Depth-Bounded Inference for Nonterminating Prologs

Arimura, Hiroki
Department of Artificial Intelligence Kyushu Institute of Technology

https://doi.org/10.5109/3139

出版情報：Bulletin of informatics and cybernetics. 25 (3/4), pp.125-136, 1990-10-04. Research Association of Statistical Sciences
バージョン：
権利関係：

Bulletin of Informatics and Cybernetics, Vol. 25, No. 3~4, 1993

DEPTH-BOUNDED INFERENCE FOR NONTERMINATING PROLOGS

By

Hiroki ARIMURA*

Abstract

In this paper, we study the completeness of the depth-bounded resolution, which is an SLD-resolution that prunes infinite derivations using the depth-bound. We introduce a class of definite programs with local variables, called linearly covering programs, and prove the completeness of the depth-bounded resolution for the class with respect to the CWA of Reiter.

1. Introduction

In logic programming, the computation mechanism is provided by the SLD-resolution procedure. Given a goal, the procedure computes a correct answer whenever the termination of the computation is ensured. For correct computations, termination is required.

In inductive knowledge acquisition and inductive logic programming [6], unfortunately, the termination of a program can not be ensured in many cases. In the frame work of inductive logic programming, given examples of the unknown target program, a learning algorithm guesses a hypothesis program from the examples and check the validity of the hypothesis by SLD-resolution. Even if a guessed program is correct, it may be nonterminating. For examples, suppose that true atoms of the following terminating program are given as examples of the target relation.

$$\begin{array}{l} app(u.x,\ y,\ u.z) \leftarrow app(x,\ y,\ z), \\ app(nil,\ x,\ x) \leftarrow . \end{array}$$

Then, a learning algorithm may find a correct but nonterminating program,

$$\begin{array}{l} app(u.x,\ y,\ u.z) \leftarrow app(x,\ y,\ z). \\ app(x,\ y,\ z) \leftarrow app(x,\ y,\ z), \\ app(nil,\ x,\ x) \leftarrow . \end{array}$$

For the latter program, suppose that a learning algorithm tries to check whether a false example $app(a.nil,\ b.nil,\ a.nil)$ is derived. Then, the SLD-resolution procedure never terminates. The SLD-resolution procedure is complete with respect to the *closed world assumption* (CWA, for short) if the derivation procedure effectively finds the answer

* Department of Artificial Intelligence, Kyushu Institute of Technology, Kawazu 680−4, Iizuka 820, Japan

for both of positive and negative literals, that is, the procedure returns a computed answer if the given atom is true and stops with failed state if the atom is not correct [9]. Because of the nontermination, the SLD-resolution procedure is not complete for CWA.

The depth-bounded resolution [1] is an SLD-resolution augmented by a mechanism to prune such infinite derivations. Given a goal, the procedure computes the value of depth-bound, then it tries to construct a refutation from it whenever the length of the derivation is smaller than the depth-bound.

The completeness of the depth-bounded procedure depends on the function to compute the adequate depth-bound. Unfortunately, since we can not effectively decide the termination of a given program, it is impossible to compute such a depth-bound. Thus, we have to pay attention to a subclass of logic programs for which the derivation is complete.

In our previous work [1, 10], we present the class of *weakly reducing programs*, for which the depth-bounded resolution is complete with respect to CWA. However, the class is too restricted. Programs in this class can not have any local variables, where local variables are variables occurring only in the body of a clause.

In this paper, we propose another class of logic programs, called *linearly covering programs*, which have a mode declaration for each predicate. The class allows that clauses contain local variables. For example, the following program is linearly covering, which is not weakly reducing.

$$\begin{aligned}
&mesh(x;\ y) \leftarrow same(x;\ z),\ mesh(z;\ y),\\
&mesh(x;\ 1) \leftarrow short-for-hole(x;),\\
&same(c15;\ c16) \leftarrow,\ same(c16;\ c17) \leftarrow,\\
&short-for-hole(a16;) \leftarrow.
\end{aligned}$$

A program in the class has a good property that it returns an ground answer for the output, whenever input arguments are ground.

After introducing the class, we show that the problem of deciding whether a program is in the class is polynomial time decidable and prove the correctness of the depth-bounded resolution for the class with respect to CWA. We further characterize a sufficient condition for the completeness by extending the condition studied in our previous paper.

2. Preliminaries

In this section, we introduce logic programs and give a brief review of their declarative and procedural semantics.

For defining logic program with mode declaration, we need some definitions on multisets and operations over them. *Multisets* are corrections of objects in which an object can occurs more than once. We denote multisets by lower case bold face letters $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}_1, \mathbf{x}_2, \ldots$. For an object a and a multiset $\mathbf{x}$, we denote by $Oc(a, \mathbf{x})$ the *number of occurrences* of a in $\mathbf{x}$. For example, if $\mathbf{x} = \{x, x, x, y\}$, then $Oc(x, \mathbf{x}) = 3$, $Oc(y, \mathbf{x}) = 1$ and $Oc(z, \mathbf{x}) = 0$. We write $\mathbf{x} \subseteq \mathbf{y}$ if $Oc(x, \mathbf{x}) - Oc(x, \mathbf{y}) \geq 0$ for any object x, and define the *sum* $\mathbf{x} + \mathbf{y}$ as the multiset for which $Oc(x, \mathbf{x} + \mathbf{y}) = Oc(x, \mathbf{x}) + Oc(x, \mathbf{y})$, and the *difference* $\mathbf{x} - \mathbf{y}$ as the multiset for which $Oc(x, \mathbf{x} - \mathbf{y}) = Oc(x, \mathbf{x}) - Oc(x, \mathbf{y}) \geq 0$.

Note that the difference $\mathbf{x} - \mathbf{y}$ is defined only if $\mathbf{x} \subseteq \mathbf{y}$. The operator Π stands for the infinite sum of multisets. The size $|\mathbf{x}|$ of $\mathbf{x}$ is defined as $|\mathbf{x}| = \Sigma_{\forall x} Oc(x, \mathbf{x})$. For sets, relations $\in$, $\subseteq$ and operations union $\cup$, intersection $\cap$ are defined by usual manner. The size $|\mathbf{x}|$ of a multiset $\mathbf{x}$ is the total number of elements in $\mathbf{x}$.

Next, we introduce the language for logic programs. Let Σ, X and Π be disjoint sets. We assume that Σ and Π are finite. We refer to elements of Σ as *function symbols*, to elements of X as *variables* and to elements of Π as *predicate symbols*. Each element of Σ and Π are associated with a nonegative integer called an *arity*. A *term* is a variable, a function symbol of arity 0, or an expression of the form $f(t_1, \ldots, t_n)$, where f is a function symbol of arity $n \geq 0$ and $t_1, \ldots, t_n$ are terms. The size $|t|$ of a term t is the total number of occurrences of variables and function symbols in t. We denote terms by $s, t, u, t_1, t_2, \ldots$ and sequences of terms by bold face letters $\mathbf{s}, \mathbf{t}, \mathbf{u}, \mathbf{t}_1, \mathbf{t}_2, \ldots$. An *atomic formula* (atomic for short) is an expression $p(t_1, \ldots, t_n)$ for which p is predicate symbol of arity $n \geqslant 0$ and $t_1, \ldots, t_n$ are terms. In our language, we assume that all the predicate symbols are associated with a mode.

Definition 2.1. A mode for a predicate symbol p of arity n is a pair (I, O) of disjoint sets of positive integers for which $I + O = \{1, \ldots, n\}$. Given an atom $A = p(t_1, \ldots, t_n)$ and a mode (I, O) for p, the *input* (*output*) *argument* of A is a multiset A/I (A/O) of terms, where for a set $J \subseteq \{1, \ldots, n\}$, A/J is the multiset defined as $A/J = \{t_i \mid i \in J\}$.

For simplicity, we assume that all the input arguments are preceding output arguments. Thus, we write a moded atom, using a semicolon ";" as a punctuation symbol, $p(s_1, \ldots, s_m; t_1, \ldots, t_n)$ or $p(\mathbf{s}; \mathbf{t})$, where $\mathbf{s} = s_1, \ldots, s_m$ and $\mathbf{t} = t_1, \ldots, t_n$ are the *input* and the *output arguments*, respectively.

Definition 2.2. A *goal* is a clause of the form $\leftarrow A_1, \ldots, A_n$ and a definite clause is a clause of the form $A \leftarrow A_1, \ldots, A_n$ $(n \geq 0)$. We do not distinguish two goals with different order of atoms so that we can treat a sequence of atoms as a multiset. For instance, we can write $A_1, \ldots, A_m + B_1, \ldots, B_n = A_1, \ldots, A_m, B_1, \ldots, B_n$. A program is a finite set P of definite clauses.

We describe the semantics of programs. Terms and definitions not found below will be found in Lloyd [5]. Let P be a program. The *Herbrand base*, denoted by $B(P)$, is the set of all the ground atoms in the language of P and an *Herbrand interpretation* is a subset I of $B(P)$. Given an Herbrand interpretation I, we can define the truth value of an atom A as A is true in I if $A \in I$ and A is false in I if $A \notin I$. Based on this interpretation, we can determine the truth value of a first order formula. An *Herbrand model* of P is an Herbrand interpretation I in which all the clauses in P are true. For a program consisting only of definite clauses, the intersection of its Herbrand models is again an Herbrand model.

Definition 2.3. The *least Herbrand model* of P is the set

$$M(P) = \cap\{I \subseteq B(P) \mid I \text{ is an Herbrand model of } P\}.$$

The least Herbrand model can be characterized as the least fixed point $T_P \uparrow \omega$ of a monotone function on the power set of $B(P)$.

DEFINITION 2.4. T_P: $2^{B(P)} \to 2^{B(P)}$ is a mapping defined as follows. For a subset I of $B(P)$,

$$T_P(I) = \left\{ A \in B(P) \;\middle|\; \begin{array}{l}\text{There is a ground instance } A \leftarrow A_1, \ldots, A_m \text{ of} \\ \text{a clause in } P \text{ such that } \{A_1, \ldots, A_m\} \subseteq I. \end{array} \right\}.$$

Then,

$$\begin{array}{ll} T_P \uparrow 0 = \emptyset, & \\ T_P \uparrow n = T_P \uparrow (n-1), & \text{for a (finite) positive integer } n, \\ T_P \uparrow \omega = \cup_{n<\omega} T_P \uparrow n, & \text{for the first transfinite ordinal } \omega. \end{array}$$

THEOREM 2.1. (Lloyd [5]) *For a program P, $M(P) = T_P \uparrow \omega$.*

One of the semantics most commonly accepted in logic programming is the *logical consequence semantics*. That is, for an ground atom A, $P \models A$ iff $A \in M(P)$. However, this semantics does not describe how to derive a false atom $\neg A$ from P. The *closed world assumption semantics* (CWA, for short) is defined as $P \models A$ iff $A \in M(P)$ and $P \models \neg A$ iff $A \notin M(P)$ [9]. The CWA supplies a fundamental semantics in deductive databases.

For programs, the procedural semantics is provided by the *SLD-resolution*. If clauses C and D coincide by renaming of variables, we say D is a *variant of* C. A *computation rule* is a function R that, given a goal, selects an atom from the goal.

DEFINITION 2.5. Let R be a computation rule, P be a program and G be a goal. A *derivation* from G is a (finite or infinite) sequence of triples (G_i, θ_i, C_i) $(i = 0, 1, \ldots)$ which satisfies the following conditions:

(2.1) G_i is a goal, θ_i is a substitution, C_i is a variant of a clause in P, and $G_0 = G$.

(2.2) $v(C_i \cap C_j) = \emptyset$ for every i and j such that $i \neq j$, and $v(C_i \cap G_i) = \emptyset$ for every i.

(2.3) If $A \in G_i$ is the atom selected by R, then C_i is $B \leftarrow B_1, \ldots, B_m$ $(m \geq 0)$ and θ_i is the most general unifer of A and B, and G_{i+1} is $((G_i - \{A\}) + \{B_1, \ldots, B_m\})\theta_i$. We say G_{i+1} is a *resolvent* of G_i and C_i by θ_i.

DEFINITION 2.6. A derivation (G_i, θ_i, C_i) $(i = 0, \ldots, n)$ is a *refutation* if G_n is the empty goal $\emptyset$ and is a *finitely failed* derivation if there is no resolvent from G_n and a variant of a clause in P. The answer for G is the substitution $\theta = \theta_0 \cdots \theta_{n-1}$.

The SLD-resolution is *sound and complete with respect to the logical consequence semantics*, that is, $P \models A$ iff there is a refutation from the goal $\{\leftarrow A\}$. However, it is not complete with respect to the CWA in the sense that if $P \models \neg A$, it is possible that there is no finitely failed derivation from $\{\leftarrow A\}$. This incompleteness comes from the fact that the complement $B(P) - M(P)$ of $M(P)$ is no longer recursively enumerable in general even if $M(P)$ is recursively enumerable.

3. Linearly Covering Programs

In this section, we deal with logic programs where the mode is declared for every predicate. Consider the following property of programs; for any goal $G = \leftarrow p(\mathbf{s};\, \mathbf{u})$ such that the input argument $\mathbf{s}$ is ground, if the derivation from G succeeds, then the answer substitution makes the output argument $\mathbf{u}$ ground. Then, we say the program is of *ground input and output* or is data-driven [2] (ground I/O, for short). The property is useful in program analysis and program transformation. However, the problem of deciding whether a given program is of ground I/O is undecidable [2]. We introduce a class of logic programs with ground I/O, linearly covering programs, for which the decision of the class is efficiently decidable.

We assume a special symbol $\mathbf{1}$ which is not contained in any of Σ, X and Π. Let $\mathbf{X}$ be the set of all the multiset consists of elements of $X \cup \{\mathbf{1}\}$. Elements of $\mathbf{X}$ are called *carriers*. Terms and sequences of terms are associated with carriers in $\mathbf{X}$ by a function $[\cdot]$ in the following way. For a term t, if t is a variable x, $[t] = \{x\}$ and if $t = f(t_1, \ldots, t_n)$ with $f \in \Sigma$, $[t] = \{\mathbf{1}\} + [t_1] + \ldots + [t_n]$. For a sequence $\mathbf{t} = t_1, \ldots, t_n$ of terms, $[\mathbf{t}] = [t_1] + \ldots + [t_n]$. The function $[\cdot]$ preserves the size of terms in the sense that $|t| = |[t]|$. Let $\mathbf{A}$ be the set of all the multiset of atoms with mode. An *argumented goal* is a triple $B = (G, \mathbf{x}, \mathbf{y})$ for which G is a goal, $\mathbf{x}, \mathbf{y} \in \mathbf{X}$ are carriers.

Definition 3.1. Let $B = (G, \mathbf{x}, \mathbf{y})$ be an argumented goal. A *proof* π for B is a labeled tree defined as follows.

(3.1) If G is the empty goal $\emptyset$ and either $\mathbf{x} = \mathbf{y}$ or $\mathbf{y} = \emptyset$, then a single node labeled with $(\mathbf{x}, \mathbf{y})$ is a proof for G, denoted by $\pi = (\emptyset, \mathbf{x}, \mathbf{y})$. If $\mathbf{x} = \mathbf{y}$, π is called an *identity* proof. If $\mathbf{y} = \emptyset$, π is called an *absorption* proof. The depth of π is 0.

(3.2) If G is a singleton goal $\{A\}$ for which $A = p(\mathbf{s};\, \mathbf{t})$, and $\mathbf{x} = [\mathbf{s}]$ and $\mathbf{y} = [\mathbf{t}]$, then a single node labeled with $(\mathbf{x}, \mathbf{y})$ is a proof for G, denoted by $\pi = (\{A\}, \mathbf{x}, \mathbf{y})$, which is called an *atomic* proof. The depth of π is 0.

(3.3) If G is a sum of two goals $G_1 + G_2$ for which there are proofs $\pi_1 = (T_1, \mathbf{x}_1, \mathbf{y}_1)$ and $\pi_2 = (T_2, \mathbf{x}_2, \mathbf{y}_2)$ for G_1 and G_2, respectively, then π is a tree for which the label of the root N is $(\mathit{parallel}, \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2)$, the left child of N is the root of a proof π_1 for G_1 and the right child of N is the root of a proof π_2 for G_2. π is called a *parallel* proof and denoted by $\pi = (\pi_1 + \pi_2, \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2)$. If the depth of π_1 and π_2 are n_1 and n_2, respectively, the depth of π is $\max\{n_1, n_2\} + 1$.

(3.4) If G is a sum of two goals $G_1 + G_2$ for which there are proofs $\pi_1 = (T_1, \mathbf{x}, \mathbf{z})$ and $\pi_2 = (T_2, \mathbf{z}, \mathbf{y})$ for G_1 and G_2, respectively, then π is a tree for which the label of the root N is $(\mathit{series}, \mathbf{x}, \mathbf{y})$, the left child of N is the root of a proof π_1 for G_1 and the right child of N is the root of a proof π_2 for G_2. π is called a *series* proof and denoted by $\pi = (\pi_1 \cdot \pi_2, \mathbf{x}, \mathbf{y})$. If the depth of π_1 and π_2 are n_1 and n_2, respectively, the depth of π is $\max\{n_1, n_2\} + 1$.

Definition 3.2. An argumented goal $B = (G, \mathbf{x}, \mathbf{y})$ is a *block* if there is a proof π

for B. A definite clause $C = p(\mathbf{s}; \mathbf{t}) \leftarrow G$ is *linearly covering* if $(G, [\mathbf{s}], [\mathbf{t}])$ is a block, and a program is *linearly covering* if its definite clauses are all *linearly covering*.

The class of linearly covering programs is incompatible with that of weakly reducing definite programs [1, 10]. Intuitively, we can think of a block $B = (G, \mathbf{x}, \mathbf{y})$ as a network of pipes along which fluid can flow, $\mathbf{x}$ as the source and $\mathbf{y}$ as the sink of the network. We can construct a larger network by combining two network modules in parallel or in series.

For a block B, the proof of B is not explicit in the syntax of B. The following procedure computes a proof of a block.

ALGORITHM 3.1.

input: a goal G and carrier multisets $\mathbf{x}$, $\mathbf{y}$;
Output: *yes* or *no*;

```
begin
    z: = x;
    While G ≠ Ø do
        if ∃A = p(s; t) ∈ G such that [s] ⊆ z then
            z: = (z − [s]) + [t] and G: = G − {A}
        else return no;
    if and y ⊆ z then return yes else no;
end;
```

PROPOSITION 3.1. *There is an algorithm that, given a program P, decides in polynomial time in the size of P as an expression whether P is linearly covering.*

PROOF. For a clause $C = p(\mathbf{s}; \mathbf{t}) \leftarrow G$, we check whether $(G, [\mathbf{s}], [\mathbf{t}])$ is a block by Algorithm 3.1. We can easily see that the algorithm runs in polynomial time. Thus, we show that (a) given a block $B = (G, \mathbf{x}, \mathbf{y})$ as the input, ALGORITHM 3.1 returns *yes*, iff(b) there is a proof π for B. First we show (a) $\Rightarrow$ (b). Assume that for an input G and carrier multiset $\mathbf{x}$, $\mathbf{y}$, the algorithm computes the sequence $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_n$ $(n \geq 0)$ of values of $\mathbf{z}$ for which $\mathbf{x} = \mathbf{z}_0$ and $\mathbf{y} \subseteq \mathbf{z}_n$, and the sequence $A_0, A_1, \ldots, A_n$ of moded atoms A in the while loop. For each $1 \leq i \leq n-1$, we can construct an atomic proof π_i for the block $B_i = (\{A_i\}, \mathbf{z}_i, \mathbf{z}_{i+1})$ by merging an atomic proof α_i for $(\{A_i\}, [\mathbf{s}_i], [t_i])$ and an identity proof β_i for $(\emptyset, \mathbf{z}_{ii} - [\mathbf{s}_i], \mathbf{z}_i - [\mathbf{s}_i])$ in parallel, since $[\mathbf{s}_i] \subseteq \mathbf{z}_i$ and $\mathbf{z}_{i+1} = (\mathbf{z}_i - [\mathbf{s}_i]) + [\mathbf{t}_i]$ for $A_i = p(\mathbf{s}_i; \mathbf{t}_i)$. Thus, we can construct a proof for $(G, \mathbf{z}_0, \mathbf{z}_n)$ by connecting $\pi_1, \ldots, \pi_{n-1}$ in series, and the claim immediately follows.

Conversely, we show (b) $\Rightarrow$ (a). The point of the proof is to ensure the correctness for arbitrary selections of atoms from G. Assume that $B = (G, \mathbf{z}, \mathbf{y})$ has a proof π. By induction on the depth of π, we can show that if G is nonempty, then there is an atom $A = p(\mathbf{s}; \mathbf{t}) \in G$ such that $[\mathbf{s}] \subseteq \mathbf{z}$. Let $\mathbf{z}' = (\mathbf{z} - [\mathbf{s}]) + [\mathbf{t}]$ and $G' = G - \{A\}$. Next, we prove that G' has a proof for $B' = (G', \mathbf{z}', \mathbf{y})$ by induction on the depth n of π. If the depth of π is 0, the claim is obvious. Assume that for any block with depth less than n, the claim holds.

If $\pi = (\pi_1 + \pi_2, \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2)$ is a parallel proof, then there are proofs $\pi_1 =$

$(T_1, \mathbf{x}_1, \mathbf{y}_1)$ and $\pi_2 = (T_2, \mathbf{x}_2, \mathbf{y}_2)$ for G_1 and G_2, respectively, where $G = G_1 + G_2$. Assume that $A \in G_1$ and let $G_1' = G_1 - \{A\}$. Using the induction hypothesis, G_1 has a proof $\pi_1' = (T_1', \mathbf{x}_1 - [\mathbf{s}] + [t], \mathbf{y}_1)$. Thus, we can combine π_1' and π_2 in parallel to obtain $\pi' = (\pi' + \pi_2, \mathbf{x}_1 - [\mathbf{s}] + [t] + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2) = (\pi_1' + \pi_2, \mathbf{z}', \mathbf{y})$ for B' because $\mathbf{x}_1 + \mathbf{x}_2 = \mathbf{z}$ and $\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{y}$.

On the other hand, if $\pi = (\pi_1 \cdot \pi_2, \mathbf{z}, \mathbf{y})$ is a series proof, then there are proofs $\pi_1 = (T_1, \mathbf{z}, \mathbf{w})$ and $\pi_2 = (T_2, \mathbf{w}, \mathbf{y})$ for G_1 and G_2, respectively. Assume that $A \in G_1$ and let $G_1' = G_1 - \{A\}$. Using the induction hypothesis, G_1 has a proof $\pi_1' = (T_1', \mathbf{z} - [\mathbf{s}] + [t], \mathbf{w})$. Thus, we can obtain the proof $\pi' = (\pi_1' \cdot \pi_2, \mathbf{z} - [\mathbf{s}] + [t], \mathbf{y})$ for B' by combining π_1' and π_2 in series. Also in the case that $A \in G_2$, we can prove the claim similarly. From the consideration above, the algorithm finds an atom $A = p(\mathbf{s}; \mathbf{t}) \in G$ such that $[\mathbf{s}] \subseteq \mathbf{z}$ whenever $G \neq \emptyset$, and $\mathbf{y} \subseteq \mathbf{z}$ when G gets $\emptyset$. ■

Let θ be a substitution. We extend substititions for carriers by defining $\mathbf{x}\theta = \Pi_{x \in \mathbf{x}}\{x\theta\}$, where $\mathbf{1}\theta = \mathbf{1}$.

Lemma 3.2. *Let $B = (G, \mathbf{x}, \mathbf{y})$ and $B' = (G', \mathbf{x}\theta, \mathbf{y}\theta)$ be argumented goals for which G' is a resolvent of G and a linearly covering clause C by θ. Then, if B is a block, then B' is a block.*

Proof. Assume that B is a block. For any atomic block $([p(\mathbf{s}; \mathbf{t})], [\mathbf{s}], [\mathbf{t}])$ and any θ, $(\{p(\mathbf{s}; \mathbf{t})\theta\}, [\mathbf{s}]\theta, [\mathbf{t}]\theta)$ is also an atomic block. Based on this fact, we can easily see the following claim by the induction on the depth of a proof.

Claim. For any substitution θ, if $(G, \mathbf{x}, \mathbf{y})$ is a block, then $(G\theta, \mathbf{x}\theta, \mathbf{y}\theta)$ is a block.

Let $C = p(\mathbf{s}; \mathbf{t}) \leftarrow D$. Since C is linearly covering. $(D, [\mathbf{s}], [\mathbf{t}])$ is a block. We prove the result by induction on the depth of the proof for B. If G is a singleton $[A]$, then $G' = D\theta$, $\mathbf{x}\theta = [\mathbf{s}]\theta$ and $\mathbf{y}\theta = [\mathbf{t}]\theta$ because θ is the mgu of A and $p(\mathbf{s}; \mathbf{t})$. Thus, the result immediately holds from Claim.

Next we assume that the result holds for any block with a proof of depth less than n and B has the proof of depth n. Suppose that $B = (G_1 + G_2, \mathbf{x}_1 + \mathbf{x}_2, \mathbf{y}_1 + \mathbf{y}_2)$ is obtained by combining blocks $B_1 = (G_1, \mathbf{x}, \mathbf{y}_1)$ and $B_2 = (G_2, \mathbf{x}_2, \mathbf{y}_2)$ in parallel. Without loss of generality, we assume $A \in G_1$. Let G_1' be the resolvent of G_1 and C by θ, and G_2' be $G_2\theta$. Then, $G_1' = ((G_1 - \{A\}) + D)\theta$. By the induction hypothesis and Claim, $B_1' = (G_1', \mathbf{x}_1\theta, \mathbf{y}_1\theta)$ and $B_2' = (G_2', \mathbf{x}_2\theta, \mathbf{y}_2\theta)$ are blocks. Thus, we obtain a block $B' = (G_1' + G_2', (\mathbf{x}_1 + \mathbf{x}_2)\theta, (\mathbf{y}_1 + \mathbf{y}_2)\theta)$ combining B_1 and B_2 in parallel. In the case that B is a series block, we can show the result in the same way. ■

Lemma 3.3. *Let P be a linearly covering program and $B = (G, \mathbf{x}, \mathbf{y})$ be a block. If there is a refutation from G with the answer θ, then $\mathbf{x}\theta \supseteq \mathbf{y}\theta$ and $\mathbf{x}\theta \supseteq \mathbf{u}\theta \supseteq \mathbf{v}\theta$ for each atom $p(\mathbf{s}; \mathbf{t}) \in G$, $\mathbf{u} = [\mathbf{s}]$ and $\mathbf{v} = [\mathbf{t}]$.*

Proof. Suppose that there is a refutation (G_i, θ_i, C_i) $(i = 0, \ldots, n)$ from G with the answer $\theta = \theta_0 \ldots \theta_{n-1}$. Let B_i be an argumented goal $(G_i, \mathbf{x}\theta_0 \ldots \theta_{i-1}, \mathbf{y}\theta_0 \ldots \theta_{i-1})$ for every $i = 1, \ldots, n$. By Lemma 3.2, the last augumented goal $B_n = (\emptyset, \mathbf{x}\theta, \mathbf{y}\theta)$ is a block because the initial B_0 is a block. For any $B = (G, \mathbf{x}, \mathbf{y})$, we can show that if G

is the empty goal, then $\mathbf{x} \supseteq \mathbf{y}$ be induction on the depth of the proof for B. Hence, $\mathbf{x}\theta \supseteq \mathbf{y}\theta$ is proved. For each $p(\mathbf{s}; \mathbf{t}) \in G$, we can show that there is a subgoal $G' \subseteq G$ that satisfies the following conditions: (1) G' contains $p(\mathbf{s}; \mathbf{t})$, (2) $(G', \mathbf{x}, \mathbf{w})$ is a block for both of $\mathbf{w} = \mathbf{u}$ and $\mathbf{w} = \mathbf{v}$, and (3) there is a refutation from G' with the answer θ' for which $G'\theta$ is an instance of $G'\theta'$. Thus, by applying the result proved above, we can show $\mathbf{x}\theta \supseteq \mathbf{u}\theta$. Since clearly $\mathbf{u}\theta \supseteq \mathbf{v}\theta$, the result immediately follows. ■

THEOREM 3.4. *Any linearly covering program has ground I/O property.*

PROOF. Since $(p(\mathbf{s}; \mathbf{u}), [\mathbf{s}], [\mathbf{u}])$ is a block for an atom $p(\mathbf{s}; \mathbf{u})$, the result immediately follows from LEMMA 3.3. ■

DEFINITION 3.3. Let P be a program. A *supporting clause* for P is a ground instance $C = a \leftarrow a_1, \ldots, a_m$ $(m \geq 0)$ of a clause in P for which $\{a_1, \ldots, a_m\} \subseteq M(P)$.

THEOREM 3.5. *Let P be a linearly covering program. Then,*

(3.5) *for any supporting clause $a \leftarrow b_1, \ldots, b_m$ of P and any atom $b = b_i$ $(1 \leq i \leq m)$ in the body, the size of a/I is greater than or equal to the size of b/I', $|a/I| \geq |b/I'|$, where a/I is the input arguments of a and b/I' is the input arguments of b, and*

(3.6) *for any ground atom $a \in M(P)$, the size of a/I is greater than or equal to the size of a/O, $|a/I| \geq |a/O|$,* where *a/I and a/O are the input arguments and the output arguments of a, respectively.*

PROOF. Assume that there is a supporting clause $C = a \leftarrow b_1, \ldots, b_m$ for which $b = b_i$. Then, C is an instance of a clause $D = A \leftarrow B_1, \ldots, B_m$ in P. Let $A = p(\mathbf{s}; \mathbf{t})$ and $B_i = q(\mathbf{u}; \mathbf{v})$. Then, there is a substitution θ for which a $= A$ and $b_i = B_i\theta$. If D is linearly covering, then $(\{B_1, \ldots, B_m\}, [\mathbf{s}], [\mathbf{t}])$ is a block. Thus, $(\{b_1, \ldots, b_m\}, [\mathbf{s}]\theta, [\mathbf{t}]\theta)$ is a block. From the completeness of SLD-resolution, $\{B_1, \ldots, B_m\} \subseteq M(P)$ implies that there is a refutation from $\{b_1, \ldots, b_m\}$ with the identity substitution as the answer. Therefore, LEMMA 3.3 derives that $|[\mathbf{s}]\theta| \geq |[\mathbf{u}]\theta|$. Hence, (a) is proved. LEMMA 3.3 also derives $|[\mathbf{s}]\theta| \geq |[\mathbf{t}]\theta|$. For any ground atom $a \in M(P)$, there is a supporting clause such as C. Thus, the result for (b) immediately follows. ■

THEOREM 3.5 says that for any successful computation, the size of the output does not exceed that of the input.

4. Depth-bounded resolution

As we have already seen, the SLD-resolution is not complete for the CWA. The existence of infinite loops causes the problem. In this section, we introduce the resolution procedure augumented with a mechanism that prunes infinite loops by bounding the depth of derivation proofs. Suppose that $\mathbf{P}$ be a class of programs and $\mathbf{G}$ is a class of goals, or queries.

DEFINITION 4.1. (Arimura [1]) A depth function is a function $f: \mathbf{G} \times \mathbf{P} \rightarrow N$ that is totally computable.

DEFINITION 4.2. (Arimura [1]) Let f be a depth function, R be a computation rule, P be a program and G be a goal. A *depth-bounded derivation* from G is a sequence of quadruple $(G_i, \theta_i, C_i, c_i)$ $(i = 0, 1, \ldots)$ which satisfies the following conditions:

(4.1) G_i is a goal, θ_i is a substitution, C_i is a variant of a clause in P, c_i is a function $c_i: G_i \to N$ called a *counter*, $G_0 = G$ and $c_0(A) = f(G, P)$ for all $A \in G_0$.

(4.2) $c_i(A) \geq 0$ for all $A \in G_i$.

(4.3) $v(C_i \cap C_j) = \emptyset$ for every i and j such that $i \neq j$, and $v(C_i \cap G_i) = \emptyset$ for every i.

(4.4) If $A \in G_i$ is the atom selected by R, then C_i is $B \leftarrow D$ and θ_i is the most general unifier of A and B, and G_{i+1} is $((G_i - \{A\}) + D)\theta_i$.

(4.5) For every $E\theta_i \in G_{i+1}$, if $E\theta_i$ is an introduced atom, that is, $E \in D$, then $c_{i+1}(E\theta_i) = c_i(A) + 1$. Otherwise, $c_{i+1}(E\theta_i) = c_i(E\theta_i)$.

DEFINITION 4.3. (Arimura [1]) A depth-bounded derivation $(G_i, \theta_i, C_i, c_i)$ $(i = 0, 1, \ldots)$ is a *refutation* if G_n is the empty goal $\emptyset$ and is *failed* derivation if there is no resolvent from G_n and a variant of a clause in P. The answer for G is the substitution $\theta = \theta_0 \ldots \theta_{n-1}$.

Every depth-bounded derivation is finite, and the set of all the resolvents of a goal is finite. Thus, given a goal G, we can effectively find a depth-bounded refutation from G if it exists. Unfortunately, depth-bounded derivation procedure is neither sound nor complete with respect to CWA in general. We characterize a subclass of logic programs for which the depth-bounded derivation procedure is sound and complete.

DEFINITION 4.4. The relation $>$ is the transititve relation defined as the smallest relation satisfying the following condition. For any $a, b \in B(P)$, $a > b$ if there is a supporting clause $a \leftarrow b_1, \ldots, b_m$ for which $b = b_i$ for some $1 \leq i \leq m$. The relation $>$ is called the *supported priority relation* of P.

DEFINITION 4.5. For a set S, a relation $>$ on S is *locally finite* if the set

$$S|_{a>} = \{b \in S|\ a > b\}$$

of descendants of a is finite for any $a \in S$.

LEMMA 4.1. *Let P be a program whose supported priority relation $>$ is locally finite. Then, for any ground atom $a \in B(P)$ and the cardinality k of $B(P)|_{a>}$,*

$$a \in T_p \uparrow \omega \Leftrightarrow a \in T_p \uparrow k.$$

PROOF. Let T_p^a be the function on the powerset of $B(P)|_{a>}$ obtained from T_p by restricting the domain to $B(P)|_{a>}$, that is, $T_p^a(I) = T_p(I) \cap B(P)|_{a>}$. For every supporting clause $b \leftarrow b_1, \ldots, b_m$, if $b \in B(P)|_{a>}$ then $b_i \in B(P)|_{a>}$ for all b_i. Thus, we can prove that $T_p \uparrow \omega \cap B(P)|_{a>} = T_p^a \uparrow \omega$. Since T_p^a is monotonic and the power set of $B(P)|_{a>}$ is finite, the limit $T_p^a \uparrow \omega$ coincides to $T_p \uparrow k$, where $k = |B(P)|_{a>}|$. ■

LEMMA 4.2. *Let* $\mathbf{P}$ *be a class of programs such that for any program P in the class, the supported priority relation* $>$ *is locally finite, and f be a depth function such that* $f(a, P) \geq |B(P)|_{a>}|$ *for any* $a \in B(P)$ *and any* $P \in \mathbf{P}$. *Then, for any ground atom* $a \in B(P)$,

(4.6) $a \in M(P)$, *iff*

(4.7) *there is a depth-bounded refutation from* $\leftarrow a$ *by f.*

PROOF. Let P be a program, $a \in B(P)$ be a ground atom and k be a positive integer. Assume that $f(a) \geq k$. Then, it is easily seen by the induction on k that $a \in T_p \uparrow k$ iff there is a depth-bounded refutation from $\leftarrow a$ by f using standard technique as described in Lloyd [5]. Thus, the result immediately follows from LEMMA 4.1. ∎

LEMMA 4.3. (Arimura [1]) *Let* Σ *and* Π *are finite sets of function symbols and predicate symbols, respectively. For any fixed size n, the number of all the atoms of size less than or equalt to n is at most* $O(2^{2^{cn}})$ *for some* $c \geq 0$. *The constant c can be effectively computed from* Σ *and* Π.

PROOF. The number of ordered trees of a fixed size is bounded by $2^{c'n}$ for some $c' \geq 0$ [4]. Thus, the result holds for fixed finite alphabets Σ and Π. ∎

LEMMA 4.4. *For a linearly covering program P, The supported priority relation* $>$ *of P is locally finite. Moreover, there is a depth function f such that* $f(a, P) \geq |B(P)|_{a>}|$ *for any linearly covering program P* and $a \in B(P)$.

PROOF. If $a > c$ then, there is a supporting clause $C = a \leftarrow b_1, \ldots, b_m$ for which $b = b_i$. Thus, $|a/I| \geq c/I'|$ by THEOREM 3.5. Therefore, for any $c \in B(P)$, if $a > c$ then $|a/I| \geq |c/I''|$, where c/I'' is the input arguments of c. THEOREM 3.5 also says that for any ground atom $c \in M(P)$, $|c/I| \geq |c/O|$ holds, where c/I and c/O are the input arguments and the output arguments of c, respectively. This implies tht the size $|c|$ of c is bounded by $2|a|$. By LEMMA 4.3, there is a total computable function f satisfying $f(a, P) \geq |B(P)|_{a>}|$. Hence, the set $|B(P)|_{a>}|$ is finite. ∎

THEOREM 4.5. *For the class of linearly covering programs, there is a depth function f for which for any ground atom* $a \in B(P)$ *and any program P in the class,*

(4.8) $a \in M(P)$, *iff*

(4.9) *there is a depth-bounded refutation from* $\leftarrow a$ *by f.*

PROOF. By LEMMA 4.2 and LEMMA 4.4. ∎

From THEOREM 3.4, we can strengthen the result above.

COROLLARY 4.6. *For the class of linearly covering programs, there is a depth function f that satisfies the following condition. For any atom A for which the input arguments is ground and any program P in the class.*

(4.10) $M(P) \models A\theta$ for some grounding substitution θ, iff

(4.11) *there is a depth-bounded refutation from* $\leftarrow A$ *by f.*

5. Related Works

We introduced a subclass of logic programs with the ground I/O property, called linearly covering programs, and proved the completeness of the depth-bounded resolution procedure with respect to CWA for the class, in the sense of effective computations. The class is efficiently decidable from their syntax.

To characterize a subclass of logic programs for which the depth-bounded resolution procedure is complete, we introduced the notion of locally finite relations. We showed that the class of logic programs, denoted by PM here, for which the supported priority relation is locally finite. Przymusinski [8] studies characterizations of several classes of logic programs using the priority relation on B(P) for programs. In the terms of the priority relation, our class PM is characterized as the class where the priority relation restricted to $M(P)$ is locally finite.

There is a class of logic programs, called locally finitely stratified definite programs, for which the depth-bounded resolution is also complete with respect to CWA [1, 10]. Since the class, denoted by PB here, can be characterized as the class where the priority relation on $B(P)$ is locally finite, clearly PM is a super class of PB. Further, any program in PB can contain no variables occurs only in the body of a clause, while PM contains such programs. Thus, the inclusion between PB and PM is proper.

Since the class of linearly covering programs is very restricted, the class excludes many of interesting clauses dealt with in inductive logic programming [6]. Dŏlsak et al. [3] reported that their system GOLEM discovered a logic program describing an adequate mesh for finite element methods from examples of the unknown mesh. The following clauses are a part of discovered knowledge.

$$
\begin{aligned}
&mesh(x;\ y) \leftarrow same(x;\ z),\ cont - loaded(z;),\ mesh(z;\ y);\\
&mesh(x;\ 1) \leftarrow short - for - hole(x;);\\
&same(c15;\ c16) \leftarrow;\ cont - loaded(c16;) \leftarrow;\ short - for - hole(a16;) \leftarrow.
\end{aligned}
$$

Unfortunately, the first clause is not linearly covering. The second atom $cont\_loaded(z;)$ violates the condition. However, since this atom only check whether the argument is equivalent to a constant, *c16*, the existence of such atoms does not effect the flow of data in a clause. Therefore, we can make the condition for linearly covering clauses weaker by treating such atoms as exceptions.

Recently, Plümer [7] independently studies the method based on linear predicate inequalities to characterize more wider class of programs.

Acknowledgements

The motivation of the present paper is due to Prof. Arikawa. The author is also grateful to him for his constant encouragement. The author would like to thank to Akihiro Yamamoto, Hiroki Ishizaka and Prof. Takeshi Shinohara for fruitful discussion, to Prof. Setsuko Otsuki and Prof. Akira Takeuchi for their constant support.

References

[1] ARIMURA, H.: *Completeness of Depth-bounded Resolution for Weakly Reducing Programs*, In Nakata, I. and Hagiya, M., editors, Software Science and Engineering, World Scientifics Series in Computer Science, **31**, (1991), 227–245.

[2] DERANSART, P. and MALUSZYNSKI, J.: *Relating Logic Programs and Attribute Grammars*, J. logic programming, **2**(2), (1985), 119–155.

[3] DŏLSAK, B. and MUGGLETON, S.: *The Application of Inductive Logic Programming to Finite Element Mesh Design*, In Muggleton, S., editor, "Proceedings of International Workshop on Inductive Logic programming", (1991), 225–241.

[4] KNUTH, D.E.: *The Art of Computer Programming*, 1, chapter 2, Addison Wesley, second edition, (1973).

[5] LLOYD, J.W.: *Foundations of Logic Programming*, Springer Verlag, second edition, (1987).

[6] MUGGLETON, S.: *Inductive Logic Programming*, In Arikawa, S. et al, editor, "Proceedings of the First International Workshop on Algorithmic Learning Theory", (1990), 42–62.

[7] PLÜMER, L.: *Termination Proofs for Logic Programs based on Predicate Inequalities*, In Warren, D. H. and Szeredi, H., editors, "Proc. of the seventh International Conference on Logic Programming", The MIT press, (1990), 634–648.

[8] PRZYMUSINSKI, T.: *On the Declarative Semantics of Deductive Databases and Logic Programs*, In Minker, J., editor, Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann, (1988), 193–216.

[9] REITER, R.: On Closed World Data Bases, In Gallaire, H. and Minker, J., editors, Logic and Data Bases, Plenum, (1978), 55–76.

[10] YAMAMOTO, A.: *Procedural semantics and negative information of elementary formal system*, J. logic programming, **13**, (1992), 89–97.

Received October 30, 1990
Revised September 30, 1992
Communicated by S. Arikawa