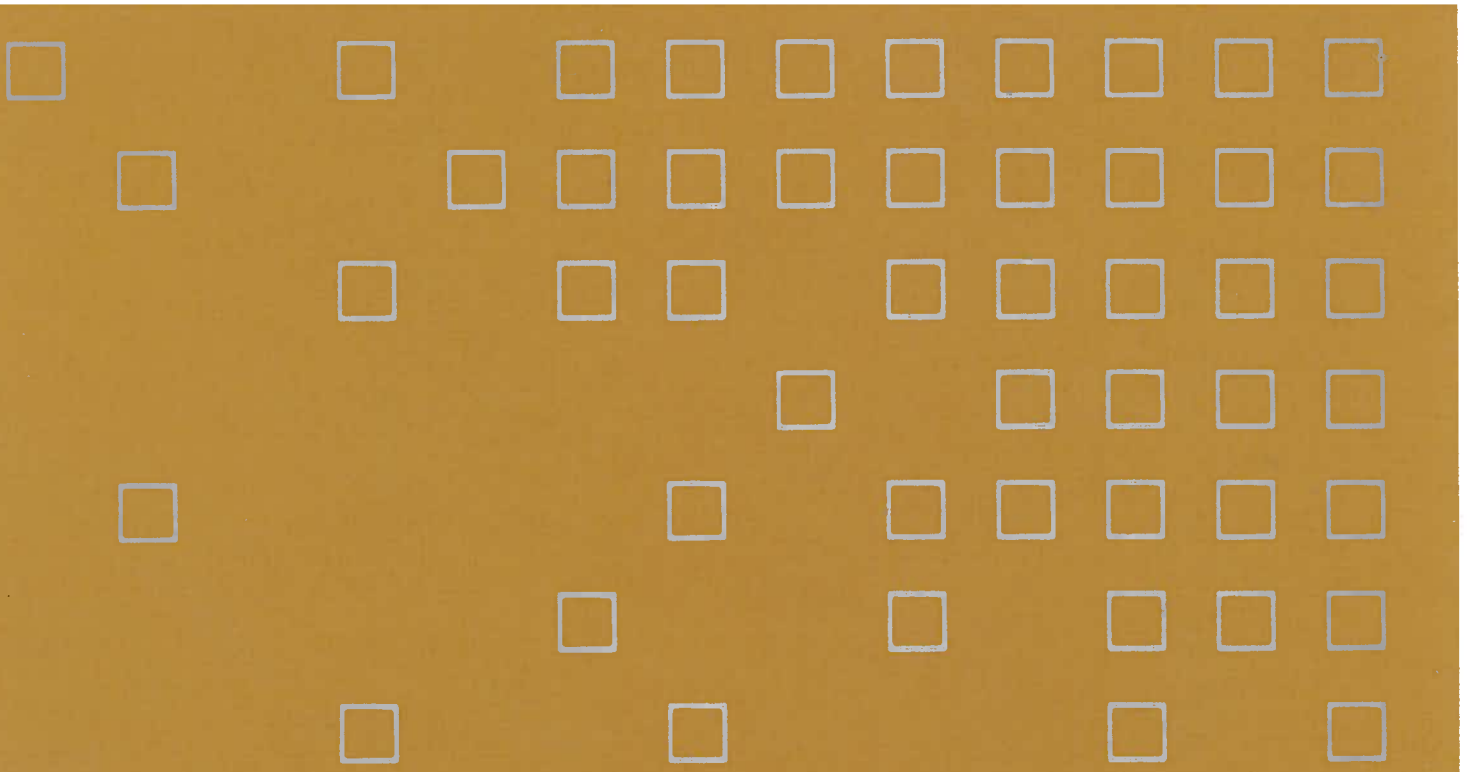

Centrum voor Wiskunde en Informatica

Centre for Mathematics and Computer Science

M. Bezem

Characterizing termination of logic programs with level mappings

Computer Science/Department of Software Technology Report CS-R8912 April

1989

Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

M. Bezem

Characterizing termination of logic programs with level mappings

Computer Science/Department of Software Technology Report CS-R8912 April

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Research (N.W.O.).

Characterizing Termination of Logic Programs with Level Mappings

Marc Bezem
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

We study a large class of logic programs which terminate with respect to a natural class of goals. Both classes are characterized in terms of level mappings. The class of logic programs is strong enough to compute every total recursive function. The class of goals considerably extends the variable-free ones. Based on the ideas developed in this paper we present a technique which improves the termination behaviour of Prolog programs.

1980 Mathematics Subject Classification (1985): 68Q40, 68T15.
1987 CR Categories: F.1.1, F.4.1, I.2.3.
Key Words & Phrases: logic programming, termination, multiset ordering, recursive functions.

I. Introduction

Termination of a logic program with respect to a given goal is of course a problem of the utmost importance. Here termination is meant in the strong sense, i.e. irrespective of the selection of atoms in the goal and of the ordering of the program clauses. In general the least Herbrand model M_P of a logic program P, consisting of all variable-free atoms which logically follow from P, is recursively enumerable. Consequently, the above halting problem is in general unsolvable. Restriction to logic programs P with recursive least Herbrand model M_P does not help very much, not even for variable-free goals, for in general the mechanism of SLD-resolution does not provide a membership test for M_P. Certainly, if a variable-free atom A occurs in M_P, then a breadth-first search procedure in an SLD-tree of $\leftarrow A$ always yields a successful refutation, but for A not in M_P it can happen that this tree is infinite, so that the search procedure does not terminate.

Let us consider the restriction to the class of determinate programs introduced by Blair in [B]. Determinate programs are logic programs with complementary success and finite failure set. Consequently, if P is a determinate program, then a breadth-first search procedure in any fair SLD-tree of a variable-free goal will always terminate. Although this constitutes an improvement, it still leaves a lot to be desired. First, breadth-first search is generally considered inefficient and it is not possible to sharpen the above property of determinate programs to depth-first search procedures. Second, it is desirable to be able to ensure termination for a larger class of goals than just the variable-free ones. Then the program can not only test, but also compute. Third, and less important than the previous two desiderata, we would like to do away with the fairness condition and obtain termination for arbitrary selection rules. Note that these three desiderata ensure the termination of a Prolog-like evaluation of goals from the larger class of goals mentioned above.

The technical tool we shall use is called *level mapping* by Cavedon [C], who studied various classes of logic programs with negation. A level mapping is a function assigning natural numbers to variable-free atoms. Level mappings are a natural refinement of Clark's finite partition of the set of predicate symbols in [Cl]. We show that if a logic program is *recurrent*, i.e. satisfies the condition that heads of variable-free instances of program clauses have higher levels than the atoms occurring in

Report CS-R8912
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

the body of the same instance, then it is terminating with respect to *bounded* goals, i.e. goals whose instances are below some fixed level. This result improves on [C], where only termination with respect to variable-free goals is obtained. We also prove the converse, namely that a logic program is recurrent if it terminates with respect to variable-free goals. Furthermore we show that every total recursive function can be computed by a recurrent program. As the class of recurrent programs can easily be shown to be a strict subclass of the determinate programs, this result improves on Blair [B], who proved that every total recursive function can be computed by a determinate program. Finally we give a technique which can improve the termination behaviour of certain Prolog programs.

2. Recurrent programs

For definitions, terminology and notation concerning logic programming we refer the reader to [A] or [L]. More specifically, for a logic program P we use U_P, B_P, T_P, $T_P\uparrow\alpha$, and $T_P\downarrow\alpha$ as abbreviations of, respectively, the Herbrand Universe of P, the Herbrand Base of P, the immediate consequence operator of P, the upward ordinal powers of T_P, and the downward ordinal powers of T_P. Furthermore we (ab)use $T_P\uparrow\alpha$, $T_P\downarrow\alpha$ as abbreviations of $T_P\uparrow\alpha(\varnothing)$, $T_P\downarrow\alpha(B_P)$, so that $M_P = T_P\uparrow\omega$ denotes the least Herbrand model of the logic program P.

Definition 2.1. Let P be a logic program. A *level mapping* for P is a function $|\ |: B_P \to \mathbb{N}$ of variable-free atoms to natural numbers. For $A \in B_P$ we call $|A|$ the *level* of A.

Definition 2.2. Let P be a logic program and $|\ |$ a level mapping for P. We call P *recurrent with respect to* $|\ |$ if for every variable-free instance $B \leftarrow A_1, \ldots, A_n$ $(n \geqslant 0)$ of a clause from P the level of B is higher than the level of every A_i $(1 \leqslant i \leqslant n)$. Moreover P is called *recurrent* if P is recurrent with respect to some level mapping for P.

In the sequel we shall show that recurrent programs have a good termination behaviour: for a large class of goals, including the variable-free ones, every SLD-derivation from a recurrent program terminates. This class of goals is characterized by Definition 2.3 below. The underlying idea is to assign elements of a well-founded ordering to these goals in such a way that SLD-derivations correspond to strictly decreasing sequences. Then termination will be ensured since the ordering is well-founded. This idea is quite old and originates from mathematical logic. It has recently been applied to term rewriting systems, see for example [D]. The well-founded ordering we shall use is called the *multiset ordering*. A *multiset*, sometimes called *bag*, is an unordered sequence. Multisets are like sets, but allow multiple occurrences of identical elements. The multiset ordering over $\mathbb{N}$ is an ordering of finite multisets of natural numbers such that X is smaller than Y if X can be obtained from Y by replacing one or more elements in Y by any (finite) number of natural numbers, each of which is smaller than one of the replaced elements. See [D] for more information on the multiset ordering and its use in term rewriting systems.

Definition 2.3. An atom A is called *bounded* with respect to a level mapping $|\ |$ if $|\ |$ is bounded on the set [A] of variable-free instances of A. If A is bounded, then $\|[A]\|$ denotes the maximum that $|\ |$ takes on [A]. A goal $G = \leftarrow A_1, \ldots, A_n$ is called *bounded* if every A_i $(1 \leqslant i \leqslant n)$ is bounded. If G is bounded then $\|[G]\|$ denotes the (finite) multiset consisting of the natural numbers $\|[A_1]\|, \ldots, \|[A_n]\|$.

Example 2.4. Consider the following familiar program.

append ([], z,z)←

append ([$w|x$], y, [$w|z$]) ← *append* (x, y, z)

This program is recurrent with respect to the level mapping $|\ |$ defined by $|append(t_1, t_2, t_3)| = min(l(t_1), l(t_3))$, where $l(t)$ denotes the length of the variable-free term t as a list. More precisely: $l([]) = 0$ and $l([h\,|\,t]) = 1 + l(t)$. In general there will be more terms in the Herbrand Universe than just lists of lists. In that case we extend $|\ |$ by putting $l(f(t_1, \ldots, t_n)) = 0$ whenever f is different from the list constructing function symbol ($n \geqslant 0$). Note that a simpler definition of $|\ |$ ($= l(t_1)$ or $l(t_3)$) would also make the *append* program into a recurrent program, but would result in a smaller class of bounded goals.

LEMMA 2.5. *Let P be a logic program which is recurrent with respect to a level mapping $|\ |$. Let G be a bounded goal and G' an SLD-resolvent of G from P. Then we have: (i) The goal G' is bounded; (ii) The multiset $\|[G']\|$ is smaller than $\|[G]\|$ in the multiset ordering.*

PROOF. Let conditions be as above. As to (i), assume A_i is the selected literal in $G = \leftarrow A_1, \ldots, A_n$, and $A \leftarrow B_1, \ldots, B_k$ ($k \geqslant 0$) the program clause used. Then we have $G' = \leftarrow (A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_n)\theta$, with θ the mgu of A and A_i. We have $[A_j\theta] \subseteq [A_j]$, so $A_j\theta$ is bounded for every $1 \leqslant j \leqslant n$. Furthermore, every $C \in [B_j\theta]$ ($1 \leqslant j \leqslant k$) occurs at the right hand side of a variable-free instance of $(A \leftarrow B_1, \ldots, B_k)\theta$, so $|C| < \|[A\theta]\|$ since P is recurrent. It follows that $B_j\theta$ is bounded ($1 \leqslant j \leqslant k$), and hence G' is bounded. Moreover we have $\|[B_j\theta]\| < \|[A\theta]\| = \|[A_i\theta]\| \leqslant \|[A_i]\|$ for all $1 \leqslant j \leqslant k$. It follows that $\|[G']\|$ is smaller than $\|[G]\|$ in the multiset ordering, which proves (ii). □

COROLLARY 2.6. *Every SLD-derivation from a recurrent program starting with a bounded goal terminates.*

PROOF. Immediate, since the multiset ordering over $\mathbb{N}$ is well-founded. □

DEFINITION 2.7. A logic program P is called *terminating* if all SLD-derivations from P starting with a variable-free goal are finite.

Terminating programs have the property that SLD-trees of variable-free goals are finite. Consequently any depth-first search procedure in such an SLD-tree will always terminate. We now proceed by showing that a logic program is terminating if and only if it is recurrent.

THEOREM 2.8. *Every recurrent program is terminating.*

PROOF. By Corollary 2.6, since variable-free goals are bounded. □

The above theorem was obtained independently by Cavedon in [C], in the more general setting of recurrent programs with negation. Cavedon's proof is different and the details do not suggest the stronger Corollary 2.6. For the converse of Theorem 2.8 we need a version of the so-called Lifting Lemma.

LEMMA 2.9. *Let G be a goal, C a program clause and θ a substitution. If $G\theta$ and C have an SLD-resolvent G', then G and C have an SLD-resolvent G'' such that $G' = G''\theta'$ for some substitution θ'.*

PROOF. The proof is very similar to the induction step in the proof of [A, 3.16]. Instead of using the Variant Corollary [A, 2.9] we give the necessary renamings explicitly. Let conditions be as stated in the lemma. Then the situation can be depicted as in Figure 1, where the existence of G'' and θ' has to be established.

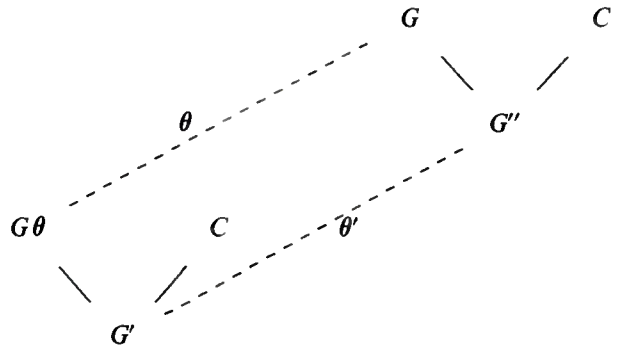

FIGURE 1

Let $G = \leftarrow A_1, \ldots, A_m$ and $C = A \leftarrow B_1, \ldots, B_n$. Without loss of generality we may assume that $G\theta$ and C have no variables in common. For convenience, let $A_1\theta$ be the selected atom in $G\theta$. Then for the mgu μ of $A_1\theta$ and A we have $G' = \leftarrow(B_1, \ldots, B_n, A_2\theta, \ldots, A_m\theta)\mu$. Let τ be a renaming for C (see [L]) such that $C\tau$ has no variables in common with either G or $G\theta$, nor does θ act on variables in $C\tau$. It follows that $A_i\theta = A_i\theta\tau^{-1}$ for all $1 \leqslant i \leqslant m$. Moreover, since θ does not act on variables occurring in $C\tau$, we have $A\tau = A\tau\theta$ and $B_j\tau = B_j\tau\theta$ for all $1 \leqslant j \leqslant n$. Hence $A\tau\theta\tau^{-1}\mu = A\tau\tau^{-1}\mu = A\mu = A_1\theta\mu = A_1\theta\tau^{-1}\mu$, i.e. $\theta\tau^{-1}\mu$ is a unifier of A_1 and $A\tau$. Let σ be the mgu of A_1 and $A\tau$, then $\theta\tau^{-1}\mu = \sigma\theta'$ for suitable θ'. We have $G'' = \leftarrow(B_1\tau, \ldots, B_n\tau, A_2, \ldots, A_m)\sigma$. Now

$$
\begin{aligned}
G''\theta' &= \leftarrow B_1\tau\sigma\theta', \ldots, B_n\tau\sigma\theta', A_2\sigma\theta', \ldots, A_m\sigma\theta' \\
&= \leftarrow B_1\tau\theta\tau^{-1}\mu, \ldots, B_n\tau\theta\tau^{-1}\mu, A_2\theta\tau^{-1}\mu, \ldots, A_m\theta\tau^{-1}\mu \\
&= \leftarrow B_1\tau\tau^{-1}\mu, \ldots, B_n\tau\tau^{-1}\mu, A_2\theta\mu, \ldots, A_m\theta\mu \\
&= \leftarrow B_1\mu, \ldots, B_n\mu, A_2\theta\mu, \ldots, A_m\theta\mu \\
&= G' \qquad \square
\end{aligned}
$$

THEOREM 2.10. *A logic program is terminating if and only if it is recurrent.*

PROOF. The if-part is Theorem 2.8. For the converse, assume P is terminating. Consider, for an arbitrary goal G, the set of *all* SLD-derivations starting from G, allowing arbitrary selection of atoms in goals. This set can be structured as a tree, which we call the *LD-tree* of G (the S is dropped since no selection rule is fixed in advance). LD-trees are finitely branching since logic programs are finite and since a goal and a program clause can have only finitely many resolvents. Since P is terminating it follows by König's Lemma that for every variable-free atom A the LD-tree of $\leftarrow A$ is finite. Hence we can define a level mapping $|\ |: B_P \rightarrow \mathbb{N}$ by taking for $|A|$ the number of nodes in the LD-tree of $\leftarrow A$. It remains to show that P is recurrent with respect to $|\ |$. Let $A\theta \leftarrow B_1\theta, \ldots, B_n\theta$ be a variable-free instance of a program clause in P. We have to show that $|A\theta| > |B_i\theta|$ for all $1 \leqslant i \leqslant n$. Consider the LD-tree of $\leftarrow A\theta$. We have that $A\theta\theta = A\theta$, so θ is a unifier of $A\theta$ and A. So for some mgu μ of $A\theta$ and A we have $\theta = \mu\theta'$ and the LD-tree of $\leftarrow B_1\mu, \ldots, B_n\mu$ is a subtree of the LD-tree of $\leftarrow A\theta$. Now consider an SLD-derivation starting with $\leftarrow B_i\theta$ $(1 \leqslant i \leqslant n)$. Since $\theta = \mu\theta'$ we can lift this derivation by applying Lemma 2.9 into a derivation starting with $\leftarrow B_i\mu$. This latter derivation can be embedded in a derivation starting with $\leftarrow B_1\mu, \ldots, B_n\mu$. It follows that the LD-tree of $\leftarrow B_i\theta$ has a smaller number of nodes than the LD-tree of $\leftarrow A\theta$, i.e. $|A| > |B_i|$ for all $1 \leqslant i \leqslant n$. $\square$

We finish this section by characterizing the class of determinate programs in terms of level mappings. Determinate programs are introduced by Blair in [B] with the following definition.

DEFINITION 2.11. A logic program P is called *determinate* if $T_P\downarrow\omega = T_P\uparrow\omega$.

Due to the well-known characterization results [A, Theorem 3.13 and Theorem 5.6] we have that a logic program P is determinate if and only if P has complementary success set and finite failure set. Consequently a breadth-first search procedure in any fair SLD-tree of a variable-free goal will always terminate. The intuition behind the relation between a determinate program P and a level mapping can be explained as follows. If P is determinate and $B \in B_P$ with $B \notin M_P$, then we have $B \notin T\downarrow(n+1)$ for some n. If $B \leftarrow A_1, \ldots, A_k$ is a variable-free instance of a clause form P, then $A_i \notin T\downarrow n$ for at least one of the A_i's. Hence the maximal level on which a failing head occurs is higher than the maximal level on which all the failing atoms of the body occur. This observation motivates the following definition.

DEFINITION 2.12. Let P be a logic program and $|\ |$ a level mapping for P. We call P *weakly recurrent with respect to* $|\ |$ if for every variable-free instance $B \leftarrow A_1, \ldots, A_n$ of a clause from P the following holds: if $B \notin M_P$, then there exists $1 \leqslant i \leqslant n$ such that $A_i \notin M_P$ and B has a higher level than A_i. P is called *weakly recurrent* if P is weakly recurrent with respect to some level mapping for P.

We proceed by showing that a logic program is determinate if and only if it is weakly recurrent. Since recurrent trivially implies weakly recurrent, it follows that every terminating program is determinate (of course this can also be seen directly). The converse does not hold: $p \leftarrow, p \leftarrow p$ form a logic program which is determinate but not terminating.

LEMMA 2.13. *Let P be weakly recurrent with respect to a level mapping $|\ |: B_P \to \mathbb{N}$. Then, for all natural numbers n, $T_P\downarrow n - T_P\uparrow\omega$ contains only atoms of level $\geqslant n$.*

PROOF. By induction on n, recalling that the immediate consequence operator T_P of P satisfies $T_P\uparrow\alpha \subseteq T_P\downarrow\beta$ for all ordinals α, β, so in particular for $\alpha = \omega, \beta = n$. Let P be weakly recurrent with respect to $|\ |$. We trivially have that $T_P\downarrow 0 - T_P\uparrow\omega$ contains only atoms of level $\geqslant 0$. Assume $T_P\downarrow n - T_P\uparrow\omega$ contains only atoms of level $\geqslant n$ and consider the case $n+1$. By definition we have $T_P\downarrow(n+1) = T_P(T_P\downarrow n)$. Using the induction hypothesis the situation can now be depicted as in Figure 2.

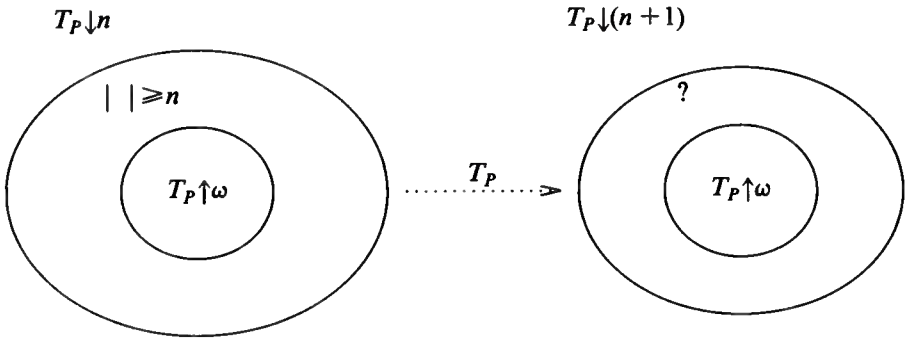

FIGURE 2

Let $B \in T_P\downarrow(n+1)$; then there exists (by the definition of T_P) a variable-free instance $B \leftarrow A_1, \ldots, A_k$ $(k \geqslant 0)$ of a clause from P such that $A_i \in T_P\downarrow n$ for all $1 \leqslant i \leqslant k$. If $B \notin T_P\uparrow\omega$, then there exists $1 \leqslant i \leqslant k$ such that $A_i \notin T_P\uparrow\omega$ and $|B| > |A_i|$, since P is weakly recurrent. By the induction hypothesis we have $|A_i| \geqslant n$, so $|B| \geqslant n+1$. This completes the induction step. □

THEOREM 2.14. *If a logic program P is weakly recurrent, then P is determinate.*

PROOF. Observe that $T_P\downarrow\omega - T_P\uparrow\omega = \bigcap_{n<\omega} (T_P\downarrow n - T_P\uparrow\omega)$. By Lemma 2.13 this latter intersection is empty. □

THEOREM 2.15. *A logic program is determinate if and only if it is weakly recurrent.*

PROOF. The if-part is Theorem 2.14. For the converse, assume P is determinate. Define a level mapping $|\ |: B_P \to \mathbb{N}$ by taking for $|A|$ the greatest k such that $A \in T_P\downarrow k$ if $A \notin T_P\uparrow\omega$, and 0 otherwise. Since $T_P\downarrow\omega = T_P\uparrow\omega$ this level mapping is well-defined. Moreover P is weakly recurrent with respect to $|\ |$. For, let $B \leftarrow A_1, \ldots, A_n$ be a variable-free instance of a clause from P. If $B \notin T_P\uparrow\omega$, then there exists $1 \leqslant i \leqslant n$ such that $A_i \notin T_P\uparrow\omega$ and $|A_i|$ is minimal. It follows that $B \in T_P\downarrow(|A_i|+1)$, so $|B| > |A_i|$. □

3. RECURSION THEORETIC CONSIDERATIONS

In this section we study the recursion theoretic properties of the class of recurrent programs. We assume the reader is familiar with the basic facts of recursion theory. We use the denotations PR, $PR(f)$, R, RE for, respectively, the classes of primitive recursive functions, functions which are primitive recursive in f, recursive functions and recursively enumerable sets. We start with establishing upper bounds on the recursion theoretic complexity of the least Herbrand model of a (weakly) recurrent program, preparing by the following lemma.

LEMMA 3.1. *Let P be recurrent with respect to a level mapping $|\ |: B_P \to \mathbb{N}$. Then, for all natural numbers n, $T_P\downarrow n - T_P\uparrow n$ contains only atoms of level $\geqslant n$.*

PROOF. By induction on n, similarly to the proof of Lemma 2.13 (see also [C]). □

LEMMA 3.2. *Let P be a logic program which is recurrent with respect to a level mapping $|\ |$. Then M_P is primitive recursive in $|\ |$.*

PROOF. Observe that $T_P\uparrow n \subseteq M_P \subseteq T_P\downarrow n$ for all n. Using Lemma 3.1 it follows that $M_P - T_P\uparrow n$ contains only atoms of level at least n. So we have:

$$A \in M_P \text{ iff } \exists k \leqslant |A|+1 \ \ A \in T_P\uparrow k .$$

After appropriate coding the sets $T_P\uparrow k$ can easily be seen to be primitive recursive. Moreover $|\ |$ may be seen as a function of natural numbers. Since the quantification in the righthand side above is bounded by $|A|+1$ it follows from well-known results in recursion theory that M_P is primitive recursive in $|\ |$. □

THEOREM 3.3. *Let P be a weakly recurrent program. Then M_P is recursive.*

PROOF. If P is weakly recurrent, then by Theorem 2.15 P is determinate, i.e. $T_P\downarrow\omega = T_P\uparrow\omega$. Generally, $T_P\uparrow\omega$ and $B_P - T_P\downarrow\omega$ are RE. So both M_P and its complement are RE. Now it follows from a well-known result in recursion theory that M_P is recursive. □

THEOREM 3.4. *If P is recurrent, then M_P is in $PR(|\ |) \cap R$.*

PROOF. By Lemma 3.2 and Theorem 2.3, since recurrent programs are weakly recurrent. □

This theorem shows that the computational power of a recurrent program largely depends on the level mapping $|\ |$. Clearly the most simple level mappings are the most appealing. For example, the

level mapping used in Example 2.4 is, after appropriate coding, primitive recursive. Consequently, by Lemma 3.2, the least Herbrand model of the *append* program is primitive recursive (of course this can also be seen directly). Although programs with primitive recursive semantics are computationally rather weak, they are not to be depreciated. Among them are many programs met in practice (such as list processing programs). Moreover we can prove the following version of Kleene's normal form theorem from recursion theory.

THEOREM 3.5. (Normal form theorem for logic programs.) *For every logic program P there exists a program P' (in which the same predicate symbols occur with incremented arities) which is recurrent with respect to a primitive recursive level mapping such that for all $p(t_1, \ldots, t_n) \in B_P$*

$$p(t_1, \ldots, t_n) \in M_P \text{ iff } \exists t' \; p(t_1, \ldots, t_n, t') \in M_{P'}.$$

PROOF. We tacitly assume that no predicate symbol occurs in P with more than one arity. First we fix the alphabet of P'. Every predicate symbol occurring in P with arity $n \geqslant 0$ occurs in P' with arity $n+1$. Furthermore, if the alphabet of P does not contain a unary function symbol s, then we add s to obtain the alphabet of P'. Now the clauses of P' are obtained by applying a simple transformation to the clauses of P, of which we shall give two typical examples:

$$p(t_1, t_2) \leftarrow$$

is transformed into

$$p(t_1, t_2, x) \leftarrow,$$

and

$$p(t_1, t_2) \leftarrow q(t_3), \; p(t_4, t_5)$$

is transformed into

$$p(t_1, t_2, s(x)) \leftarrow q(t_3, x), \; p(t_4, t_5, x),$$

where x is a variable not occurring in $t_1, \ldots, t_5$. It is immediately clear that P' thus obtained is recurrent with respect to the level mapping $|\;| : B_{P'} \to \mathbb{N}$ defined by

$$|p(t_1, \ldots, t_n, t')| = \|t'\|,$$

where $\|\;\|$ is defined by $\|s(t)\| = 1 + \|t\|$ and $\|f(t_1, \ldots, t_k)\| = 0$ whenever f differs from s $(k \geqslant 0)$. Moreover, we obviously have for all $p(t_1, \ldots, t_n) \in B_P$:

$$p(t_1, \ldots, t_n) \in M_P \text{ iff } \exists t' \; p(t_1, \ldots, t_n, t') \in M_{P'}. \quad \square$$

COROLLARY 3.6. *Let P'' be the program P' augmented with clauses $p(x_1, \ldots, x_n) \leftarrow p(x_1, \ldots, x_n, x')$ for all relations p occurring in P. Then we have $M_P = M_{P''} \cap B_P$. The program P'' may be viewed as a normal form of P. Note that P'' satisfies $T_{P''} \uparrow \omega = T_{P''} \downarrow (\omega + 1)$.*

From Theorem 3.4 we know that both $PR(|\;|)$ and R are upper bounds of the recursion theoretic complexity of the least Herbrand model of a recurrent program. We experienced considerable difficulty in establishing the exact computational power of the class of recurrent programs. These difficulties can be traced back to the fact that the class of recurrent programs (as well as the class of weakly recurrent programs) does not enjoy a natural closure property such as composition. Let us first define how a logic program computes a function.

DEFINITION 3.7. For every $n \in \mathbb{N}$, let $\bar{n}$ denote the term $s^n(0)$. A logic program P with Herbrand Universe $\{\bar{n} \mid n \in \mathbb{N}\}$ is said to compute a partial function $f : \mathbb{N}^k \to \mathbb{N}$ if for some predicate symbol p we have

$$f(n_1, \ldots, n_k) = n \text{ iff } p(\bar{n}_1, \ldots, \bar{n}_k, \bar{n}) \in M_P$$

for all $n_1, \ldots, n_k, n \in \mathbb{N}$.

The technical obstacle mentioned above becomes apparent when one tries to prove that the class of functions which can be computed by recurrent programs is closed under composition. If $f(x) = g(h(x))$, then it seems natural to add the clause

$$p_f(x, y) \leftarrow p_h(x, z), p_g(z, y)$$

to (disjoint) recurrent programs computing g and h in order to obtain a program that computes f. However, this latter program will in general not be recurrent, due to the presence of the variable z in the body of the above clause. Consider, for example, $p_h(x, s(x)) \leftarrow$ and $p_g(0, s(0)) \leftarrow$, $p_g(s(x), y) \leftarrow p_g(x, y)$, with p_f as defined above. Then $\leftarrow p_f(0, s(0))$ heads an infinite (right-most) SLD-derivation. The solution turned out to be the choice of the right machine model, namely the register machine (see [S1]).

DEFINITION 3.8. A *register machine program* is a finite sequence $I_1, \ldots, I_n$ of instructions which operate on registers $x_1, \ldots, x_m$, where each instruction is of one of the following two forms (with $1 \leq i \leq m$, $1 \leq j \leq n+1$).

$$x_i := x_i + 1$$

$$IF\ x_i \neq 0\ THEN\ x_i := x_i - 1\ AND\ GOTO\ j$$

The program is completed with a halt instruction I_{n+1}. Execution of a register machine program with respect to given contents $n_1, \ldots, n_m \in \mathbb{N}$ of the registers $x_1, \ldots, x_m$ starts from I_1, executing the instructions in the obvious sequential way, and terminates when the halt instruction I_{n+1} is reached. A register machine program is said to compute a (partial) function $f : \mathbb{N}^k \to \mathbb{N}$ $(k \leq m)$ if, for all $n_1, \ldots, n_k \in \mathbb{N}$, the execution of the register machine program with respect to contents $n_1, \ldots, n_k, 0, \ldots, 0$ of the registers terminates with value $f(n_1, \ldots, n_k)$ in register x_1 (if this function value is not defined, then the execution is not allowed to terminate).

A classical result from [S1] states that every partial recursive function can be computed by a register machine program. This same result can be obtained for logic programs by transforming register machine programs into logic programs. This can be done in a very natural way, as shown in [S]. Every instruction corresponds to the definition of a predicate symbol, every register corresponds to an argument of these predicate symbols. All predicate symbols have one more argument through which the function value is passed. The predicate symbol p_k corresponding to the instruction I_k has one of the following two definitions, corresponding to the possible forms of the instruction as displayed above (the second definition consists of two clauses).

$$p_k(x_1, \ldots, x_i, \ldots, x_m, y) \leftarrow p_{k+1}(x_1, \ldots, s(x_i), \ldots, x_m, y)$$

$$p_k(x_1, \ldots, s(x_i), \ldots, x_m, y) \leftarrow p_j(x_1, \ldots, x_i, \ldots, x_m, y)$$
$$p_k(x_1, \ldots, 0, \ldots, x_m, y) \leftarrow p_{k+1}(x_1, \ldots, 0, \ldots, x_m, y)$$

The halt instruction corresponds to the following program clause, by which the function value is transported from register x_1 to the last argument of the predicate symbols.

$$p_{n+1}(x_1, \ldots, x_m, x_1) \leftarrow$$

If we finally add the clause

$$p(x_1, \ldots, x_l, y) \leftarrow p_1(x_1, \ldots, x_l, 0, \ldots, 0, y)$$

to the logic program corresponding to a register machine program, then it is easy to see that both compute the same partial function. It is also easy to see that the SLD-trees corresponding to register

machine computations consist of one single path, and that this path is finite if and only if the computation terminates. However, this does not imply that the logic program corresponding to a register machine program which computes a total recursive function is terminating. For example, it is not guaranteed that the goal $\leftarrow p_7(0,\ldots,0,0)$ terminates (the malicious programmer could have taken *GOTO* 7 for I_7, taking care that any normal execution of his program never falls into this trap). Evidently we need a stronger result than can be extracted from [S1]. Goals of the form $\leftarrow p_k(\overline{n}_1,\ldots,\overline{n}_m,\overline{n})$ correspond to starting the execution of the register machine program at the k-th instruction, with the registers initialized on $n_1,\ldots,n_m$. Hence the result we actually need is that every total recursive function can be computed by a register machine program which always halts, irrespective of the initialization of the registers and of the instruction at which the execution starts. This result was proved by Shepherdson in [S2]. An analogous result for Turing machines has been proved by Davis [Da, Theorem V.3]. We are now in a position to state and prove the main result of this paper.

THEOREM 3.9. *Every total recursive function can be computed by a recurrent program.*

PROOF. By Theorem 2.10 and [S2, Theorem 4], using the transformation of register machine programs to logic programs as described above. Note that the level of a closed atom is independent of its last argument. □

To illustrate the use of recurrent programs in improving the termination behaviour of Prolog programs we show how the familiar Ackermann function can be computed by a recurrent program. The underlying idea is that of using the arguments of a predicate as a depth-bound. This technique may be applicable in many practical programs such as list processing programs, where the depth-bounds are often even easier to find than in the example below.

EXAMPLE 3.12. The Ackermann function $A:\mathbb{N}\times\mathbb{N}\rightarrow\mathbb{N}$ is a recursive function which is not primitive recursive, and is defined as follows:

$$A(0,n) = n+1;$$

$$A(i+1,0) = A(i,1);$$

$$A(i+1,n+1) = A(i,A(i+1,n)).$$

A straightforward translation of this definition suggests the following logic program.

$$p(0,y,s(y))\leftarrow$$

$$p(s(x),0,z)\leftarrow p(x,\overline{1},z)$$

$$p(s(x),s(y),z)\leftarrow p(s(x),y,z'),\ p(x,z',z)$$

However, this program is not recurrent since it is not terminating. For example, we have $p(\overline{2},\overline{2},0)\leftarrow p(\overline{2},\overline{1},z), p(\overline{1},z,0)$, and $\leftarrow p(\overline{1},z,0)$ heads an infinite left-most SLD-derivation by applying the third program clause again and again. By using the ad hoc observations that the Ackermann function is monotonic (so $A(i+1,n)<A(i+1,n+1)$) and that the function value constitutes (in some sense) a logarithmic upper bound on the length of the computation, we devised the following logic program P.

$$p(0,y,s(y),w)\leftarrow$$

$$p(s(x),0,z,w)\leftarrow p(x,\overline{1},z,w)$$

$$p(s(x),s(y),z,s(w))\leftarrow p(s(x),y,z',w),p(x,z',z,s(w))$$

This program is recurrent with respect to the level mapping defined by $|p(t_1,t_2,t_3,t_4)| = \|t_1\| + \|t_4\|$,

where $\|\overline{n}\| = n$. With some technical effort (notably transfinite induction upto ω^2) one proves that for all $n,i,m \in \mathbb{N}$

$$m = A(n,i) \text{ if and only if } p(\overline{n},\overline{i},\overline{m},\overline{m}) \in M_P.$$

Hence the Ackermann function is computed by the recurrent program obtained by augmenting P with the clause

$$p_A(x,y,z) \leftarrow p(x,y,z,z),$$

where the level mapping is extended by putting $|p_A(t_1,t_2,t_3)| = 1+|p(t_1,t_2,t_3,t_3)|$. Since P is recurrent, bounded goals, such as $\leftarrow p_A(\overline{1},z,0)$, are correctly evaluated by a depth-first search procedure in the SLD-tree, no matter the ordering of the program clauses and the selection of atoms in goals. Note that the Prolog evaluation of a goal such as $\leftarrow p_A(x,y,\overline{13})$, which is not bounded, has also improved. Due to the left-most selection rule of Prolog all five solutions to this goal are successively found, after which the evaluation correctly terminates. The naive implementation of the Ackermann function does not evaluate the goal $\leftarrow p(x,y,\overline{13})$ correctly: after two solutions the evaluator hits an infinite branch in the SLD-tree.

It may seem paradoxical that the level mapping above is primitive recursive, so that P has primitive recursive semantics by Lemma 3.2. However, it so happens that some non primitive recursive functions do have primitive recursive graphs. Note that the construction of P bears some similarity to the proof of Theorem 3.5, the normal form theorem for logic programs.

ACKNOWLEDGEMENTS. I would like to thank Krzysztof Apt, Roland Bol, Jan Heering, Jan Willem Klop, Hans Mulder, Dirk Roorda and Professor J.C. Shepherdson for helpful advise and stimulating discussions on this paper.

REFERENCES

[A] K.R. APT, *Introduction to Logic Programming*, Report CS-R8826, Centre for Mathematics and Computer Science, Amsterdam, 1988. To appear in: J. VAN LEEUWEN (editor), *Handbook of Theoretical Computer Science*, North Holland, Amsterdam.

[B] H. BLAIR, *Decidability in the Herbrand Base*, manuscript (presented at the Workshop on Foundations of Deductive Databases and Logic Programming, Washington D.C., August 1986).

[C] L. CAVEDON, *Continuity, Consistency, and Completeness Properties for Logic Programs*, to appear in the Proceedings of the Fifth International conference on Logic Programming, Lisbon, 1989.

[Cl] K.L. CLARK, *Negation as failure*, in: H. GALLAIRE, J. MINKER (editors), *Logic and Data Bases*, Plenum Press, New York, 1978, pp. 293-322.

[D] N. DERSHOWITZ, *Termination of Rewriting*, Journal of Symbolic Computation 3, pp. 69-116, 1987.

[Da] M. DAVIS, *A note on Universal Turing Machines*, In: J. MCCARTHY, C.J. SHANNON (editors), *Automata studies*, Princeton University Press, Princeton, pp. 167-175, 1956.

[L] J.W. LLOYD, *Foundations of Logic Programming*, Second Edition, Springer-Verlag, Berlin, 1987.

[S] J.C. SHEPHERDSON, *Undecidability of Horn Clause Logic and Pure Prolog*, unpublished manuscript, 1985.

[Sh] J.R. SHOENFIELD, *Mathematical Logic*, Addison-Wesley, Reading, Mass., 1967.

[S1] J.C. SHEPHERDSON, H.E. STURGIS, *Computability of Recursive Functions*, Journal of the ACM 10, pp. 217-255, 1963.

[S2] J.C. SHEPHERDSON, *Machine configuration and word problems of given degree of unsolvability*, Zeitsch. f. math. Logik und Grundlagen d. Math. 11, pp. 149-175, 1965.