Logic Programs for Primitive Recursive Sets

URS-MARTIN KÜNZI, *Institut für Informatik und angewandte Mathematik, Universität Bern, Länggassstr. 51, 3012 Bern, Switzerland.*
E-mail: kuenzi@iam.unibe.ch

Abstract

Meyer and Ritchie have previously given a description of primitive recursive functions by loop-programs. In this paper a class of logic programs is described which computes the primitive recursive sets on Herbrand universes. Furthermore, an internal description of primitive recursive functions and sets on Herbrand universes is given.

Keywords. Logic programming, Prolog, primitive recursive sets

1 Introduction

Let L be a finite set of constants and function symbols, containing at least one constant and one function symbol, and let $\mathfrak{H}$ be the *Herbrand structure* of L (the L-structure with the set of closed L-terms as underlying set H; this set is the *Herbrand universe* of L). A subset of H^n is *primitive recursive* iff its characteristic function is primitive recursive with respect to a fixed Gödelization of H. In the next section, an intrinsic description of primitive recursive predicates is given (that does not depend on a Gödelization). An *L-program* is a definite program P [3, Chapter 2], containing only constants and function symbols from L, together with a distinguished predicate p. The *set computed by* P is the set $\{(t_1, \ldots, t_n) \in H^n \mid \mathrm{P} \vdash \mathrm{p}(t_1, \ldots, t_n)\}$. In this article we describe syntactically a class of L-programs, computing precisely the primitive recursive subsets of H^n.

A program clause is called *closed* iff its body contains only variables also contained in its head. For an L-term t with variables $\mathtt{X}_1, \ldots, \mathtt{X}_n$ we define $|t|$ to be the linear polynomial with coefficients in $\mathbf{N}$ and with variables $|\mathtt{X}_1|, \ldots, |\mathtt{X}_n|$, recursively defined by $|c| := 1$ for all constants and $|f(t_1, \ldots, t_n)| := 1 + |t_1| + \cdots + |t_n|$. If t is a variable-free term, then $|t|$ is the number of constants and function symbols contained in t. We order the polynomials by means of

$$f(\bar{\mathtt{X}}) \leq g(\bar{\mathtt{X}}) \ :\Longleftrightarrow\ \forall \bar{m} \in (\mathbf{N} \setminus \{0\})^n \ f(\bar{m}) \leq g(\bar{m}) \tag{1.1}$$

and

$$f(\bar{\mathtt{X}}) < g(\bar{\mathtt{X}}) \ :\Longleftrightarrow\ \forall \bar{m} \in (\mathbf{N} \setminus \{0\})^n \ f(\bar{m}) < g(\bar{m}) \tag{1.2}$$

So given $f(\bar{\mathtt{X}}) := a + b_1|\mathtt{X}_1| + \cdots + b_n|\mathtt{X}_n|$ and $g(\bar{\mathtt{X}}) := c + d_1|\mathtt{X}_1| + \cdots + d_n|\mathtt{X}_n|$, we have

$$f(\bar{\mathtt{X}}) \leq g(\bar{\mathtt{X}}) \iff \forall i(b_i \leq d_i) \wedge a + b_1 + \cdots + b_n \leq c + d_1 + \cdots + d_n \tag{1.3}$$

and

$$f(\bar{\mathtt{X}}) < g(\bar{\mathtt{X}}) \iff \forall i(b_i \leq d_i) \wedge a + b_1 + \cdots + b_n < c + d_1 + \cdots + d_n \tag{1.4}$$

Given terms $s(\mathtt{X}_1,\ldots,\mathtt{X}_n)$ and $t(\mathtt{X}_1,\ldots,\mathtt{X}_n)$ we have $|s| \le |t|$ (respectively $|s| < |t|$) iff for all tupels of variable-free terms $(u_1,\ldots,u_n)$ it holds $|s(u_1,\ldots,u_n)| \le |t(u_1,\ldots,u_n)|$ (respectively $|s(u_1,\ldots,u_n)| < |t(u_1,\ldots,u_n)|$). n-tupels of terms are ordered (lexicographically) by

$$(t_1,\ldots,t_n) \prec (s_1,\ldots,s_n) \iff \exists i \le n\,(\forall j < i(|t_j| = |s_j|) \wedge |t_i| < |s_i|). \tag{1.5}$$

A L-program P is called *tame* iff it satisfies the following conditions.

(i) All clauses of P are closed.

(ii) The predicates of P can be linearly ordered in such a way that p is the greatest predicate, and for every clause of P, the predicate of its head is greater than or equal to every predicate of its body.

(iii) If a clause of P with head $\mathtt{r}(t_1,\ldots,t_n)$ contains in its body the formula $\mathtt{r}(s_1,\ldots,s_n)$, then $(s_1,\ldots,s_n) \prec (t_1,\ldots,t_n)$

We now choose a fixed order satisfying (ii) Then we can order atomic formulas by

$$\mathtt{q}(t_1,\ldots,t_n) \prec \mathtt{r}(s_1,\ \ldots,s_m) \iff \begin{cases} \mathtt{q} < \mathtt{r} & \text{or} \\ \mathtt{q} = \mathtt{r} \ \&\ (t_1,\ldots,t_n) \prec (s_1,\ldots,s_n). \end{cases} \tag{1.6}$$

This is a well-founded ordering of the variable-free atomic formulas. If we resolve a variable-free atomic formula Φ with a clause of a tame program, the resolvent consists of variable-free atoms smaller than Φ. It follows that tame programs, applied to variable-free atomic formulas, always stop.

We shall show that the tame programs compute primitive recursive sets and that every primitive recursive set is computed by a tame program. We do not claim that the class of tame programs is adequate for practical purposes. Condition (i) is very restrictive. Our aim was to find a class of programs with a structure as simple as possible, but strong enough to compute primitive recursive sets. (A more practice-oriented approach is done by Stroetmann in [6] and [7], see also Section 7). Natural tame programs exist, e.g. for list operations like membership, concatenation, inversion etc., but the following program, expressing that one list is a permutation of another one, seems already to be a little artificial (we use the Prolog notation with square brackets for lists):

$$\mathtt{perm}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{permh}(\mathtt{X},[],\mathtt{Y},[]). \tag{1.7}$$

$$\mathtt{permh}([\mathtt{X}|\mathtt{Y}],\mathtt{Z},[\mathtt{X}|\mathtt{U}],\mathtt{V}) \leftarrow \mathtt{permh}(\mathtt{Y},[[]],\mathtt{U},\mathtt{V}). \tag{1.8}$$

$$\mathtt{permh}(\mathtt{Y},\mathtt{Z},[\mathtt{X}|\mathtt{U}],\mathtt{V}) \leftarrow \mathtt{permh}(\mathtt{Y},\mathtt{Z},\mathtt{U},[\mathtt{X}|\mathtt{V}]). \tag{1.9}$$

$$\mathtt{permh}(\mathtt{Y},[[]],[],\mathtt{V}) \leftarrow \mathtt{permh}(\mathtt{Y},[],\mathtt{V},[]). \tag{1.10}$$

$$\mathtt{permh}([],\mathtt{V},[],[]). \tag{1.11}$$

2 Primitive recursion on a Herbrand universe

Before we continue our investigations on tame programs, we want to give an internal description of primitive recursion on Herbrand universes. This approach follows [2]. With respect to the fixed language L, let PR_L be the smallest set of functions $H^n \to H$ (for all $n \in \mathbb{N}$) satisfying the following conditions.

(iv) Every function $\tilde{t} : H^n \to H : (a_1,\ldots,a_n) \mapsto t(a_1,\ldots,a_n)$, with $t(X_1,\ldots,X_n)$ an arbitrary L-term, is in PR_L.

(v) PR_L is closed under compositions of functions, i.e. if $\varphi(x_1, \ldots, x_n)$ and $\psi_i(\bar{y})$ are in PR_L, then $\varphi(\psi_1(\bar{y}), \ldots, \psi_n(\bar{y}))$ is also in PR_L.

(vi) For every constant c and for every n-ary function symbol f, let $\varphi_c(\bar{y})$ and $\varphi_f(x_1, \ldots, x_n, z_1, \ldots, z_n, \bar{y})$ be functions of PR_L. Then the function φ, defined by

$$\varphi(c, \bar{y}) := \varphi_c(\bar{y}) \tag{2.1}$$

$$\varphi(f(x_1, \ldots, x_n), \bar{y}) := \varphi_f(x_1, \ldots, x_n, \varphi(x_1, \bar{y}), \ldots, \varphi(x_n, \bar{y}), \bar{y}), \tag{2.2}$$

is in PR_L.

By (iv), PR_L contains all projection functions (take $t(\mathtt{X}_1, \ldots, \mathtt{X}_n) := \mathtt{X}_i$), so the recursion schema (vi) holds for all arguments (and not only for the first one).

Let $\gamma : H \to \mathbf{N}$ be a primitive recursive Gödelization of H, i.e. an injection with the following properties.

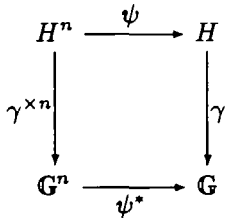
(vii) γ is increasing, i.e. if s is a subterm of t then $\gamma(s) < \gamma(t)$.

(viii) If f is a function symbol with arity n_f then there is a primitive recursive function $\gamma_f : \mathbf{N}^{n_f} \to \mathbf{N}$ such that $\gamma(f(t_1, \ldots, t_{n_f})) = \gamma_f(\gamma(t_1), \ldots, \gamma(t_{n_f}))$ for all t_i.

(ix) Let $f_1, \ldots, f_r$ be an enumeration of the function symbols and constants of L (where the constants are thought of as 0-ary function symbols), and let n_i be the arity of f_i. Then the map $\pi_0 : \mathbf{N} \to \mathbf{N}$, defined by $\pi_0(\gamma(f_i(t_1, \ldots, t_{n_i}))) := i$ and $\pi_0(x) := 0$ if $x \notin \mathrm{Im}(\gamma)$, is primitive recursive.

(x) For any natural number $i > 0$ which does not exceed all arities of function symbols of L, the map $\pi_i : \mathbf{N} \to \mathbf{N}$, defined by $\pi_i(\gamma(f_j(t_1, \ldots, t_{n_j}))) := \gamma(t_i)$ if $i \leq n_j$, else $\pi_i(x) := 0$, is primitive recursive.

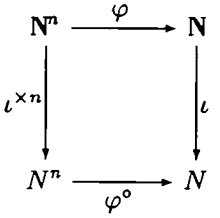
We remark that (ix) and (x) are consequences of the other conditions. Let $\mathbb{G} \subset \mathbf{N}$ be the image of γ, and let $\gamma^{\times n} : H^n \to \mathbb{G}^n$ be the map with $(t_1, \ldots, t_n) \mapsto (\gamma(t_1), \ldots, \gamma(t_n))$. Then to any map $\psi : H^n \to H$ corresponds a unique map $\psi^* : \mathbb{G}^n \to \mathbb{G}$ for which the following diagram commutes:



The function $\varphi : H^n \to H$ is called *primitive recursive* iff φ^* is the restriction of a primitive recursive function; a subset $U \subset H^n$ is called *primitive recursive* iff $\gamma^{\times n}(U)$ is a primitive recursive set. Exploiting (viii) it is easy to see by course-of-values-recursion that the functions of PR_L are primitive recursive. We show that the contrary is also true.

Now for the whole paper we fix a constant 0 and a 1-ary function-symbol $'$; if there is no 1-ary function-symbol in L, then we take x' to be an abbreviation for $g(x, 0, \ldots, 0)$, where g is a fixed function-symbol. Then $\iota : \mathbf{N} \to H$, defined by $\iota(0) := 0$ and $\iota(n+1) := \iota(n)'$, is an injection. Let N be the image of ι. Again given any function $\varphi : \mathbf{N} \to \mathbf{N}$, there is a unique function

$\varphi^\circ : N^n \to N$, for which the following diagram commutes:



It follows immediately from the definitions that if φ is primitive recursive, then φ° is the restriction of a function of PR_L. Let $f(t_1, \ldots, t_n)$ be a term. Then

$$\begin{aligned} \iota\gamma(f(t_1,\ldots,t_n)) &= \iota\gamma_f(\gamma(t_1),\ldots,\gamma(t_n)) \\ &= \iota\gamma_f \circ (\iota^{\times n})^{-1}(\iota\gamma(t_1),\ldots,\iota\gamma(t_n)) \\ &= \gamma_f^\circ(\iota\gamma(t_1),\ldots,\iota\gamma(t_n)) \end{aligned}$$

so that the bijection $\iota\gamma : H \to N$ is in PR_L. In order to show that the inverse of this bijection is the restriction of a map from PR_L we need a lemma:

LEMMA 2.1 (Definition by cases)
Let $t_1, \ldots, t_{m-1}$ be distinct terms and let $\varphi_1(\bar{y}), \ldots, \varphi_{m-1}(\bar{y})$ and $\varphi_m(x, \bar{y})$ be functions of PR_L. Then the function ψ, defined by

$$\psi(x,\bar{y}) := \begin{cases} \varphi_i(\bar{y}) & \text{if } x = t_i \\ \varphi_m(x,\bar{y}) & \text{else} \end{cases} \tag{2.3}$$

is also in PR_L.

PROOF. We may assume that $m = 2$. (The general case follows by an obvious induction.) It follows by (vi), that in PR_L there is a function $\chi(z, x, \bar{y})$ with $\chi(0, x, \bar{y}) = \varphi_1(\bar{y})$ and $\chi(0', x, \bar{y}) = \varphi_2(x, \bar{y})$. Now let $\rho : N \to N$ be the restriction of a function of PR_L with $\rho(\iota\gamma(t_1)) := 0$ and $\rho(x) := 0'$ for $x \in N$ and $x \neq \iota\gamma(t_1)$. We get $\psi(x, \bar{y}) = \chi(\rho\iota\gamma(x), x, \bar{y})$, so that ψ is in PR_L. ∎

LEMMA 2.2
There is a function $\eta : H \to H$ in PR_L such that $\eta | N$ is the inverse of $\iota\gamma$.

PROOF. First we assume that L contains a function symbol q with arity greater than 1. Then we may assume that there is a 2-ary function symbol p; (otherwise we consider $p(x, y) := q(x, y, 0, \ldots, 0)$ as an abbreviation). We can use p to encode lists: $p(s, t)$ encodes the list with head s and tail t. There is a 2-ary function $[x]_i$ in PR_L with $[p(x, y)]_0 := x$ and $[p(x, y)]_{i'} := [y]_i$. Let f_i, π_i and n_i be as in (ix) and (x). Define $s - t$ to be the difference between s and t when they are both in N and $s \geq t$, and $s - t := 0$ otherwise. Then $-$ is in PR_L. By Lemma 2.1, the function $\lambda(i, x_1, \ldots, x_n, a, l) := f_i([l]_{a-x_1}, \ldots, [l]_{a-x_{n_i}})$ is in PR_L. Now there is a function μ in PR_L with $\mu(0) := 0$ and $\mu(x') := p(\lambda(\pi_0(x), \pi_1(x), \ldots \pi_n(x), x, \mu(x)), \mu(x))$. If x is in N, then $\mu(x)$ can be considered to be the list $((\iota\gamma)^{-1}(x - 1), \ldots, (\iota\gamma)^{-1}(0))$, so we may set $\eta(x) := [\mu(x')]_0$ and the lemma is proved for this case.

We now assume that every function-symbols of L is 1-ary. There is a primitive recursive function $\rho : \mathbf{N}^* \to \mathbf{N}$ with $\rho(i, \gamma(f_{m_j} \ldots f_{m_1}(f_{m_0}))) := m_i$ if $i \leq m_j$, and $\rho(i, x) := 0$ otherwise. Now use Lemma 2.1 to define $\xi : H^2 \to H$ by $\xi(0, r) := f_{\rho(0,r)}$ and $\xi(i', r) := f_{\rho(i',r)}(\xi(i, r))$ if $\rho(i', r) \neq 0$ and $\xi(i', r) := \xi(i, r)$ otherwise. We set $\eta(r) := \xi(r, r)$. ∎

We now obtain the desired internal description of primitive recursive functions and sets:

THEOREM 2.3
A function $\varphi : H^n \to H$ is primitive recursive iff it is in PR_L. A subset $U \subset H^n$ is a primitive recursive set iff there is a function $\xi : H^n \to H$ in PR_L such that $U = \{\bar{x} \in H \mid \xi(\bar{x}) = 0\}$.

PROOF. We have already mentioned that every function of PR_L is primitive recursive. We now assume that φ^* is the restriction of a primitive recursive function. Then, by Lemma 2.2, we get that $\varphi = \gamma^{-1}\varphi^*\gamma = \gamma^{-1}\iota^{-1}\varphi^{*\circ}\iota\gamma = \eta\varphi^{*\circ}(\iota\gamma)$ is in PR_L. The second part of the theorem follows immediately from the first part. ∎

3 Ackermann predicates

In this section L consists of the constant 0 and the 1-ary function-symbol $'$. We identify N and N with ι, so that $H = N = \mathrm{N}$. The programs A_m consists of the following clauses.

$$\mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_2, \mathtt{X}_1', \mathtt{Z}) \leftarrow \mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_2, \mathtt{X}_1, \mathtt{Z}'). \tag{3.1}$$

$$\mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_3, \mathtt{X}_2', 0, \mathtt{Z}) \leftarrow \mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_3, \mathtt{X}_2, \mathtt{Z}', \mathtt{Z}'). \tag{3.2}$$

$$\vdots$$

$$\mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_{\imath+1}, \mathtt{X}_\imath', 0, \ldots, 0, \mathtt{Z}) \leftarrow \mathsf{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_\imath, \mathtt{Z}', 0, \ldots, 0, \mathtt{Z}'). \tag{3.3}$$

$$\vdots$$

$$\mathsf{a}_m(\mathtt{X}_m', 0, \ldots, 0, \mathtt{Z}) \leftarrow \mathsf{a}_m(\mathtt{X}_m, \mathtt{Z}', 0, \ldots, 0, \mathtt{Z}'). \tag{3.4}$$

These programs are tame. For every variable-free tuple $(t_m, \ldots, t_0)$ there is precisely one variable-free term u with $\mathsf{A}_m \vdash \mathsf{a}_m(t_m, \ldots, t_0) \leftarrow \mathsf{a}_m(0, \ldots, 0, u)$. We define $\alpha_m(t_m, \ldots, t_0) := u$. The functions α_m satisfy the following equations.

$$\alpha_m(0, \ldots, 0, x) = x \tag{3.5}$$

$$\alpha_m(x_m, \ldots, x_2, x_1 + 1, z) = \alpha_m(x_m, \ldots, x_2, x_1, z + 1) \tag{3.6}$$

$$\alpha_m(x_m, \ldots, x_3, x_2 + 1, 0, z) = \alpha_m(x_m, \ldots, x_3, x_2, z + 1, z + 1) \tag{3.7}$$

$$\vdots$$

$$\alpha_m(x_m, \ldots, x_{\imath+1}, x_\imath + 1, 0, \ldots, 0, z) = \alpha_m(x_m, \ldots, x_\imath, z + 1, 0, \ldots, 0, z + 1) \tag{3.8}$$

$$\vdots$$

$$\alpha_m(x_m + 1, 0, \ldots, 0, z) = \alpha_m(x_m, z + 1, 0, \ldots, 0, z + 1). \tag{3.9}$$

The functions α_m are uniquely defined by these equations. It follows that they are strictly increasing in all arguments. The following equations also hold:

$$\alpha_1(x, y) = x + y \tag{3.10}$$

$$\alpha_2(x, y, z) = 2^x(y + z + 2) - 2 \tag{3.11}$$

$$\alpha_m(0, x_{m-1}, \ldots, x_0) = \alpha_{m-1}(x_{m-1}, \ldots, x_0) \tag{3.12}$$

$$\alpha_m(x_m, \ldots, x_0) = \alpha_m(x_m, \ldots, x_{\imath+1}, 0, \ldots, 0, \alpha_\imath(x_\imath, \ldots, x_0)). \tag{3.13}$$

The equation (3.13) follows by transfinite induction over the lexicographic ordering $\prec$ on the tupels $(x_\imath, \ldots, x_1)$ (defined as in (1.5)): If $x_\imath = \ldots = x_1 = 0$ then there is nothing to

prove. Otherwise there is a $j \in \{1, \ldots, i\}$ with $x_j \neq 0$ and $x_{j-1} = \ldots = x_1 = 0$. We get $\alpha_m(x_m, \ldots, x_0) = \alpha_m(x_m, \ldots, x_{j+1}, x_j - 1, x_0 + 1, 0, \ldots, 0, x_0 + 1) = \alpha_m(x_m, \ldots, x_{i+1}, 0, \ldots, 0, \alpha_i(x_i, \ldots, x_{j+1}, x_j - 1, x_0 + 1, 0, \ldots, 0, x_0 + 1)) = \alpha_m(x_m, \ldots, x_{i+1}, 0, \ldots, 0, \alpha_i(x_i, \ldots, x_0))$ (the first and the third equalitiy follow by (3.8), the second follows from the induction hypothesis).

Let $\beta_m(x, y) := \alpha_m(x, 0, \ldots, 0, y) = \alpha_m(x, 0, \ldots, 0, y, 0)$. Then

$$\beta_1(x, y) = x + y \tag{3.14}$$

$$\beta_m(0, y) = y \tag{3.15}$$

$$\beta_m(x + 1, y) = \beta_m(x, \beta_{m-1}(y + 1, y + 1)) \tag{3.16}$$

If we set $\delta(y) := \beta_{m-1}(y+1, y+1)$, then (3.16) becomes $\beta_m(x+1, y) = \beta_m(x, \delta(y))$, hence $\beta_m(x, y) = \delta^x(y)$. It follows that the functions β_m are primitive recursive. It follows from $\alpha_m(x_m, \ldots, x_0) = \beta_m(x_m, \alpha_{m-1}(x_{m-1}, \ldots, x_0))$ that the α_m are primitive recursive too.

$\beta_m(x, y)$ is a variant of the Ackermann function. In fact it majorizes the Ackermann function. So given any primitive recursive function $\varphi : \mathbf{N} \to \mathbf{N}$ there is an m with $\beta_m(1, x) > \varphi(x)$ for every $x \in \mathbf{N}$. Also, for any primitive recursive function $\varphi : \mathbf{N}^n \to \mathbf{N}$ there is an m with $\alpha_m(1, 0, \ldots, 0, x_1, \ldots, x_n) > \varphi(x_1, \ldots, x_n)$.

4 'Tame' implies 'primitive recursive'

We first show that it suffices to consider programs with only one predicate. This is in fact nothing else than a variant of the transitivity theorem for inductive definitions [4, theorem 1C.3]. Let P be a L-program with predicates $\mathrm{p}_1, \ldots, \mathrm{p}_s = \mathrm{p}$, where the predicates are ordered by $\mathrm{p}_i < \mathrm{p}_j \;:\Longleftrightarrow\; i < j$. We choose a constant $0 \in L$ and variable-free terms $l_1, \ldots, l_s$ with $|l_1| < \cdots < |l_s|$. Let q be a new predicate whose arity is greater than all the arities of the predicates of P. Given an atomic formula $\Pi = \mathrm{p}_i(t_1, \ldots, t_{n_i})$ we construct a formula $\tilde{\Pi} := \mathrm{q}(l_i; t_1, \ldots, t_{n_i}, 0, \ldots, 0)$. We attach to P a program $\tilde{\mathrm{P}}$ by replacing all atomic formulas Π contained in P by $\tilde{\Pi}$. Then a tupel $(t_1, \ldots, t_n) \in H^n$ is in the set computed by P iff $(l_s; t_1, \ldots, t_n, 0, \ldots, 0)$ is in the set computed by $\tilde{\mathrm{P}}$. If P is tame, $\tilde{\mathrm{P}}$ is tame also. So we get:

LEMMA 4.1
Every subset of H^n computed by a tame program is a section of a set computed by a tame program containing only one predicate.

We really do need sections: If $L = \{0, '\}$, then the subsets of $H = N$ computable by a tame program with only one 1-ary predicate are the sets definable in the Pressburger arithmetic. Let Φ be an atomic formula. A *proof tree for* Φ (with respect to P) is a finite tree $\mathcal{F}$ with the following properties.

(xi) The nodes of $\mathcal{F}$ are atomic formulas.

(xii) The root of $\mathcal{F}$ is Φ.

(xiii) If Ψ is a node of $\mathcal{F}$ and if $\Xi_1, \ldots, \Xi_r$ are its successors, then $\Psi \leftarrow \Xi_1 \wedge \cdots \wedge \Xi_r$ is a substitution instance of a clause of P.

Let $\mathcal{B}$ be the set of all variable-free atomic formulas and $\mathbb{T}$ the finite set of all proof trees with respect to P. The map $\Theta_{\mathrm{P}} : \mathcal{B} \times \mathbf{N} \to \mathcal{P}_\omega(\mathbb{T})$, moving a pair (Ψ, m) to the set of all proof trees for Ψ with a depth not greater than m, is primitive recursive (with respect to some coding).

LEMMA 4.2
Tame programs compute primitive recursive sets.

PROOF. Let Q be a tame program. By Lemma 4.1, we may assume that Q contains only one predicate q, because sections of primitive recursive sets are primitive recursive themselves. Choose $p \in \mathbf{N}$ such that for any term t occurring in the body of some clause of Q, every coefficient of $|t|$ is smaller than p. Let n be the arity of q. Now for $(t_1, \ldots, t_n) \in H^n$, we set $\|t_1, \ldots, t_n\| := \alpha_{2n+1}(|t_1|, \ldots, |t_n|, |t_1|, \ldots, |t_n|, p, 0) \in \mathbf{N}$. Let $\mathrm{q}(t_1, \ldots, t_n)$ be a node of a proof tree and $\mathrm{q}(s_1, \ldots, s_n)$ a successor of this node. For each $\imath$, we have $p + p\sum_{j=1}^{n} |t_j| > |s_\imath|$; as Q is tame there is a $m \leq n$ such that $|t_\imath| \geq |s_\imath|$ for $\imath < m$ and $|t_m| > |s_m|$. It follows that

$$\begin{aligned}
\|t_1, \ldots, t_n\| &= \alpha_{2n+1}(|t_1|, \ldots, |t_n|, |t_1|, \ldots, |t_n|, p, 0) \\
&\geq \alpha_{2n+1}(|t_1|, \ldots, |t_n|, 0, \ldots, 0, \sum_{j=1}^{n} |t_j|, p, 0) \\
&\geq \alpha_{2n+1}(|t_1|, \ldots, |t_n|, 0, \ldots, 0, p + p\sum_{j=1}^{n} |t_j|) \\
&= \alpha_{2n+1}(|t_1|, \ldots, |t_{m-1}|, |t_m| - 1, \\
&\qquad 1 + p + p\sum_{j=1}^{n} |t_j|, 0, \ldots, 0, 1 + p + p\sum_{j=1}^{n} |t_j|) \qquad (4.1) \\
&> \alpha_{2n+1}(|t_1|, \ldots, |t_{m-1}|, |t_m| - 1, p + p\sum_{j=1}^{n} |t_j|, \ldots, p + p\sum_{j=1}^{n} |t_j|) \\
&> \alpha_{2n+1}(|s_1|, \ldots, |s_n|, |s_1|, \ldots, |s_n|, p, 0) \\
&= \|s_1, \ldots, s_n\|.
\end{aligned}$$

So the depth of a proof tree for $\mathrm{q}(t_1, \ldots, t_n)$ is bounded by $\|t_1, \ldots, t_n\|$, and hence

$$\mathrm{Q} \vdash \mathrm{q}(t_1, \ldots, t_n) \iff \exists m \leq \|t_1, \ldots, t_n\| \, (\Theta_{\mathrm{Q}}(\mathrm{q}(t_1, \ldots, t_n), m) \neq \emptyset). \qquad (4.2)$$

The existential quantor in the formula above is bounded by a primitive recursive function, so the set computed by Q is primitive recursive. ∎

5 Computation power of tame programs

In this section we show that all primitive predicates can be computed by tame predicates. We again embed the natural numbers in the Herbrand universe by means of $\iota : \mathbf{N} \to N \subset H$ (as in the second section). N can be computed by the following tame program:

$$\mathtt{nat}(0). \qquad (5.1)$$

$$\mathtt{nat}(\mathtt{X}') \leftarrow \mathtt{nat}(\mathtt{X}). \qquad (5.2)$$

We show in the next two lemmas, that the graphs of some functions can be computed by tame programs.

LEMMA 5.1
For every primitive recursive function $\varphi : \mathbf{N}^n \to \mathbf{N}$ there is a tame program, computing the graph of φ°.

PROOF. The computability of the graph of a projection, of a constant function or of the successor function is trivial.

Before we treat the composition of function and the schema of primitive recursion, we consider the Ackermann predicates again. Let A_m^r be the program we obtain from A_m by replacing $\mathtt{a}_m(\mathtt{X}_m, \ldots, \mathtt{X}_0)$ by $\mathtt{a}_m^r(\mathtt{Y}_1, \ldots, \mathtt{Y}_r; \mathtt{X}_m, \ldots, \mathtt{X}_0)$. The new parameters have no influence on a computation.

Let $\varphi : \mathbf{N} \to \mathbf{N}$ and $\psi : \mathbf{N} \to \mathbf{N}$ be primitive recursive functions, and let F and P be tame programs, computing the graphs of φ° and ψ°. There is an $m \in \mathbf{N}$ such that $\psi(x) < \beta_m(1, x)$ for every $x \in \mathbf{N}$. Let K be the program containing F, P, A_m^2 and the following clauses.

$$\mathtt{k}(\mathtt{X}, \mathtt{Z}) \leftarrow \mathtt{a}_m^2(\mathtt{X}, \mathtt{Z}; 1, 0, \ldots, 0, \mathtt{X}, 0). \tag{5.3}$$

$$\mathtt{a}_m^2(\mathtt{X}, \mathtt{Z}; \mathtt{Y}_m, \ldots, \mathtt{Y}_0) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{Y}_0) \wedge \mathtt{f}(\mathtt{Y}_0, \mathtt{Z}). \tag{5.4}$$

We may assume without loss of generality that the programs F and P neither have a predicate in common nor does one of them contain $\mathtt{a}_m^2$ or k (otherwise we must rename these predicates). Then K is a tame program; we claim that K computes the graph of $\varphi \circ \psi$. We apply this program to $\mathtt{k}(x, z)$; this resolves first to $\mathtt{a}_m^2(x, z; 1, 0, \ldots, 0, x, 0)$; then, 'running A_m^2', it resolves to $\mathtt{a}_m^2(x, z; t_m, \ldots, t_1, \psi(x))$. So $\mathsf{K} \vdash \mathtt{k}(x, z)$ iff $z = \varphi\psi(x)$. A similar proof works for functions with several variables.

It remains to prove that the graph of functions defined with primitive recursion can be represented. Let $\varphi : \mathbf{N}^3 \to \mathbf{N}$ and $\psi : \mathbf{N} \to \mathbf{N}$ again primitive recursive functions, and let F and P be programs computing the graphs of φ° and ψ°. κ is the function defined by $\kappa(x, 0) := \psi(x)$ and $\kappa(x, i+1) := \varphi(x, \kappa(x, i), i)$. Now choose $m \in \mathbf{N}$ with $\kappa(x, i) < \beta_m(1, x + i)$ for every $x, i \in \mathbf{N}$. Let K be the program containing the programs F, P and a_m^3 and the following clauses.

$$\mathtt{k}(\mathtt{X}, \mathtt{I}, \mathtt{Z}) \leftarrow \mathtt{a}_m^3(\mathtt{X}, \mathtt{I}, \mathtt{Z}; 1, 0, \ldots, 0, \mathtt{X}, \mathtt{I}, 0). \tag{5.5}$$

$$\mathtt{a}_m^3(\mathtt{X}, 0, \mathtt{Z}; \mathtt{Y}_m, \ldots, \mathtt{Y}_0) \leftarrow \mathtt{p}(\mathtt{X}, \mathtt{Z}). \tag{5.6}$$

$$\mathtt{a}_m^3(\mathtt{X}, \mathtt{I}', \mathtt{Z}; \mathtt{Y}_m, \ldots, \mathtt{Y}_0) \leftarrow \mathtt{a}_m^3(\mathtt{X}, \mathtt{I}, \mathtt{Y}_0; 1, 0, \ldots, 0, \mathtt{X}, \mathtt{I}, 0) \wedge \mathtt{f}(\mathtt{X}, \mathtt{Y}_0, \mathtt{I}, \mathtt{Z}). \tag{5.7}$$

It is easy to see that K computes the graph of κ. The recursion schema with parameters can be handled in the same way. ∎

LEMMA 5.2

The graph of a primitive recursive function $\varphi : H^n \to N \subset H$ is computable by a tame program.

PROOF. We first show that the graph of the map $\iota\gamma : H \to N$ is computable by a tame program. For the function symbol $f \in L$, the map $\gamma_f : \mathbf{N}^{n_f} \to \mathbf{N}$ in (viii) is primitive recursive and hence by Lemma 5.1, there are tame programs G_f computing the graphs of the functions γ_f°. Let n be the maximum of all the arities of function symbols of L. Let G be the program containing the following clauses.

$$\mathtt{g}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y}; \underbrace{\mathtt{Y}, \ldots, \mathtt{Y}}_{n \text{ times}}). \tag{5.8}$$

$$\mathtt{q}(\mathtt{X}, \mathtt{Y}; \mathtt{Y}_1', \mathtt{Y}_2, \ldots, \mathtt{Y}_n) \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y}; \mathtt{Y}_1, \mathtt{Y}_2, \ldots, \mathtt{Y}_n). \tag{5.9}$$

$$\vdots$$

$$\mathtt{q}(\mathtt{X}, \mathtt{Y}; \mathtt{Y}_1, \ldots, \mathtt{Y}_{n-1}, \mathtt{Y}_n') \leftarrow \mathtt{q}(\mathtt{X}, \mathtt{Y}; \mathtt{Y}_1, \ldots, \mathtt{Y}_{n-1}, \mathtt{Y}_n). \tag{5.10}$$

Furthermore for any constant $c \in L$ there is a clause

$$\mathtt{q}(c, \iota\gamma(c); \mathtt{Y}_1, \ldots, \mathtt{Y}_n). \tag{5.11}$$

and for every function symbol f there is a clause

$$\mathtt{q}(f(\mathtt{X}_1,\ldots,\mathtt{X}_{n_f}),\mathtt{Y};\mathtt{Y}_1,\ldots,\mathtt{Y}_n) \leftarrow \mathtt{g}_f(\mathtt{Y}_1,\ldots,\mathtt{Y}_{n_f};\mathtt{Y}) \wedge \bigwedge_{i=1}^{n_f} \mathtt{q}(\mathtt{X}_i,\mathtt{Y}_i;\mathtt{Y}_1,\ldots,\mathtt{Y}_i). \quad (5.12)$$

By (vii), γ is increasing, so it is easy to see that G is a tame program, computing the graph of $\iota\gamma$.

Now let $\varphi : H \to N$ be a primitive recursive function. Then, by Lemma 2.2, the function $\varphi^\star := \varphi\eta : N \to N$ is primitive recursive, and so it follows by Lemma 5.1 that its graph is computable by a tame program F. Because of $\varphi = \varphi^\star\iota\gamma$ we must construct a program K for the composition of F and G; the construction is similar to the construction in the proof of Lemma 5.1.

Let $t \in H$ be a term. Then the *depth* $\delta(t)$ of t is defined recursively by $\delta(c) := 1$ for constants c and $\delta(f(t_1,\ldots,t_n)) := 1 + \max(\delta(t_1),\ldots,\delta(t_n))$. As $\iota\gamma$ is primitive recursive, there exists an m such that $\iota\gamma(t) < \beta_m(1,\delta(t))$ for all $t \in H$. K is the program, containing the programs F, G, the program A^4_m defined in the proof of Lemma 5.1 and the following clauses.

$$\mathtt{k}(\mathtt{X},\mathtt{Z}) \leftarrow \mathtt{a}^4_m(\mathtt{X},\mathtt{Z},\mathtt{X},0';0,\ldots,0). \quad (5.13)$$

$$\mathtt{a}^4_m(\mathtt{X},\mathtt{Z},f(\mathtt{U}_1,\ldots,\mathtt{U}_{n_f}),\mathtt{V};0,\ldots,0) \leftarrow \mathtt{a}^4_m(\mathtt{X},\mathtt{Z},\mathtt{U}_i,\mathtt{V}';0,\ldots,0). \quad (5.14)$$

$$\mathtt{a}^4_m(\mathtt{X},\mathtt{Z},c,\mathtt{V}';0,\ldots,0) \leftarrow \mathtt{a}^4_m(\mathtt{X},\mathtt{Z},0,0;1,0,\ldots,0,\mathtt{V}',0). \quad (5.15)$$

$$\mathtt{a}^4_m(\mathtt{X},\mathtt{Z},0,0;t_m,\ldots,t_0) \leftarrow \mathtt{g}(\mathtt{X},t_0) \wedge \mathtt{f}(t_0,\mathtt{Z}). \quad (5.16)$$

(For each function symbol f and for each $i \le n_f$, there is a clause of the form (5.14), and for every constant there is a clause of the form (5.15).) Again we may assume that k and $\mathtt{a}^4_m$ are not contained in the programs F and G and we may assume that these two programs have no common predicate. Then K is tame. If we apply this program to $\mathtt{k}(x,z)$ then it resolves first to $\mathtt{a}^4_m(x,z,c,\delta(x);0,\ldots,0)$ if resolving with the clause (5.14) the right i is always chosen. Furthermore this resolve to $\mathtt{a}^4_m(x,z,0,0;t_m,\ldots,t_0)$, where t_0 can be any element from m with $t_0 \le \beta_m(1,\delta(x))$; so it resolves to $\mathtt{a}^4_m(x,z,0,0;t_m,\ldots,t_1,\iota\gamma(x))$, and so K resolves to true iff $\iota\gamma(x) = z$. For $\varphi : H^n \to N$, the proof is similar. ∎

Now we can prove the theorem announced:

THEOREM 5.3
Tame programs compute primitive recursive sets and every primitive recursive set is computed by a tame program.

PROOF. Lemma 4.2 says that sets computed by tame programs are primitive recursive. So let $U \subset H^n$ be a primitive recursive set. Let $\xi : H^n \to H$ be a primitive recursive function with $U = \{x \in H \mid \xi(x) = 0\}$. Then the graph of ξ is computable by a tame program, and so U is computable by a tame program too. ∎

6 Complements

The class of primitive recursive sets is closed under Boolean operations. So it follows from Theorem 5.3 that the class of sets computable by tame programs is also closed under Boolean operations. Therefore we can include negation in the definition of tame programs without changing the computation power of this class of programs. In this section we give a direct proof of this fact without using Theorem 5.3. Given a tame program P, we shall construct a program $\bar{\mathsf{P}}$ computing the complement of the set computed by P.

We start with the following program eq for equality containing a clause

$$\mathtt{eq}(c, c). \qquad \text{for every constant } c \text{ and} \tag{6.1}$$

$$\mathtt{eq}(f(\mathtt{X}_1, \ldots, \mathtt{X}_n), f(\mathtt{Y}_1, \ldots, \mathtt{Y}_n)) \leftarrow \mathtt{eq}(\mathtt{X}_1, \mathtt{Y}_1) \wedge \cdots \wedge \mathtt{eq}(\mathtt{X}_n, \mathtt{Y}_n). \tag{6.2}$$

for every function symbol f. A tame program P is called *free* if the following hold:

(xiv) The variables in the head of a clause of P are all distinct.

(xv) If $\mathtt{q}(t_1, \ldots, t_n)$ and $\mathtt{q}(s_1, \ldots, s_n)$ are the heads of two clauses of P, then either there is an $i \leq n$ such that t_i and s_i are not unifiable, or the two heads are equal.

(xvi) If q is a predicate occurring in P and if $\mathtt{q}(t_1, \ldots, t_n)$ is a variable-free atomic formula, then there is a clause in P whose head is unifiable with $\mathtt{q}(t_1, \ldots, t_n)$.

We remark that the program eq is free (in contrast to the usual way of defining equality by $\mathtt{equal}(\mathtt{X}, \mathtt{X})$.). We call two programs P, Q *equivalent* if they compute the same sets. We shall show that an equivalent free tame program can be constructed for every tame program. We begin with two lemmas:

LEMMA 6.1
For every tame program P, an equivalent tame program satisfying (xiv) of the definition above can be constructed effectively.

PROOF. Let

$$\mathtt{r}(t_1, \ldots, t_n) \leftarrow \bigwedge_i \mathtt{r}(t_1^i, \ldots, t_n^i) \wedge \Psi \tag{6.3}$$

be a clause, where Ψ is a formula not containing r, and assume that the variable X occurs in t_u and t_v (with $u \neq v$). Let Y be a new variable. If in P we replace the clause above by

$$\mathtt{r}(t_1, \ldots, t_{v-1}, t_v\left[\frac{\mathtt{Y}}{\mathtt{X}}\right], t_{v+1}, \ldots, t_n) \leftarrow \mathtt{eq}(\mathtt{X}, \mathtt{Y}) \wedge \bigwedge_i \mathtt{r}(t_1^i, \ldots, t_{v-1}^i, t_v\left[\frac{\mathtt{Y}}{\mathtt{X}}\right], t_{v+1}, \ldots, t_n^i) \wedge \Psi,$$

then we obtain a tame program computing the same set as P. ($t\left[\frac{\mathtt{Y}}{\mathtt{X}}\right]$ is the term we get from t by substituting X by Y.)

By iterated application of the above argument, we may assume that in all clauses of P having the form of (6.3) the terms t_i and t_j do not have any variables in common ($i \neq j$). We now assume that the variable X occurs v times in the term t_k of (6.3). Then we replace these v occurrences in t_k by the new variables $\mathtt{X}_1, \ldots, \mathtt{X}_v$. If $|t_k| \geq |t_k^j|$, then X occurs not more than v times in t_k^j; in this case we replace all the occurrences of X in $|t_k^j|$ by different $\mathtt{X}_i$. All occurrences of X in (6.3) not contained in such a t_k^i we replace by an arbitrary $\mathtt{X}_i$. Finally we add $\mathtt{eq}(\mathtt{X}_1, \mathtt{X}_2) \wedge \mathtt{eq}(\mathtt{X}_1, \mathtt{X}_3) \wedge \cdots \wedge \mathtt{eq}(\mathtt{X}_1, \mathtt{X}_v)$ to the body of the modified clause (6.3). If we do this for every k and every clause, then we get a tame program that computes the same set as P and satisfies (xiv). ∎

LEMMA 6.2
Let $t_1, \ldots, t_v$ be terms. Assume that for every i the variables contained in t_i are distinct. Then there is a finite set of terms $\{s_1, \ldots, s_w\}$ with the following properties.

(xvii) The variables in a s_i are all distinct.

(xviii) Every closed term is unifiable with exactly one s_i.

(xix) Each s_i is a substitution instance of a t_j.

(xx) If s_i and t_j are unifiable, then s_i is substitution instance of t_j.

PROOF. Let $\mathcal{L}_0 := \{\mathtt{X}_i \mid i \in \mathbf{N}\}$ be a set of variables. We define $\mathcal{L}_{m+1}$ recursively to be the sets of all constants $c \in L$ and all terms of the form $f(t_1, \ldots, t_n)$ where $f \in L$ and $t_i \in \mathcal{L}_m$. Let $\hat{\mathcal{L}}_m$ be the subset of $\mathcal{L}_m$ consisting of the terms t with the following properties.

(xxi) The variables in t are all distinct.

(xxii) If $\mathtt{X}_i$ occurs in t and $j < i$ then $\mathtt{X}_j$ occurs in t too. Furthermore, t_j stands on the left side of $\mathtt{X}_i$.

Then the sets $\hat{\mathcal{L}}_j$ satisfy (xvii) and (xviii). If j is large enough for all t_i to be contained in $\mathcal{L}_j$, then $\{s_1, \ldots, s_w\} := \hat{\mathcal{L}}_j$ also satisfies (xx). ∎

We take a tame program P satisfying (xiv). Let Ξ be a clause of P and t a term whose variables are all distinct and not contained in Ξ. If we substitute an arbitrary variable of Ξ by t, we again obtain a tame program satisfying (xiv). Now we are ready to prove the announced proposition:

PROPOSITION 6.3
An equivalent free tame program can be effectively constructed for any tame program P.

PROOF. By Lemma 6.1, we may assume that P satisfies (xiv). For every predicate q occurring in P we add the clause $\mathtt{q}(\mathtt{X}_1, \ldots, \mathtt{X}_n) \leftarrow \mathtt{false}$ to P. Let $\mathtt{q}(t_1^i, \ldots, t_n^i) \leftarrow \Xi^i$ be clauses of P, where $i = 1, \ldots, v$. We fix $k \leq n$ and choose $s_1, \ldots, s_w$ satisfying (xvii) to (xx) with respect to $t_k^1, \ldots, t_k^v$. We now replace a clause $\mathtt{q}(t_1^i, \ldots, t_n^i) \leftarrow \Xi^i$ by all of its substitution instances transforming t_k^i in a s_l and not changing the terms t_j^i for $j \neq k$. We repeat this for every k. Then these replacements lead to a free tame program equivalent to the original one. ∎

With this preparation, the construction of a program $\bar{\mathsf{P}}$ computing the complement of the set computed by P is easy:

PROPOSITION 6.4
If P is a tame program, then a tame program $\bar{\mathsf{P}}$ computing the complement of the set computed by P can be constructed effectively.

PROOF. By Proposition 6.3, we may assume that P is a free tame program. Let

$$\Phi \leftarrow \Psi_{i,1} \wedge \cdots \wedge \Psi_{i,m_i} \qquad i = 1, \ldots, k \tag{6.4}$$

be all the clauses of P with head Φ. Then we add to $\bar{\mathsf{P}}$ the clauses

$$\bar{\Phi} \leftarrow \bar{\Psi}_{1,\sigma(1)} \wedge \cdots \wedge \bar{\Psi}_{k,\sigma(k)} \tag{6.5}$$

where σ runs over all maps $\{1, \ldots, k\} \to \mathbf{N} \setminus \{0\}$ with $\sigma(i) \leq m_i$. By induction on the ordering (1.6) defined in the first section and by de Morgan's laws it follows, that for a variable-free formula $\mathtt{q}(t_1, \ldots, t_n)$ the following equivalence holds: $\bar{\mathsf{P}} \vdash \bar{\mathtt{q}}(t_1, \ldots, t_n) \iff \mathsf{P} \not\vdash \mathtt{q}(t_1, \ldots, t_n)$. ∎

7 Other classes of logic programs

In [1], Apt and Bezem investigate the class of acyclic programs. A normal program T is *acyclic* iff there is a level-mapping $\ell : \mathcal{L} \to \mathbf{N}$ (where $\mathcal{L}$ is the set of all variable-free literals) with the following properties:

(xxiii) $|\Phi| = |\neg\Phi|$ for all atomic formulas $\Phi \in \mathcal{L}$.

(xxiv) If $\Phi \leftarrow \Psi_1 \wedge \ldots \wedge \Psi_m$ is a variable-free instance of a clause from T, then $|\Phi| > |\Psi_i|$ for $i = 1, \ldots, m$.

Every total recursive function can be computed by an acyclic program (without negation). In the contrary to the class of tame programs the class of acyclic programs is undecidable.

It follows from the proofs of Lemmas 4.1 and 4.2 that tame programs are acyclic. So the class of tame programs is a subclass of the class of definite acyclic programs having a weaker computation power. The inclusion of the class of tame programs in the class of acyclic is not uniform in the sense that there is a set of level-mappings corresponding to the tame programs: consider the the tame program over the language $L := \{0, '\}$ and with the two clauses $\mathtt{a}(\mathtt{X}, \mathtt{Y}') \leftarrow \mathtt{a}(\mathtt{X}, \mathtt{Y})$ and $\mathtt{a}(\mathtt{X}', 0) \leftarrow \mathtt{a}(\mathtt{X}, \mathtt{X})$. Every level-function ℓ witnessing that this program is acyclic satisfies $\ell(x, y+1) > \ell(x, y)$ and $\ell(x+1, 0) > \ell(x, x)$. Then there is an $m \in \mathbb{N}$ such that $\ell(0, m) > \ell(1, 0)$. But ℓ is a witness that the program consisting only of the clause $\mathtt{b}(0, 0^{(m)}) \leftarrow \mathtt{b}(0', 0)$ is acyclic, and this program is not tame ($0^{(m)}$ stands for a zero followed by m primes).

Another class of logic programs is described by Stroetmann in [6] and [7]. He uses a distinction between input and output places of the predicates. Here we give a simplified version of this approach without negation. Therefore all clauses and goals considered in the sequel are definite. An I/O-*specification* for a program P is a map σ from the set of predicate symbols of P to $\{+, -\}^*$, where for all q the arity of q is equal to the length of $\sigma(\mathtt{q})$. t_i is called *input term* in $\mathtt{q}(t_1, \ldots, t_n)$ iff $\sigma(\mathtt{q})_i = \text{`}+\text{'}$; if $\sigma(\mathtt{q})_i = \text{`}-\text{'}$ then t_i is an *output term*. Let $FV(t)$ be the variables of t; for an atomic formula $\mathtt{q}(t_1, \ldots, t_n)$ we define

$$FV^+(\mathtt{q}(t_1, \ldots, t_n)) := \bigcup\{FV(t_i) \mid \sigma(\mathtt{q})_i = +\} \tag{7.1}$$

$$FV^-(\mathtt{q}(t_1, \ldots, t_n)) := \bigcup\{FV(t_i) \mid \sigma(\mathtt{q})_i = -\}. \tag{7.2}$$

A clause $\Phi \leftarrow \Psi_1 \wedge \ldots \wedge \Psi_m$ is called *σ-allowed* iff it satisfies the following two conditions:

$$\mathtt{X} \in FV^-(\Phi) \Rightarrow \mathtt{X} \in FV^+(\Phi) \vee \exists j \in \{1, \ldots, m\}(\mathtt{X} \in FV(\Psi_j)) \tag{7.3}$$

$$\mathtt{X} \in FV^+(\Psi_i) \Rightarrow \mathtt{X} \in FV^+(\Phi) \vee \exists j \in \{1, \ldots, i-1\}(\mathtt{X} \in FV(\Psi_j)) \tag{7.4}$$

A goal $\leftarrow \Psi_1 \wedge \ldots \wedge \Psi_m$ is called *σ-allowed* iff it satisfies the following condition:

$$\mathtt{X} \in FV^+(\Psi_i) \Rightarrow \exists j \in \{1, \ldots, i-1\}(\mathtt{X} \in FV(\Psi_j)). \tag{7.5}$$

Obviously, in an allowed goal FV^+ of the leftmost atom is always empty. If Φ is an allowed goal and Φ' is derived from Φ by an allowed clause, then Φ' is allowed, too.

For a term t let $\ln(t)$ be the number of occurrences of function symbols and constants contained in t (so $\ln(\mathtt{X}) = 0$ for variables). (Here we made the second simplification: Stroetmann defines ln with respect to a weight function for constants and function symbols). For an atomic formula $\mathtt{q}(t_1, \ldots, t_n)$ let $\ln(\mathtt{q}(t_1, \ldots, t_n)) := 1 + \Sigma_i \ln(t_i)$ and $\ln^+(\mathtt{q}(t_1, \ldots, t_n)) := 1 + \Sigma_{\sigma(\mathtt{q})_i = +} \ln(t_i)$; for a goal Φ let $\ln(\Phi)$ and $\ln^+(\Phi)$, respectively, be the sum of ln (or $\ln^+$) of the atomic formulas contained in Φ. A definite program T is *σ-ordered* iff there is a map level from the set of all predicates contained in P to the set of natural numbers such that for all clauses $\Phi \leftarrow \Psi_1 \wedge \ldots \wedge \Psi_m$ of P the following conditions are satisfied:

(xxv) $\mathrm{level}(\Phi) \geq \mathrm{level}(\Psi_i)$ for $i \in \{1, \ldots, m\}$.

(xxvi) If the set $I := \{i \in \{1,\ldots,m\} \mid \underline{\text{level}}(\Phi) = \underline{\text{level}}(\Psi_i)\}$ is non-empty, then

$$\sum_{i\in I}(\ln^+(\psi_i)) < \ln^+(\Phi) \tag{7.6}$$

$$\bigcup_{i\in I} FV^+(\Psi_i) \subset_{multi} FV^+(\Phi). \tag{7.7}$$

(In (7.7) the sets FV^+ are considered as multisets; $\cup$ stands for the union of multisets and $\subset_{multi}$ for multiset-inclusion.)

For short we call a σ-ordered program whose classes are σ-allowed just a *σ-program*.

Stroetmann proved that every primitive recursive function can be computed by a σ-program. On the other hand he proved that if P is a σ-program and if Φ is a σ-allowed goal then the SLD-tree for $P \cup \{\Phi\}$ is finite, provided that the SLD-tree is built with respect to the computation rule to select always the leftmost literal in a goal. With other computation rules infinite SLD-trees can exist, so there are σ-programs that are not acyclic. In his proof Stroetmann used (infinite) ordinals. The following proposition gives bounds for the size of SLD-trees for σ-programs. From now we also fix the mentioned computation rule for SLD-resolution (choosing always the leftmost atom).

PROPOSITION 7.1
Let Q be a σ-program. Then there is a primitive recursive function $\underline{dp}$ such that for every allowed goal Φ the depth of the SLD-tree for $Q \cup \{\Phi\}$ is less than $\underline{dp}(\ln^+(\Phi))$.

PROOF. Let $\leftarrow p_1(t_{1,1},\ldots t_{1,n_1}) \wedge \ldots \wedge p_m(t_{m,1},\ldots,t_{m,n_m})$ be a goal. We call an occurrence of a subterm u in $t_{i,j}$ to be *inessential* if $t_{i,j}$ is an output term or if there are such $k < i$ and l that u occurs also in the output term $t_{k,l}$. For a goal Φ and for an occurrence t of a term in Φ let $\ln^{\#}(t)$ be the number of all occurrences of constants and function symbols contained in t, but not inside of an inessential subterm; furthermore we define $\ln^{\#}(\Phi)$ to be the sum of the number of atomic subformulas of Φ and of the number of all occurrences of constants and function symbols contained in Φ, but not inside of an inessential subterm. Obviously, $\ln^{\#}(\Phi) \leq \ln^+(\Phi)$; if Φ is an allowed goal and if Φ' is a substitution instance of Φ then $\ln^{\#}(\Phi') \leq \ln^{\#}(\Phi)$. We construct a primitive recursive function $\nu : \mathbf{N} \to \mathbf{N}$ with the following property:

(xxvii) If Ξ is an allowed goal and if Ξ' is a goal derived from Ξ in one resolution step with a clause from Q then $\ln^{\#}(\Xi') \leq \nu(\ln^{\#}(\Xi))$.

As in the proof of Lemma 4.2 we choose $p \in \mathbf{N}$ with the following properties:

(xxviii) For every clause Φ of Q and for every input term t of the body of Φ the following is true: if the polynomial $|t| = |t|(|X_1|,\ldots,|X_r|;|Y_1|,\ldots,|Y_s|)$ is defined as in section 1, $X_1,\ldots,X_r$ are the variables contained in an input term of the head of Φ and $Y_1,\ldots,Y_s$ are the other variables, then $p > |t|(1,\ldots,1;0,\ldots,0)$.

(xxix) p is greater than $p_a p_o$, where p_a is the maximum of all arities of function symbols and p_o is greater than all $\ln(t)$, where t ranges over all output terms of the heads of the clauses of Q.

Let Ξ be the allowed goal $\leftarrow \Phi_1 \wedge \ldots \wedge \Phi_s$ and let Θ be a clause $\Phi \leftarrow \Psi_1 \wedge \ldots \wedge \Psi_r$ of Q Ξ' is the goal obtained from Ξ in one resolution step with the above clause, so Ξ' is $\leftarrow \Psi'_1 \wedge \ldots \wedge \Psi'_r \wedge \Phi'_2 \wedge \ldots \wedge \Phi'_s$, where $\Phi'_i := \Phi_i\theta$ and $\Psi'_i := \Psi_i\theta$, provided that $\theta := \text{mgu}(\Phi,\Phi_1)$. Let k be the maximum of 1 and of all $\ln^{\#}(t)$ for input terms t in Ξ. Then

$k \geq \ln^{\#}(t')$ for all terms t' in the goal $\Xi^\circ := \leftarrow \Phi'_1 \wedge \ldots \wedge \Phi'_s$. If we replace Φ'_1 in Ξ° by $\Psi'_1 \wedge \ldots \wedge \Psi'_r$ we get the goal Ξ' and we claim that $pk \geq \ln^{\#}(t')$ for all terms t' in Ξ'.

Assume first that t' is in Ψ'_i. Then $t' = t(\bar{u}; \bar{v})$, where t is the corresponding term in Θ, $\bar{u} = (u_1, \ldots, u_m)$ are terms substituted for variables contained in the input terms of Φ and $\bar{v}$ are terms substituted for the other variables. With (xxviii) we obtain $\ln^{\#}(t') \leq |t|(|\bar{u}|; \bar{0}) \leq p \max\{1, \ln(u_1), \ldots, \ln(u_m)\}$. But the terms u_i are contained in the input terms of Φ', hence also in the input terms of Φ_1 (the input terms of Φ' and Φ_1 are identical, because Ξ is allowed). So it follows that $\ln^{\#}(t') \leq pk$.

Let t' be in Φ'_i ($i > 1$). Then not every subterm u' of t' that is inessential with respect to the goal Ξ° remains inessential inside the goal Ξ'. But t' contains at most kp_a maximal inessential subterms inside Ξ°; furthermore, such a subterm contains at most p_o function symbols and constants that are not contained in an inessential subterm with respect to Ξ'. It follows $\ln^{\#}(t') \leq k + (kp_a)p_o = k(1 + p_a p_o) \leq pk$. So the claim is proved.

There is a natural number p' such that making a resolution step the number of terms in the derived goal is at most by p' greater than the number of terms in the original goal. Let k_t be the number of atomic formulas of Ξ and let p_b be the maximum of the arities of the predicates in Q. Then it follows that $\ln^{\#}(\Xi') \leq (p' + k_t) + pk(p' + p_b k_t) \leq (1 + pk)(p' + p_b k_t) \leq (1 + p \ln^{\#}(\Xi))(p' + p_b \ln^{\#}(\Xi))$. We define $\nu(x) := (1 + px)(p' + p_b x)$, so $\ln^{\#}(\Xi') \leq \nu(\ln^{\#}(\Xi))$. This ν satisfies condition (xxvii).

Now we can begin to define the function $\underline{\mathrm{dp}}$. We define primitive recursive functions $\underline{\mathrm{dp}}_l$ with the property that the depth of the SLD-tree for $Q \cup \{\Phi\}$ is less than $\underline{\mathrm{dp}}_l(\ln^{\#}(\Phi))$ for all allowed goals Φ whose predicates all have a level not greater than l. Then we define $\underline{\mathrm{dp}} := \underline{\mathrm{dp}}_\ell$, where ℓ is the maximum of the levels of the predicates in Q.

We assume that the functions $\underline{\mathrm{dp}}_{l-1}$ exist (let $\underline{\mathrm{dp}}_{-1} := 0$). For a goal Φ let $\ln_l^+(\Phi)$ be the sum of all $\ln^+(\Psi)$, where Ψ ranges over all atomic subformulas of Φ with level l. First we define a function $\underline{\mathrm{dp}}'_l$ so that for an allowed goal Φ the depth of the SLD-tree for $Q \cup \{\Phi\}$ is less than $\underline{\mathrm{dp}}'_l(\ln^{\#}(\Phi), \ln_l^+(\Phi))$, provided that the following condition is true:

(xxx) All predicates of Φ have a level not greater than l and there are no free variables in an input term of a predicate of Φ with level l.

We can set $\underline{\mathrm{dp}}'_l(x, 0) := \underline{\mathrm{dp}}_{l-1}(x)$. Now let Φ be a goal satisfying (xxx). We split Φ in two subgoals Φ_1 and Φ_2 so that Φ_1 does not contain a predicate symbol of level l, but the first predicate symbol of Φ_2 has exactly level l. We want to estimate the length of a branch of the SLD-tree for Φ. The length of the branch corresponding to the subgoal Φ_1 is less than $\underline{\mathrm{dp}}_{l-1}(\ln^{\#}(\Phi))$. The remaining subgoal Φ'_2 is a substitution instance of Φ_2. Going further one resolution step we obtain a goal Φ''_2; from (xxvi) it follows $\ln_l^+(\Phi''_2) < \ln_l^+(\Phi_2)$. Furthermore, $\ln^{\#}(\Phi''_2) \leq \nu'(\ln^{\#}(\Phi))$, where $\nu'(x) := \nu^{\underline{\mathrm{dp}}_{l-1}(x)}(x)$, so we can set

$$\underline{\mathrm{dp}}'_l(x, y+1) := 1 + \underline{\mathrm{dp}}_{l-1}(x) + \underline{\mathrm{dp}}'_l(\nu'(x), y). \tag{7.8}$$

In the next step we construct a primitive recursive function $\underline{\mathrm{dp}}''_l$ such that $\underline{\mathrm{dp}}''_l(\ln^{\#}(\Phi), y)$ gives a bound for the SLD-trees for Φ if the predicates of Φ have a level not greater than l and if y is the number of atomic formulas in Φ. This can be done by

$$\underline{\mathrm{dp}}''_l(x, 1) := \underline{\mathrm{dp}}'_l(x, x) \tag{7.9}$$

$$\underline{\mathrm{dp}}''_l(x, y+1) := \underline{\mathrm{dp}}'_l(x, x) + \underline{\mathrm{dp}}''_l(\nu''(x), y) \tag{7.10}$$

where $\nu''(x) := \nu^{\underline{\mathrm{dp}}'_i(x,x)}(x)$. Now we are ready to define $\underline{\mathrm{dp}}_i(x) := \underline{\mathrm{dp}}''_i(x,x)$. ∎

From the above proposition it follows immediately (as in Lemma 4.2) that all functions and predicates computed by σ-programs are primitive recursive. (The generalization to Stroetmann's class with negation and weight function is straightforward.)

References

[1] K R. Apt and M Bezem, Acyclic programs. *New Generation Computing* **9**, 335–363, 1991

[2] S. Feferman, Finitary inductively presented logics, in *Logic Colloquium '88*, North-Holland, Amsterdam, pp 191–220, 1989.

[3] J W Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Heidelberg, 1987

[4] Y. N Moschovakis, *Elementary Induction on Abstract Structures*, North-Holland, Amsterdam, 1974

[5] A R. Meyer and D. M Ritchie, The complexity of loop programs In *Proceedings of the ACM 22nd National Conference*, Thomson Book Co , Washington, DC, pp 465–469, 1967.

[6] K. Stroetmann, Vollständige Resolutionskalküle für PROLOG PhD dissertation, Westfählische Wilhelms-Universität Münster, Germany, 1981

[7] K. Stroetmann, A completeness result for SLDNF-*resolution*. Preprint, 1992, to appear in *Journal of Logic Programming*.

Received 10 February 1992