, Bern Martens, and Konstantinos Sagonas. Termination Analysis for Tabled Logic Programming. In Norbert Fuchs, editor, *Proceedings of LOPSTR'97: Logic Program Synthesis and Transformation*, number 1463 in LNCS, pages 107–123. Springer-Verlag, 1997.
- [11] Jay Early. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM*, 13(2):94–102, 1970.
- [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] C. K. Holst. Finiteness Analysis. In John Hughes, editor, *Proceedings of the 5th ACM Conference on Functional Programming Languages and Computer Architecture (FPCA)*, number 523 in LNCS, pages 473–495. Springer-Verlag, august 1991.
- [14] T. Kanamori and T. Kawamura. OLDT-based abstract interpretation. *Journal of Logic Programming*, 15(1 & 2):1–30, january 1993.
- [15] M. Leuschel, B. Martens, and K. Sagonas. Preserving termination of tabled logic programs while unfolding. In *Proceedings of LOPSTR'97: Logic Program Synthesis and Transformation*, Leuven, Belgium, july 1997.
- [16] J.W. Lloyd. *Foundations of logic programming*. Springer-Verlag, 1987.
- [17] L. Plümer. *Termination proofs for logic programs*. Number 446 in LNAI. Springer-Verlag, 1990.
- [18] H. Tamaki and T. Sato. OLD Resolution with Tabulation. In *Proceedings ICLP'86*, Lecture Notes in Computer Science 225, pages 84–98. Springer Verlag, 1986.
- [19] S. Verbaeten and D. De Schreye. Termination analysis of tabled logic programs using mode and type information. Technical Report 277, Department of Computer Science, K.U.Leuven. Available at <http://www.cs.kuleuven.ac.be/~sofie>.
- [20] L. Vieille. Recursive query processing: the power of logic. *Theoretical Computer Science*, 69(1):1–53, 1989.
- [21] David S. Warren. Notes for “Programming in Tabled Prolog”. Early draft available at <http://www.cs.sunysb.edu/~warren/>, 1998.

A Construction of the Level Mapping in a Modular Termination Proof

In this appendix, we present modular proofs for the quasi-termination of the program $P \cup R$, where P extends R , by constructing a level mapping such that $P \cup R$ is quasi-acceptable w.r.t. this level mapping (see Definition 4.3 and Theorem 4.1). These modular termination conditions are of a similar nature as the ones in [4], where modular proofs are given for SLD-termination (i.e. termination under SLD-resolution w.r.t. all selection rules) and LD-termination. First, we give some properties of finitely partitioning level mappings.

Lemma A.1 • *Let P be a program and $L \subseteq B_P^E$. Let $f, g : L \rightarrow \mathbb{N}$ be level mappings. If f is finitely partitioning on $C \subseteq L$, then $f + g : L \rightarrow \mathbb{N} : A \mapsto (f + g)(A) = f(A) + g(A)$ is finitely partitioning on C .*

- *Let P_1, P_2 be two programs and $L_1 \subseteq B_{P_1}^E, L_2 \subseteq B_{P_2}^E$. Let $f_1 : L_1 \rightarrow \mathbb{N}$ and $f_2 : L_2 \rightarrow \mathbb{N}$ be level mappings. If f_1 , resp. f_2 , are finitely partitioning on $C_1 \subseteq L_1$, resp. $C_2 \subseteq L_2$, then $m(f_1, f_2) : L_1 \cup L_2 \rightarrow \mathbb{N} :$*

$$A \mapsto m(f_1, f_2)(A) := \begin{cases} \min(f_1(A), f_2(A)) & , A \in L_1 \cap L_2 \\ f_1(A) & , A \in L_1 \setminus L_2 \\ f_2(A) & , A \in L_2 \setminus L_1 \end{cases}$$

is finitely partitioning on $C_1 \cup C_2$.

Proof • Let $n \in \mathbb{N}$. We prove that $\sharp((f + g)^{-1}(n) \cap C) < \infty$.

$$\begin{aligned} (f + g)^{-1}(n) \cap C &= \{A \in C \mid (f + g)(A) = n\} \\ &\subseteq \{A \in C \mid f(A) \leq n\} \\ &= \bigcup_{0 \leq m \leq n} \{A \in C \mid f(A) = m\} \end{aligned}$$

and this last set is finite.

- Let $n \in \mathbb{N}$. We prove that $\sharp(m(f_1, f_2)^{-1}(n) \cap (C_1 \cup C_2)) < \infty$.

$$\begin{aligned} m(f_1, f_2)^{-1}(n) \cap (C_1 \cup C_2) &= \{A \in C_1 \cup C_2 \mid m(f_1, f_2)(A) = n\} \\ &= \{A \in C_1 \setminus C_2 \mid f_1(A) = n\} \cup \\ &\quad \{A \in C_2 \setminus C_1 \mid f_2(A) = n\} \cup \\ &\quad \{A \in C_1 \cap C_2 \mid \min(f_1(A), f_2(A)) = n\} \end{aligned}$$

It is obvious that the first two sets in the union are finite (f_1 and f_2 are finitely partitioning on C_1 , resp. C_2). The third set $\{A \in C_1 \cap C_2 \mid \min(f_1(A), f_2(A)) = n\}$ is finite, because it is a subset of the finite set $\{A \in C_1 \cap C_2 \mid f_1(A) = n\} \cup \{A \in C_1 \cap C_2 \mid f_2(A) = n\}$.

□

We next give a modular termination condition for the quasi-termination of $P \cup R$ where P extends R , by constructing a level mapping, from simpler ones, such that $P \cup R$ is quasi-acceptable w.r.t. this level mapping. Notice that we consider the same case as in Proposition 4.1.

Proposition A.1 *Let P, R be two programs such that P extends R . Let $S \subseteq B_{P \cup R}^E$. If*

1. R is quasi-acceptable w.r.t. Tab_R , $Call(P \cup R, S)$ and the level mapping l_R , defined on B_R^E and finitely partitioning on $Call(P \cup R, S) \cap B_{Tab_R}^E$,
2. there is a level mapping l_P defined on $B_P^E \setminus B_R^E$ and finitely partitioning on $Call(P \cup R, S) \cap B_{Tab_P \setminus Tab_R}^E$ such that
 - for every atom A such that $\tilde{A} \in Call(P \cup R, S)$,
 - for every clause $H \leftarrow B_1, \dots, B_n$ in P , such that $mgu(A, H) = \theta$ exists,
 - for every $1 \leq i \leq n$ such that $B_i \in B_P^E \setminus B_R^E$ and for every LD-c.a.s. θ_{i-1} in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$\begin{aligned}
& l_P(A) \geq l_P(B_i\theta\theta_{i-1}) \\
& \text{and} \\
& l_P(A) > l_P(B_i\theta\theta_{i-1}) \quad \text{if } Rel(A) \simeq Rel(B_i) \in NTab_P \text{ and} \\
& \quad C_2(Rel(A), Rel(B_i)) \text{ does not hold,}
\end{aligned}$$

3. there exists a level mapping $\| \cdot \|_P$ on $B_P^E \setminus B_R^E$ such that

- for every atom A such that $\tilde{A} \in Call(P \cup R, S)$,
- for every clause $H \leftarrow B_1, \dots, B_n$ in P , such that $mgu(A, H) = \theta$ exists,
- for every $1 \leq i \leq n$ and every LD-c.a.s. θ_{i-1} in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$:

$$\| A \|_P \geq \begin{cases} \| B_i\theta\theta_{i-1} \|_P & , \quad B_i\theta\theta_{i-1} \in B_P^E \setminus B_R^E \\ l_R(B_i\theta\theta_{i-1}) & , \quad B_i\theta\theta_{i-1} \in B_R^E \end{cases}$$

then, for the following level mapping l defined on $B_{P \cup R}^E$ and finitely partitioning on $Call(P \cup R, S) \cap B_{Tab_{P \cup R}}^E$

$$l(A) = \begin{cases} l_P(A) + \| A \|_P & \text{if } A \in B_P^E \setminus B_R^E, \\ l_R(A) & \text{if } A \in B_R^E, \end{cases}$$

$P \cup R$ is quasi-acceptable w.r.t. $Tab_{P \cup R}$, S , and the level mapping l . Hence, $P \cup R$ quasi-terminates w.r.t. $Tab_{P \cup R}$ and S .

Proof Because of Lemma A.1, l is finitely partitioning on $Call(P \cup R, S) \cap B_{Tab_{P \cup R}}^E$. We prove that $P \cup R$ is quasi-acceptable w.r.t. $Tab_{P \cup R}$, S and the level mapping l (see Definition 4.3).

Let A be an atom such that $\tilde{A} \in Call(P \cup R, S)$. Let $H \leftarrow B_1, \dots, B_n$ be a clause of $P \cup R$ such that $mgu(A, H) = \theta$ exists. Let θ_{i-1} be an LD-c.a.s. in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$. There are two cases to consider:

- A is defined in R , $l(A) = l_R(A)$.

Then, because of condition 1 in the proposition statement, $l_R(A) \geq l_R(B_i\theta\theta_{i-1})$ (note that because P extends R , for a clause $H \leftarrow B_1, \dots, B_n$ in R and $mgu(A, H) = \theta$, an LD-c.a.s. in $P \cup R$ for $\leftarrow (B_1, \dots, B_{i-1})\theta$ is the same as an LD-c.a.s. for $\leftarrow (B_1, \dots, B_{i-1})\theta$ in R only). Since $A, B_i\theta\theta_{i-1} \in B_R^E$, $l(A) = l_R(A) \geq l_R(B_i\theta\theta_{i-1}) = l(B_i\theta\theta_{i-1})$. In case $Rel(A) \simeq Rel(B_i) \in NTab_R$ and $C_2(Rel(A), Rel(B_i))$ doesn't hold, $l(A) = l_R(A) > l_R(B_i\theta\theta_{i-1}) = l(B_i\theta\theta_{i-1})$.

- A is defined in P , $l(A) = l_P(A) + \|A\|_P$.
 - $B_i \in B_R^E$, $l(B_i\theta\theta_{i-1}) = l_R(B_i\theta\theta_{i-1})$.
 Because of condition 3 in the proposition statement, $\|A\|_P \geq l_R(B_i\theta\theta_{i-1})$.
 Hence, $l(A) = l_P(A) + \|A\|_P \geq l_R(B_i\theta\theta_{i-1}) = l(B_i\theta\theta_{i-1})$.
 Note that in this case we always have that $Rel(A) \not\subseteq Rel(B_i)$ (because P extends R).
 - $B_i \in B_P^E \setminus B_R^E$, $l(B_i\theta\theta_{i-1}) = l_P(B_i\theta\theta_{i-1}) + \|B_i\theta\theta_{i-1}\|_P$.
 Because of condition 2 in the proposition statement, $l_P(A) \geq l_P(B_i\theta\theta_{i-1})$. And
 because of condition 3, $\|A\|_P \geq \|B_i\theta\theta_{i-1}\|_P$. Hence, $l_P(A) + \|A\|_P \geq l_P(B_i\theta\theta_{i-1}) + \|B_i\theta\theta_{i-1}\|_P$. In case $Rel(A) \simeq Rel(B_i) \in NTab_P$ and $C_2(Rel(A), Rel(B_i))$ doesn't hold, we have that $l_P(A) > l_P(B_i\theta\theta_{i-1})$, hence $l_P(A) + \|A\|_P > l_P(B_i\theta\theta_{i-1}) + \|B_i\theta\theta_{i-1}\|_P$.

In each case, we conclude that $l(A) \geq l(B_i\theta\theta_{i-1})$ and that, in case $Rel(A) \simeq Rel(B_i) \in NTab_{P \cup R}$ and $C_2(Rel(A), Rel(B_i))$ doesn't hold, $l(A) > l(B_i\theta\theta_{i-1})$. \square

The next proposition considers the special case in which two programs P_1 and P_2 extend each other (this is the same case as considered in Proposition 4.3).

Proposition A.2 *Let P_1, P_2 be programs such that P_1 extends P_2 and P_2 extends P_1 . Let $S \subseteq B_{P_1 \cup P_2}^E$. Suppose that*

1. P_1 is quasi-acceptable w.r.t. Tab_{P_1} , $S \cap B_{P_1}^E$ and a level mapping l_1 on $B_{P_1}^E$ which is finitely partitioning on $Call(P_1, S \cap B_{P_1}^E) \cap B_{Tab_{P_1}}^E$,
2. P_2 is quasi-acceptable w.r.t. Tab_{P_2} , $S \cap B_{P_2}^E$ and a level mapping l_2 on $B_{P_2}^E$ which is finitely partitioning on $Call(P_2, S \cap B_{P_2}^E) \cap B_{Tab_{P_2}}^E$,

then $m(l_1, l_2)$ (see Lemma A.1) is a finitely partitioning level mapping on $Call(P_1 \cup P_2, S) \cap B_{Tab_{P_1 \cup P_2}}^E$, and $P_1 \cup P_2$ is quasi-acceptable w.r.t. $Tab_{P_1 \cup P_2}$, S and $m(l_1, l_2)$. Hence, $P_1 \cup P_2$ quasi-terminates w.r.t. $Tab_{P_1 \cup P_2}$ and S .

Proof Note that, because P_1 extends P_2 and vice versa, $Call(P_1 \cup P_2, S) \cap B_{P_i}^E = Call(P_i, S \cap B_{P_i}^E)$, $i \in \{1, 2\}$. By Lemma A.1, $m(l_1, l_2)$ is finitely partitioning on $Call(P_1 \cup P_2, S) \cap B_{Tab_{P_1 \cup P_2}}^E$. Also, if $H \leftarrow B_1, \dots, B_n$ is a clause in P_i and $mgu(A, H) = \theta$, then an LD-c.a.s. in P_i for $\leftarrow (B_1, \dots, B_{i-1})\theta$ is an LD-c.a.s. in $P_i \cup P_j$ ($\{i, j\} = \{1, 2\}$) for $\leftarrow (B_1, \dots, B_{i-1})\theta$ and vice versa. Then it directly follows that $P_1 \cup P_2$ is quasi-acceptable w.r.t. $Tab_{P_1 \cup P_2}$, S and $m(l_1, l_2)$. \square

Modular proofs for the LG-termination of $P \cup R$, which construct an appropriate level mapping such that $P \cup R$ is LG-acceptable w.r.t. this level mapping, can be found in the same way as in the above Propositions A.1 and A.2 for quasi-termination. We omit the details here.