Completeness of Dynamic Time-Bounded Derivation for Locally Weak Reducing Programs

Shi, Yi-Hua
Department of Information Systems, Kyushu University

Arikawa, Setsuo
Research Institute of Fundamental Information Science Kyushu University

https://doi.org/10.5109/3138

出版情報：Bulletin of informatics and cybernetics. 24 (3/4), pp.93-109, 1990-08-22. Research Association of Statistical Sciences
バージョン：
権利関係：

Bulletin of Informatics and Cybernetics, Vol. 24, No. 3~4, 1991

COMPLETENESS OF DYNAMIC TIME-BOUNDED DERIVATION FOR LOCALLY WEAK REDUCING PROGRAMS

By

Yihua SHI* and Setsuo ARIKAWA**

Abstract

In the logic programming, it is generally undecidable whether or not a formula is deducible from a program. In this paper, we discuss the relations between structure of programs and termination of derivations, and show that the predicate symbols in a program can be classified and stratified by a relation between them. We propose two classes of logic programs called locally weak reducing and locally reducing programs. We also give a new dynamic time-bounded derivation procedure, and prove the completeness of this procedure for locally weak reducing programs.

1. Introduction

The termination problem of derivations in logic programming is generally unsolvable. There are, however, some terminating or terminable subclasses of logic programs such as hierarchical programs, and function-free logic programs. Unfortunately, they have not enough power to express many practical programs. To enlarge the terminable classes, some new classes such as reducing programs and weakly reducing programs have been proposed [2].

This paper discusses the relations between clauses, programs and the termination problem. Then, we propose two larger classes than that of weakly reducing programs which are terminable and easy to write.

In Section 3, we consider the structure of programs, the finite termination and the relation between them. We show that the predicate symbols of a program can be classified and stratified in a natural way, and point out that a program is terminable if its special parts satisfy some syntactic conditions. Using these properties, in Section 4, we define two new classes called locally weak reducing programs and locally reducing programs which have less syntactic restrictions and are easy to write. In Section 5, we discuss the termination problem for locally weak reducing programs. For this purpose, we provide a dynamic time-bounded derivation procedure, and prove its completeness

* Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University 39, Kasuga 816, Japan
E-mail: shi@rifis.sci.kyushu-u.ac.jp

** Research Institute of Fundamental Information Science, Kyushu University 33, Fukuoka 812, Japan

for locally reducing and locally weak reducing programs. In Section 6, we compare our results with the previous works. Then, we propose a use of the abstract base to simplify the operations on programs and to make the derivations efficient. Furthermore, we describe the relations between the dynamic time-bounded derivation procedure and the time-bounded reasoning system [10].

2. Basic Definitions

We first define some notions for our discussions. The other basic terminology and concepts not defined in this paper may be found in [6].

A *goal* is a clause of the form "$\leftarrow L_1, L_2, \ldots, L_n$" ($n \geq 1$). A *program clause* is a clause of the form "$A \leftarrow L_1, L_2, \ldots, L_n$" ($n \geq 0$). A program clause is called a *fact* if it is of the form "$A \leftarrow$", otherwise called a *rule*, where A is an atom and L_i ($i = 1, 2, \ldots, n$) is a literal. A *program* is a finite set of program clauses. A program clause or goal is *ground* if it does not contain any variable symbol. We use the program clause or goal is *ground* if it does not contain any variable symbol. We use the following notations:

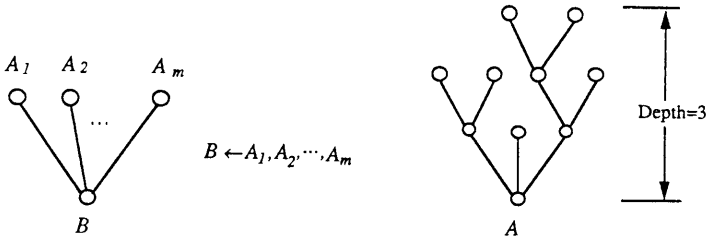
pred(L) : the predicate symbol of the literal L.

$B(P)$: the Herbrand base of program P.

$PS(P)$: the set of predicate symbols in program P.

$\#(S)$: the *size* of the set S, i.e., the number of elements in S.

We also use the concepts of *breadth-first derivation* (BF-derivation for short), *derivation tree*, and the *depth of derivation tree* as in Fig. 1, where $B \leftarrow A_1, \ldots, A_m$ is a clause, and A is an atom.



(a) BF-derivation (b) derivation tree for A

Fig. 1. BF-derivation and BF derivation tree

The derivation tree of Fig. 1(b) is constructd by BF-derivations like Fig. 1(a). We denote the *depth* of the derivation tree by the length of its longest branch, e.g., the depth of derivation tree for A is 3 as shown in Fig. 1(b).

To describe the structure of programs, we need the following concepts.

DEFINITION 1. Let P be a program. A binary relation *Rel* on $PS(P)$ is defined as follows:

$$Rel = \{(p,q_i) | A \leftarrow L_1, \ldots, L_k \in P,\ p = pred(A),\ q_i = pred(L_i),\ 1 \leq i \leq k\}$$

DEFINITION 2. Let p be a predicate symbol in program P.

$$IS(p) = \{q | (p, q) \in Rel^+\}$$

is the set of the predicate symbols called *an influence set of the predicate symbol* p, where Rel^+ is the transitive closure of Rel.

For providing some syntactic restrictions for program clauses, we introduce the following notions:

$o(x, A)$: the number of all occurrences of a variable x in an atom A.

$var(A)$: the set of all variables in an expression A.

An *expression* is either a term, a literal, a conjunction or disjunction of literals. The *size* of an expression e, denoted by $|e|$, is the total number of occurrences of variable symbols, constant symbols, function symbols and predicate symbols in e. Note that $|\neg A| = |A|$ for negative literal $\neg A$.

Now, we give an example to explain these concepts.

EXAMPLE 1. Let P_1 be a program below:

$$P_1 = \left\{ \begin{array}{l} p_0(x, y, z) \leftarrow \neg p_1(x, f^4(y)). \\ p_1(x, f(y)) \leftarrow p(a, x), p_2(x, y). \\ p_2(\ldots) \leftarrow p_1(\ldots), p_3(\ldots). \\ p(\ldots) \leftarrow q(\ldots), t(\ldots). \\ q(\ldots) \leftarrow r(\ldots), s(\ldots), u(\ldots). \\ r(\ldots) \leftarrow p(\ldots), u(\ldots). \\ s(\ldots) \leftarrow p_3(\ldots), \neg t(\ldots). \end{array} \right\}$$

where x, y, z are variable symbols, a is a constant symbol, f is a function symbol, and f^4 denotes a four-fold application of f. Then,

$$pred(p_0(x, y, z)) = p_0, \quad pred(p_1(x, f^4(y))) = p_1, \quad pred(p_3(\ldots)) = p_3,$$

$$PS(P_1) = \{p_0, p_1, p_2, p_3, p, q, r, s, t, u\},$$

$$Rel = \left\{ \begin{array}{l} (p_0, p_1), (p_1, p), (p_1, p_2), (p_2, p_1), (p_2, p_3), (p, q), (p, t), \\ (q, r), (q, s), (q, u), (r, p), (r, u), (s, p_3), (s, t) \end{array} \right\},$$

$$IS(p_0) = \{p_1, p_2, p_3, p, q, r, s, t, u\},$$

$$IS(p) = \{p_3, p, q, r, s, t, u\},$$

$$IS(s) = \{p_3, t\},$$

$$o(z, p_1(x, f^4(y))) = 0, \quad o(x, p_1(x, f^4(y))) = 1, \quad o(y, p_1(x, f^4(y))) = 1,$$

$$var(p_0(x, y, z)) = \{x, y, z\}, \quad var(p_1(x, f^4(y))) = \{x, y\}, \quad var(f^4(y)) = \{y\},$$

$$|x| = 1, \quad |a| = 1, \quad |f^4(x)| = 5, \quad |\neg p(x, f^4(y))| = 7. \qquad \square$$

3. Program Clauses, Programs and Termination

As described in the introduction, there are some terminating programs, but their power of expression is not strong. So we need some more powerful classes of logic programs. However, there seems to be a tradeoff between the expressive power and the finite termination in logic programs. We consider this tradeoff from the viewpoint of relations between termination and structure of programs.

The many researchers [10, 12, 2] have discussed whether and when a derivation procedure can be terminated based on the sizes of Herbrand universe and base. The search space will explosively grow with the increment of Herbrand base. In general, its size may be $O(2^{\#B(P)})$. Thus, it is also necessary to discuss the relations between the search space and the structure of programs. We need the following concepts.

DEFINITION 3. *A level mapping* of a program P, denoted by δ, is a mapping from $PS(P)$ to positive integers which satisfies the following conditions for any $A \leftarrow L_1, \ldots, L_m \in P$ and any $i(1 \leq i \leq m)$:

(1) $\delta(pred(A)) > \delta(pred(L_i))$ if $pred(A) \notin IS(predL_i))$,

(2) $\delta)pred(A)) = \delta(pred(L_i))$ *if* $pred(\mathrm{A}) \in IS(pred(L_i))$.

We refer to $\delta(p)$ as the level of the predicate symbol p.

DEFINITION 4. Let P be a program.

$$R_1(P) = \{p \mid \mathrm{p} \in IS(p) \text{ and } p \in PS(P)\},$$

$$R_2(P) = \{p \mid (p, q) \in Rel^+, q \in IS(q), p \notin R_1(P) \text{ and } p \in PS(P)\},$$

$$S(P) = \{p \mid p \notin R_1(P) \cup R_2(P) \text{ and } p \in PS(P)\}.$$

A predicate symbol is called *simple, directly recursive, or indirectly recursive* if it is in $S(P)$, $R_1(P)$, or $R_2(P)$, respectively.

Form Definition 3, trivially we have the following.

PROPOSITION 1. For any program, there is a level mapping which satisfies the conditions (1) and (2) in Definition 3.

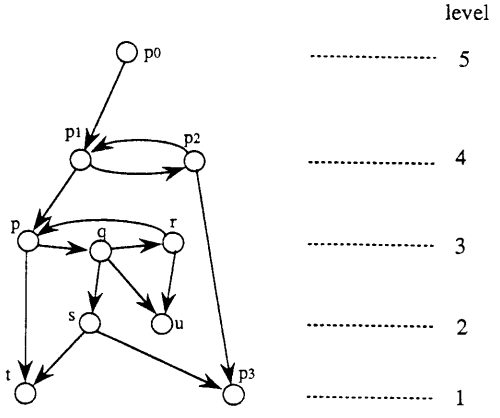


Fig. 2. The stratums of program P_1 with level mapping

EXAMPLE 2. Consider the program P_1 in Example 1. The predicate symbols in P_1 are stratified by Definition 3 as shown in Fig. 2. By Definition 4, $PS(P_1)$ is classified into three subsets:

$$S(P_1) = \{p_3, s, t, u\},$$
$$R_1(P_1) = \{p_1, p_2, p, q, r\},$$
$$R_2(P_1) = \{p_0\}. \quad \square$$

From the example above, the followings are observed: (1) The levels of predicate symbols, which can reach each other, are the same, e.g., $\delta(p_1) = \delta(p_2) = 4$, and $\delta(p) = \delta(q) = \delta(r) = 3$. (2) The recursive calls can not occur between predicate symbols with different levels. (3) The derivation for a goal may use only the predicate symbols whose levels are less than or equals to this goal's. For example, only p, q, r, s, u, t are used to the goal $\leftarrow p(\ldots)$, and only s, t, p_3 are used to $\leftarrow s(\ldots)$. (4) If $A_0 \rightarrow A_1 \rightarrow \ldots \rightarrow A_n \ldots$ is a branch(path) of a BF-derivation tree, then $\delta(p_0) \geq \delta(p_1) \geq \ldots \geq \delta(p_n) \ldots$ and $\delta(p_i) = \delta(p_{i+1})$ only when p_i, p_{i+1} are directly recursive, where $p_i = pred(A_i)$ $(i = 1,\ldots,n,\ldots)$.

The level mapping shows some strata of predicate symbols in a program, and such a stratum structure shows some relations among predicate symbols in a program. We can also see that *the derivation for* $P \cup \{\leftarrow L\}$ *uses only a part of program clauses and predicate symbols in the program* P, that is, its influence set $IS(pred(L))$.

From Definition 4, $S(P)$, $R_1(P)$ and $R_2(P)$ are mutually disjoint and

$$S(P) \cup R_1(P) \cup R_2(P) = PS(P).$$

A special attention should be paid to $R_1(P)$, that is, to the recursive calls among predicate symbols, since if there are no recursive calls in the program P, that is, $R_1(P) = \emptyset$, then the P is hierarchical and any derivation in such a program will terminate.

PROPOSITION 2. Let P be a program, and $\leftarrow L$ be a goal. The derivation for $P \cup \{\leftarrow L\}$ terminates in a finite time if $pred(L) \in S(P)$.

PROOF. It is trivial, since one step of derivation for a simple predicate symbol decreases the level of each subgoal by at least one, and the level of any predicate symbol is finite. $\square$

In this sense, the recursive call is a main cause that leads to the undecidability of the derivation procedure. Therefore, only by restricting the recursive calls of directly recursive predicate symbols to finite times, the termination of derivations is guaranteed. According to such a stratification structure of programs and the classification of predicates, the search space for deriving a goal can be reduced radically. Hence we can only pay attention to some small parts of a program, that is, to the directly recursive predicate symbols.

4. Locally Weak Reducing Programs

Along the line discussed above, we introduce two new classes called locally weak

reducing programs and locally reducing programs.

DEFINITION 5. A rule $A \leftarrow L_1, \ldots, L_n$ in a program P is *locally weak reducing* (*local reducing*) if for any substitution θ and any $i = 1, \ldots, n$ the following conditions (1) and (2) hold:

(1) $|A\theta| \geq (>) |L_i\theta|$ if $\delta(pred(A)) = \delta(pred(L_i))$ and L_i is a positive literal,

(2) $|A\theta| > |L_i\theta|$ if $\delta(pred(A)) = \delta(pred(L_i))$ and L_i is a negative literal,

where δ is a level mapping of P.

DEFINITION 6. A program P is *locally weak reducing* if every rule in P is locally weak reducing. A program P is *locally reducing* if every rule in P is locally reducing.

Now we give the syntactic conditions that checks whether or not a program is locally weak reducing (locally reducing).

PROPOSITION 3. Let P be a program which contains at least one function symbol and δ be a level mapping of P. A rule $A \leftarrow L_1, \ldots, L_n$ in P is locally weak reducing (locally reducing) if the following conditions (a), (b) and (c) hold for any variable x in this rule and any i in $\delta(pred(A)) = \delta(pred(L_i))$ $(1 \leq i \leq n)$.

(a) $o(x, A) \geq o(x, L_i)$.

(b) $|A| \geq (>) |L_i|$, if L_i is a positive literal.

(c) $|A| > |L_i|$, if L_i is a negative literal.

PROOF. A similar proof can be found in [2]. □

We can see that the conditions (b) and (c) are independent of the presence of function symbols. Therefore, also in case P contains no function symbols, we can similarly show that a rule $A \leftarrow L_1, \ldots, L_n$ in P is locally weak reducing of the above conditions (b) and (c) hold for any i in $\delta(pred(A)) = \delta(pred(L_i))$, $(1 \leq i \leq n)$.

EXAMPLE 3. The programs

$$P_2 = \begin{Bmatrix} p(x, y, z) \leftarrow s(f_4(w), y), p(y, z, x). \\ s(x, f(x)) \leftarrow . \end{Bmatrix},$$

$$P_3 = \begin{Bmatrix} p(x, y, z) \leftarrow s(x, y), p(y, z, x). \\ s(x, y) \leftarrow p_1(x, y, z). \\ p_1(x, y, z) \leftarrow p_1(x, y, z). \end{Bmatrix}$$

are locally weak reducing. The derivation for $P_3 \cup \{\leftarrow p(a, b, c)\}$ is obviously infinite. □

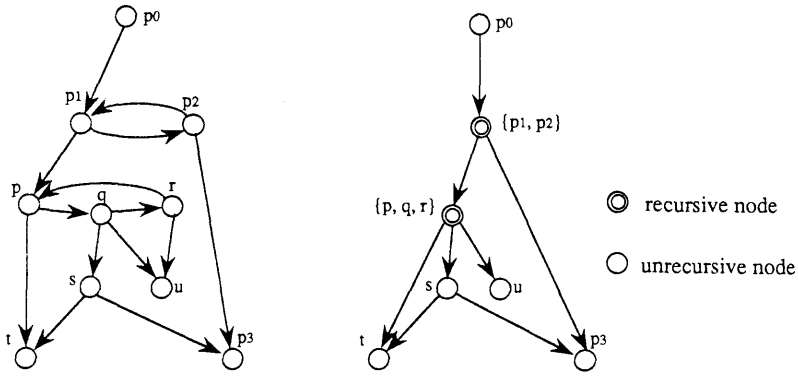
5. Termination of Locally Weak Reducing Programs

Example 3 above shows that the derivations may be infinite even for locally weak reducing programs. In order to discuss the termination problem for locally weak reducing programs, we introduce a time-bound into the derivation procedure. The time-bounded derivation will terminate the derivation within finite steps. The number

of such steps is called a time-function of the time-bounded derivation.

For the discussions in this section, we recall the concept of abstract graph [4] as shown in Fig. 3.

For rule $A \leftarrow L_1, L_2, \ldots, L_n$ in P, we put arcs from p to q_i ($i = 1, 2, \ldots, n$) into the graph, where $p = pred(A)$ and $q_i = pred(L_i)$. Then, we identify the strongly connected components (SCCs) of the graph, which are the maximal sets of nodes that can reach each other. The *abstract graph* of P is obtained by combining all nodes that are in $R_1(P)$ and also in the same SCC into a *recursive node* labeled a set of nodes of this SCC, and eliminating arcs among them. Hence the abstract graph is acyclic. The *abstract proof graph* of a node r is the maximal subgraph which can be reached from r. We refer to a node as a leaf, if no nodes can be reached from it.



(a) the stratification of $PS(P)$ (b) the abstract graph of P

Fig. 3. The level mapping and the abstract graph

PROPOSITION 4. (Knuth[5]) Let $T(n)$ be the number of all ordered trees each of which has n nodes. Then,

$$T(n) = \frac{1}{n}\binom{2(n-1)}{n-1}$$
$$= O(4^n \cdot n^{-\frac{3}{2}}).$$

Now we can regard an atom as an ordered tree whose nodes are labeled by the predicate symbols, function symbols, variable symbols, and constant symbols. Thus, the number of atoms whose sizes are less than or equal to n is finite.

PROPOSITION 5. Let P be a locally weak reducing program, N_j be a recursive node of abstract graph of P, Π be the set of constant symbols and function symbols, and Σ be the set of predicate symbols that occur in N_j. Then, the total number of atoms whose predicate symbols are in N_j and sizes $\leq n + 1$ is less than or equal to

$$d(n+1) = O(n \cdot \#(\Sigma) \cdot (\#(\Pi) + 1)^{4^n \cdot n^{-\frac{3}{2}}}).$$

PROOF. We can look upon the constant symbols, function symbols, and predicate

symbols as the nodes of ordered trees. By using Proposition 4 and the results in Section 3, we can prove this proposition straightforward. □

In order to use the time-bounded derivation procedure, we need to calculate a time-function for the goal and the program before the derivation starts. In this paper, we estimate the time-function by the sizes of goals which appear in the derivation. However, in the locally weak reducing and weak reducing programs, only the directly recursive predicate symbols of rules satisfy the condition (a), (b) and (c) in Proposition 3, while the simple predicate symbols and the indirectly recursive predicate symbols may not satisfy them. Thus, the new variables may be introduced into the goals such as in Example 4 below. Furthermore, the un-ground goals(queries) are permitted.

EXAMPLE 4. Consider a locally weak reducing program P_3 given in Example 3. In the derivation for $P_3 \cup \{ \leftarrow p(a, b, c)\}$, the goal $\leftarrow p(a, b, c)$ is ground, but with the rule "$s(x, y) \leftarrow p_1(x, y, z)$", a new variable symbol z is introduced as in a subgoal $\leftarrow p_1(a, b, z)$ in Fig. 4. □

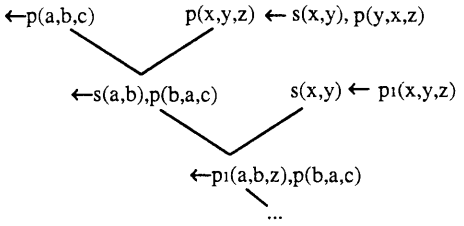


Fig. 4. Introducing a new variable to subgoals

The sizes of goals which contain the variables can be changed by some substitutions during derivations as shown in Example 5.

EXAMPLE 5. Consider a locally weak reducing program P_2 given in Example 3, and consider the derivation for $P_2 \cup \{ \leftarrow p(x, y, z)\}$ as shown in Fig. 5. At the start of derivation, the size of goal $\leftarrow p(x, y, z)$ is 4. The rule $p(x, y, z) \leftarrow s(f^4(w), y), p(y, z, x)$ satisfies $|p(x, y, z)| \leq |p(y, z, x)|$ and $var(p(x, y, z)) = var(p(y, z, x)) = \{x, y, z\}$, but $var(s(f^4(w), y)) \notin var(p(x, y, z))$. Thus, with the substitution $\{x: = f^4(w), y: = g(f^4(w))\}$, the size of the goal $\leftarrow p(g(f^4(w)), x, z)$ becomes 9. □

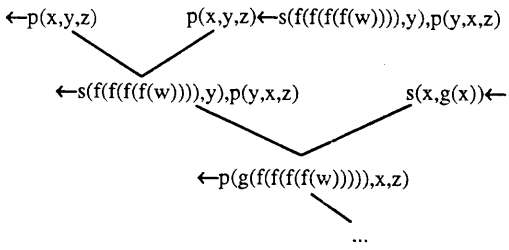


Fig. 5. An increase of the sizes of subgoals

As in the above example, the sizes of goals may increase or decrease during a derivation. Therefore, it is difficult to calculate the time-function exactly before the derivation starts. In order to solve these problems, we need a dynamic time-bounded derivation procedure that can check and modify the time-function while deriving.

Now, we give an algorithm of such derivation procedure. It is constructed by adding a counter to each goal and uses a derivation procedure which also changes and checks its values as we are showing in Fig. 8(b). Although it only gives one solution of the goal, we can easily modify it so as to give all solutions.

Algorithm 1: Dynamic Time-Bounded Derivation Procedure
Input : a goal $\leftarrow G$ and a program P
Output: a solution of the goal

```
  procedure Derivation(P, G)
  begin
    ans1 := DTBD(G, |G|, f(P, |G|));
    if ans1 = 0 then answer :="No"
      else answer := G∘ans1;
    output answer;
  end
  procedure DTBD(G, l, d)
  begin
    counter := d;
    if counter = 0 then return (0);
    if no clause can be unified with G then return(0);
    for every clause whose head can be unified with G do
    /* Let R = A ← B1, ..., Bn and θ = unifier(G, A)*/
        if n = 0 then return(θ) else
        begin
          for i := 1 to n do
            if δ(pred(A)) = δ(pred(Bi)) then
              if l < |Biθ| then
                  begin
(1)                     θ1 := DTBD(Biθ, |Biθ|, f(P, |Biθ|)));
                        θ  := θ∘θ1
                  end
                else
                  begin
(2)                     θ1 := DTBD(Biθ, l, counter − 1);
                        θ  := θ∘θ1
                  end
              else
                begin
(3)                     θ1 := DTBD(Biθ, |Biθ|, f(P,|Biθ|)));
                        θ  := θ∘θ1
```

```
            end;
        return(θ)
    end
end
```

/* f is a time-function, and it is computed by Proposition 5, $\circ$is the composition of unifiers, and assume θ hold $|\theta\circ 0| = |0\circ\theta| = 0$ for any unifier. */

As described in Section 3, roughly a derivation for $P \cup \{\leftarrow L\}$ can always be terminated and gives the solutions after the Herbrand base of program P has been enumerated. If we can show that, for any locally weak reducing program and any goal, the sizes of all goals occurring in its derivation are less than a fixed value, then we can terminate the derivation.

Let t_1 and t_2 be terms which consist of constant symbols, variable symbols, and unary function symbols. Assume $var(t_1) \cap var(t_2) = \emptyset$. A substitution $\{x: = t\}$ is called "form t_1 to t_2" if the variable symbol x is in t_2, and in this case, we put an arc from t_1 to t_2 as in Fig. 6 and Fig. 7. Since $|x| = 1$ and $|t| \geq 1$, if a unifier of t_1 and t_2 is form t_1 to t_2 then $|t_1| \geq |t_2|$. Thus, the direction of arcs is always from larger terms to smaller terms in the case of unifications. By the definition of unification, the following properties hold:

PROPERTY 5.1. For any unifier θ of t_1 and t_2, $|t_1\theta| = |t_2\theta| = max(|t_1|, |t_2|)$.

PROPERTY 5.2. Throughout the derivation, the sizes of goals are unfixed, the sizes of atoms in a rule are fixed, and the unifiers always from larger terms to smaller terms.

PROPOSITION 6. For any goal $\leftarrow G$, the dynamic time-bounded derivation procedure *Derivation*(P, G) terminates in a finite time and gives a solution of G if P is a locally reducing program.

PROOF. The dynamic time-bounded derivation procedure *Derivation*(G, P) calls the recursive sub-procedure *DTBD*(G, l, d), where $\leftarrow G$ is a goal, l is the current bound of sizes, and d is a counter. If d is 0, then this sub-procedure terminates and returns 0 for the goal.

Let $G_0, G_1, \ldots$ be any path of BF-derivation tree for $P \cup \{\leftarrow G_0\}$ and δ be a level mapping. Consider the derivation for G_i. The counter is modified only in Lines (1), (2) and (3). If $\delta(G_i) > \delta(G_{i+1})$, then the *DTBD* in Line (3) is called and d is calculated again. Since the level of the goal G_i is finite and Line (3) results in a decrease of the level of the goal by at least one, Line (3) can be called only finite times. If $\delta(G_i) = \delta(G_{i+1})$ and $l \geq |G_{i+1}|$, then the *DTBD* in Line (2) is called and counter d is decreased by 1. Since $\delta(G_i) = \delta(G_{i+1})$ means that G_i and G_{i+1} may form a loop as described before, the counter d is used to detect such an infinite loop by Proposition 5. If $\delta(G_i) = \delta(G_{i+1})$ and $l < |G_{i+1}|$, then the *DTBD* in Line (1) is called and the counter d is increased by Proposition 5. Therefore, if we show the Line (1) is called only finite times, that is, $|G_i|$ is finite, then we can prove this proposition.

Let the above path be abstractively corresponded to $A_1, A_2, \ldots, A_m$ by the definition of abstract tree. If any A_i is not a recursive node, then the path terminates in a finite time. Now, by using an induction on the number of recursive nodes, we prove that any path of BF-derivation for $P \cup \{\leftarrow G\}$ terminates in a finite time, and does not

affect the correctness of the results.

For simplifying the proof, we assume only unary function symbols are in P.

1. (Base step)

Assume there is only one recursive node in the path. Then, assume also $\delta(G_0) > \delta(G_1) > \ldots > \delta(G_{n_1-1}) > \delta(G_{n_1}) = \delta(G_{n_1+1}) = \ldots = \delta(G_{n_2}) > \delta(G_{n_2+1}) > \ldots$. Since the $\delta(G_0)$ is finite and $\delta(G_i) > 0 (i \geq 1)$, the path terminates in a finite time if the length $n_2 - n_1$ of the sub-path G_n, G_{n_1+1}, ..., G_{n_2} is finite.

With Proposition 5, if there exists a number k which bounds the sizes of all subgoals in G_{n_1}, G_{n_1+1}, ..., G_{n_2}, then $n_2 - n_1$ is finite.

We classify the cases by the independency § of terms and the unification of them. Without losing the generality, we may assume G_{n_1} is $\leftarrow p(t_1, \ldots, t_k)$ and

$$p(s_1, \ldots, s_k) \leftarrow \ldots, p(r_1, \ldots, r_k), \ldots$$

is a rule in P. Consider a unification of $p(t_1, \ldots, t_k)$ and $p(s_1, \ldots, s_k)$.

Case (i): Both $p(t_1, \ldots, t_k)$ *and* $p(s_1, \ldots, s_k)$ are independent.

In this case $var(s_i) \cap var(s_j) = \phi$ and $var(t_i) \cap var(t_j) = \phi$ for $i, j (1 \leq i, j \leq k, i \neq j)$. Assume θ_1 is a unifier of $p(t_1, \ldots, t_k)$ and $p(s_1, \ldots, s_k)$. Then by Property 5.1 and Property 5.2, we have

$$\sum_{i=1}^{k} max(|t_i|,|s_i|) + 1 = |p(s_1,\ldots,s_k)\theta_1| \geq |p(r_1,\ldots,r_k)\theta_1|.$$

Let $p(t'_1,\ldots,t'_k) = p(r_1,\ldots,r_k)\theta_1$ and θ_2 be a unifier of $p(t'_1,\ldots,r_k)$ and $p(s_1,\ldots,s_k)$. Then,

$$\sum_{i=1}^{k} max(|t_i|,|s_i|) + 1 \geq \sum_{i=1}^{k} max(|t'_i|,|s_i|) + 1 \geq |p(s_1, \ldots, s_k)\theta_2|$$
$$\geq |p(r_1, \ldots, r_k)\theta_2| \geq \ldots.$$

Thus, the number $\sum_{i=1}^{k} max(|t_i|,|s_i|)+1$ bounds the sizes of all goals G_{n_1}, G_{n_1+1}, $\ldots, G_{n_2}$.

Case (ii): Either $p(t_1, \ldots, t_k)$ or $p(s_1 \ldots s_k)$ is independent.

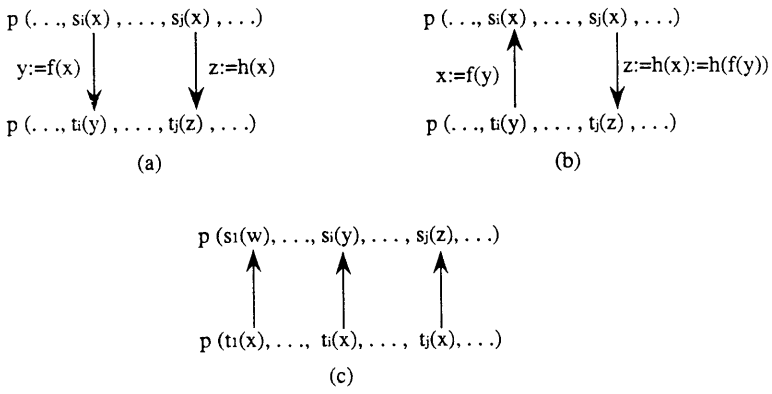


Fig. 6. Three kinds of substitutions

§ Let t_1 and t_2 be two terms. Then they are independent if $var(t_1) \cap var(t_2) = \phi$. A predicate $p(t_1, \ldots, t_k)$ is independent if t_i and t_j are independent for all $i \neq j$.

Assume $p(s_1, \ldots, s_k)$ is not independent and $var(s_i) \cap var(s_j) = \{x\}$. Then there are three kinds of substitutions such as (a), (b) and (c) in Fig. 6 ¶.

(a) If $\theta = \{y: = f(x), z: = h(x)\}$, then by Properties 5.1, 5.2, we have

$$\sum_{i=1}^{k} max(|t_i|,|s_i|) + 1 = |p(s_1, \ldots, s_k)\theta_1| \geq |p(r_1, \ldots, r_k)\theta_1|.$$

As discussed in the case (i), all the sizes of goals created by repeated uses of such substitutions as Fig. 6(a) are bounded.

(b) If $\theta = \{x: = f(y), z: = h(x)\}$, then

$$\sum_{i=1}^{k} max(|t_i|,|s_i|) + 1 \leq |p(s_1, \ldots, s_k)\theta_1|,$$

since $|x| \leq |f(y)|$ and $|z| \leq |h(x)| \leq h(f(y))|$. That is, the unifier may increase the sizes of goals. However, at the same time, the number of variables decreases by 1 as shown in Fig. 6(b).

(c) Furthermore, since $G_{n_1}, \ldots, G_{n_2}$ are directly recursive atoms, the numbers of variable symbols in them do not increase. According to Property 5.2, when the sizes of goals increase to a finite value, the substitutions such as (b) can not be applied and only the substitutions such as Fig. 6(c) can be done, since the arc is always from larger terms to smaller ones. Thus, the sizes of goals do not increase.

Case (iii): Neither $p(t_1, \ldots, t_k)$ nor $p(s_1, \ldots, s_k)$ is not independent.

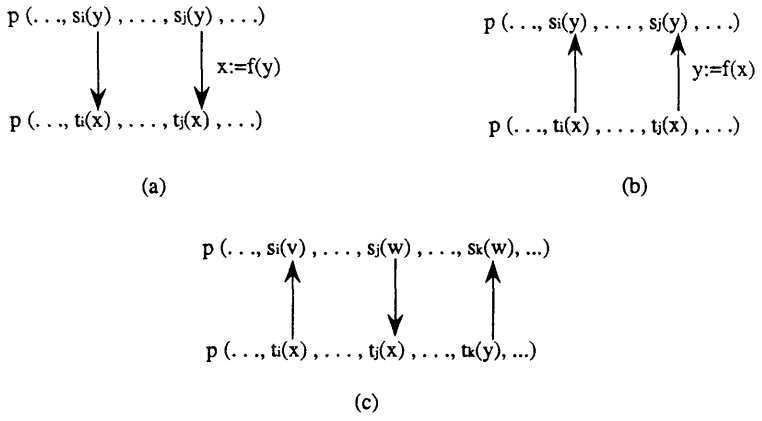


Fig. 7. Three kinds of substitutions

As shown in Fig. 7, they look like combinations of cases (i) and (ii). Thus, by a similar discussion to cases (i) and (ii), we can prove that the sizes of all subgoals G_{n_1}, $G_{n_1+1}, \ldots, G_{n_2}$ are bounded by a finite value.

2. (Induction step)

Assume the length of any path which contains m recursive node $N_i (i \leq m)$ is finite. Now we prove that the path which contains $m + 1$ recursive nodes is finite.

The lengths of sub-paths in which levels of predicate symbols are lower than the

¶ In the figures, $t_i(x)$, $s_i(x)$ means the term t_i, s_i containing the variable x, respectively.

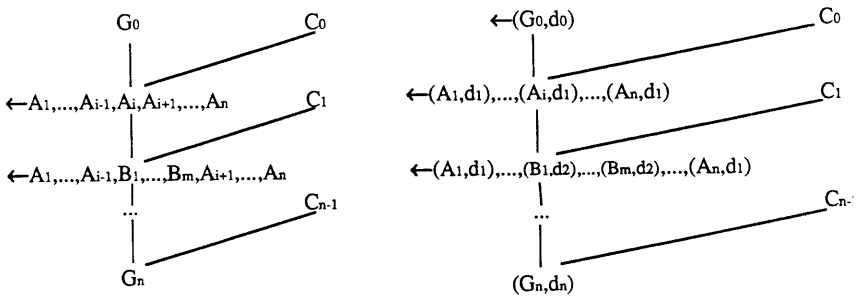
level of the recursive node N_{m+1} are finite, by the assumption of induction. Then, by the same discussions as in (1), the sub-path caused by the recursive calls of the predicate symbols in N_{m+1} is finite. Therefore, the length of any path which has $m + 1$ recursive nodes is finite.

Finally, note that the above proof on unary function symbols is still valid in the general case of the proposition, since a size of the term which contains n-ary function symbols can be treated as an n-ary predicate symbols such as $p(t_1, t_2, \ldots, t_n)$ in the above. □

THEOREM 1. The dynamic time-bounded derivation procedure is sound and complete for the locally reducing and locally weak reducing programs.

PROOF. The soundness is trivial, since the dynamic time-bounded derivation procedure is constructed by adding functions for size check of subgoals, calculation of time-function and counting to the SLDNF-derivation procedure as shown in Fig. 8.

The completeness can be proved immediately by Proposition 6. □



(a) The SLDNF derivation procedure (b) The dynamic time-bounded derivation procedure

Fig. 8. SLDNF and *DTBD*

EXAMPLE 6. A program

$$P_4 = \left\{ \begin{array}{l} q(x) \leftarrow p(f(x), y, z). \\ p(x, f(x), y) \leftarrow p(x, y, x). \\ p(f(a), f(a), f(a)) \leftarrow . \end{array} \right\}$$

is locally weak reducing. Fig. 9 gives a successful derivation tree for $P_4 \cup \{ \leftarrow q(t_1)\}$ which includes variable symbols, and shows the number of variable symbols decreases while the size of goals increases.

If a new rule "$p(x, y,z) \leftarrow p(x, y, z)$" is added into P_4, obviously the dynamic time-bounded derivation procedure returns the solutions of the goal in a finite time, since this rule will not change the sizes of subgoals. □

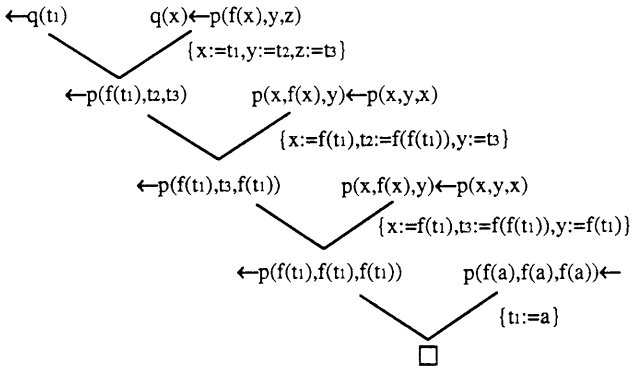


Fig. 9. A dynamic time-bounded derivation (1)

EXAMPLE 7. A program

$$P_5 = \left\{ \begin{array}{l} p(x, y, z) \leftarrow s(x, y), p(y, z, x). \\ s(x, f(x)) \leftarrow . \end{array} \right\}$$

is locally weak reducing. A goal $\leftarrow p(t_1, t_2, t_3)$ with variables t_1, t_2, t_3 produces a fail derivation tree as in Fig. 10. Hence the dynamic time-bounded derivation procedure returns the answer "No" in a finite time. □

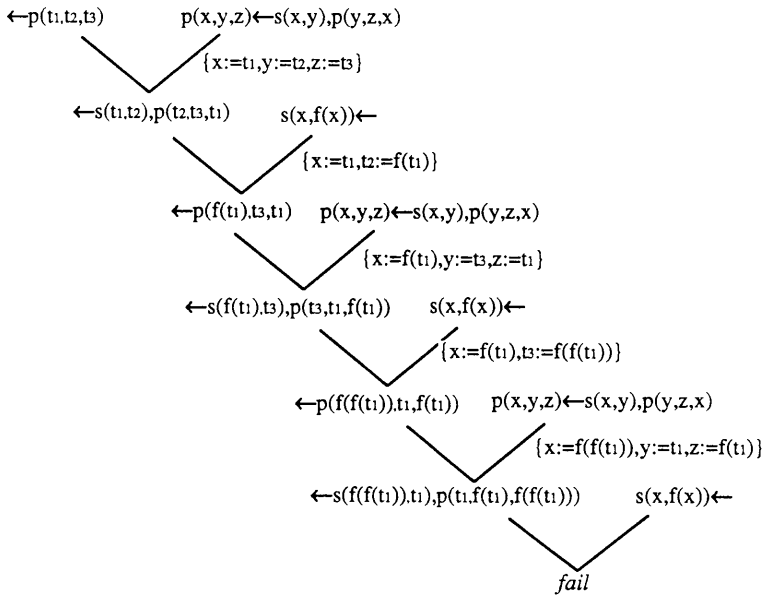


Fig. 10. A dynamic time-bounded derivation (2)

6. Related Works

6.1. Weakly reducing programs

Arimura [2] has proposed the classes of reducing and weakly reducing programs, and proved that the depth-bounded derivations is complete under the prefect model semantics[Przymusinski [7]] for the weakly reducing programs. In the reducing and weakly reducing programs, however, all rules and all parts of a rule must satisfy the condition of reducing or weakly reducing, and also the goals (queries) must be ground. These restrictions should be hard for practical applications.

We expect them to easily express our knowledge and to answer the general queries with variables as well as to the "yes"/"no" (ground) queries. The locally weak reducing programs and the dynamic time-bounded derivation procedure meet our expectations, since they only put such restrictions on some predicate symbols and some small parts of programs, but not to the whole, and permit queries with variables. Trivially, every weakly reducing (reducing) program is locally weakly reducing (locally reducing).

For a locally weak reducing program P and a goal $\leftarrow G$, although the time-function $f(P, G)$ is not calculated before derivation starts, the dynamic time-bounded derivation procedure can dynamically calculate the values of the time function using the sizes of subgoals. The weakly reducing programs have a good property that they keep all subgoals ground and reduce sizes with derivations. Furthermore, the dynamic time-bounded derivation procedure is efficient for the weakly reducing programs such as in P_6 below.

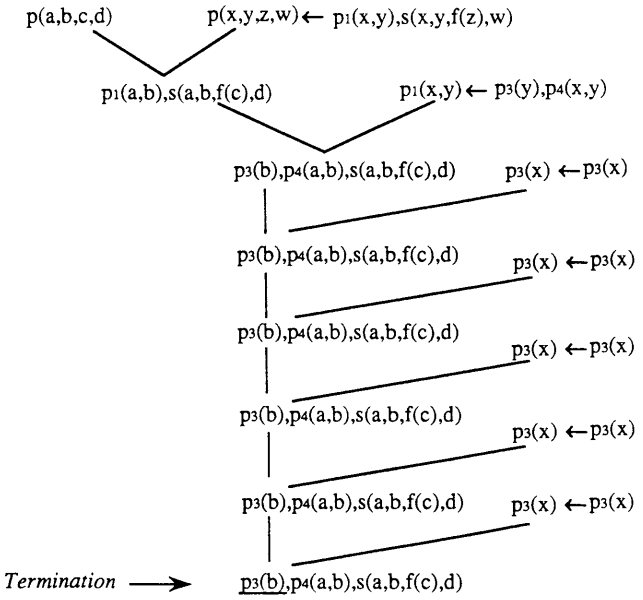


Fig. 11. Termination of the dynamic time-bounded derivation

EXAMPLE 8. Consider a weakly reducing program

$$P_6 = \left\{ \begin{array}{l} p(x, y, z, w) \leftarrow p_1(x, y), p_2(x, y, z, w). \\ p_1(x, y) \leftarrow p_3(y), p_4(x, y). \\ p_3(x) \leftarrow p_3(x). \end{array} \right\}.$$

To a goal $\leftarrow p(a, b, c, d)$, the time-bound[2] of its derivation is

$$O(5 \cdot (4 + 1)^{4^{5 \cdot 5^{-\frac{3}{2}}}}) > O(5^{4^3}) = 5^{64}.$$

With the dynamic time-bounded derivation procedure, however, it can be terminated within $1 + 1 + 1 + N$ steps, where N is the number of terms whose sizes are less than or equal to 1, so $N = 6$. Thus, *DTBD* terminates within 9 steps as shown in Fig. 11.

This example shows the dynamic time-bounded derivation procedure is more efficient than "static" time-bounded derivation procedure.

6.2. Locally weak reducing program and TBR

In this section, we discuss relations between the time-bounded reasoning [9, 10] and the dynamic time-bounded derivation procedure for the locally weak reducing programs. Since the structures of first order predicate programs are complicated, we have introduced abstract base to simplify them.

DEFINITION 7. (Shi [10]) Let P be a program. An *abstract base* AB is a set of atoms

$$AB = \{ab(id, p, n, S_1, S_2, S_3, S_4) | p \text{ is an n-ary predicate symbol in } P\},$$

which expresses the relations among the predicate symbols in P, where

id is a natural number called an identifier of the predicate symbol p,
S_1 is the set of consequences of p,
S_2 is the set of antecedents of p,
S_3 is the set of foundations of p, and
S_4 is the influence set of p.

With the abstract base, we can simplify the structure of a first order program to a propositional program. Trivially,

$$p(\ldots) \text{ is related to } q(\ldots) \text{ in } P \quad \Rightarrow p \text{ is related to } q \text{ in } AB.$$

Since the calculations and operations on AB are much simpler than those on the original P, we use it to record the abstract structure of the program P [9, 10].

Furthermore, the value of time-function may be very large for many goals as seen in Proposition 5. In order to terminate the derivation in a finite time users can wait for, the dynamic time-bounded procedure should be merged into the TBR system. Using the AB, the procedure can be smoothly added to our TBR system[9, 10]. In these senses, the locally weak reducing program and the dynamic time-bounded derivation procedure are important in our time-bounded reasoning.

7. Conclusions

This paper has proposed two classes of locally reducing and locally weak reducing programs. We have paid attention to the recursive predicates, function symbols, variable symbols and the relations among them, and shown several conditions that may cause the infinite loops. Furthermore, using them, we have presented a dynamic time-bounded derivation and proved its completeness for locally weakly reducing (locally reducing) programs. Finally, we have compared our results with related ones, and shown that the dynamic time-bounded derivation is efficient to process ground queries in the weakly reducing programs. On the other words, by the classification and stratification of predicate symbols in programs, we can select a more accurate value of time bound. This is useful to raise the efficiency of the derivation systems.

By the discussion about the functions, we have seen how the sizes of terms are changed by substitutions of variables and functions. The discussion should be also valid in understanding the relations between the function symbols and the stratum of programs.

References

[1] ARIKAWA, S., SHINOHARA, T. and YAMAMOTO, A.: *Elementary formal system as a unifying framework for language learning*, Proc. of the Second Annual Workshop on Computational Learning Theory. Morgan Kaufmann, (1989).

[2] ARIMURA, A.: *Completeness of depth-bounded resolution in logic programming*, Proc. of 6th Japan Society for Software Science and Technology Conference, (1989).

[3] CAVEDON, L.: *Continuity, consistency, and completeness properties for logic programs*, Proc. of the Sixth International Conference on Logic Programming, (1989).

[4] GELDER, A.V.: *Negation as failure using tight derivations for general logic programs*, J. of Logic Programming, (1989), 109–133.

[5] KNUTH, D. E.: *The art of computer programming*, volume **1**, chapter 2. Addison Wesley, second edition, (1973).

[6] LLOYD, J. W.: *Foundations of logic programming*, Springer Verlag, second edition, (1987).

[7] PRZYMUSINSKI, T.: *On the declarative semantics of deductive databases and logic programs*, J. Minker, editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, (1988).

[8] REITER, R.: *On closed world data bases*, H. Gallaire and J. Minker, editors, Logic and Data Bases, Plenum, (1978).

[9] SHI, Y.: *The principles of time-bounded knowledge base management systems*, Engineering Sciences Reports, Kyushu University, Vol. **11**, No. 1, June. in Japanese, (1989), 77–84.

[10] SHI, Y., ARIKAWA, S.: *Time-bounded reasoning in first order knowledge base systems*, Proc. of the Logic Programming Conference '**89**, (1989).

[11] WIKFRAM, D. A., MAHER, M. J. and LASEZZ, J.-L.: *A unified treatment of resolution strategies for logic program*, Proceedings of the Second International Conference of Logic Programming, (1984), 263–276.

[12] YAMAMOTO, A.: *Elementary formal system as a logic programming language*, Proc. of the Logic Programming Conference '**89**, (1989).

Received September 2, 1990