7-1991

The Complexity of Local Stratification

Peter Cholak
University of Michigan, Department of Mathematics

Howard A. Blair
Syracuse University, School of Computer and Information Science, blair@top.cis.syr.edu

SU-CIS-91-27

The Complexity of Local Stratification

Peter Cholak and Howard A. Blair

July 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

The Complexity of Local Stratification

Peter Cholak
Victoria University of Wellington
Department of Mathematics
P.O. Box 600
Wellington, New Zealand
cholak@auri.vuw.ac.nz

Howard A. Blair
School of Computer and Information Science
Syracuse University
blair@top.cis.syr.edu
and
Xerox Webster Research Center

ABSTRACT

The class of locally stratified logic programs is shown to be Π_1^1-complete by the construction of a reducibility of the class of infinitely branching nondeterministic finite register machines.

1 Introduction

Stratified logic programs, which restrict the manner in which recursion and negation can occur together in a logic program were introduced by Apt, Blair and Walker [ABW88], and Van Gelder [VG88]. The class of stratified logic programs over an effectively presented language for first-order logic is decidable in linear time as a problem of cycle detection in dependency graphs. Kolaitis [Ko87] showed that the perfect model of a stratified program is Δ_1^1; subsequently Apt and Blair [AB90] established the more precise result that the perfect model of a stratified program with n strata is Σ_n^0 and that for each n one can find in a uniform way a stratified program with a Σ_n^0-complete perfect model. Przymusinski [Pr88] introduced a wider class of programs, the locally stratified logic programs, that enjoy many of the properties of stratified programs with regard to managing negation. The main result of this paper is concerned with the complexity of this class.

We show that the class of programs with a well-founded ground negative dependency relation is Π_1^1-complete. The ground negative dependency relation is given in the following definition.

Definition 1.1: Let P be a *normal* logic program (*i.e.* a program in which positive as well as negative literals may occur in the bodies of the program's clauses), and let ground(P) be the set of ground instances of the program's clauses with respect to the Herbrand universe of the language of P. Ground atom A *refers positively to* ground atom B (in P) if there is a clause in ground(P) of the form

$$A \leftarrow L_1 \,\&\, \ldots \,\&\, B \,\&\, \ldots \,\&\, L_n$$

A *refers negatively to* B if there is a clause in ground(P) of the form

$$A \leftarrow L_1 \,\&\, \ldots \,\&\, \neg B \,\&\, \ldots \,\&\, L_n$$

A *refers to* B if A refers to B either positively or negatively. (Note that A may both positively and negatively refer to B.) A *depends on* B if (A, B) is in the transitive closure of the *refers to* relation. A *positively depends on* B if (A, B) is in the transitive closure of the *refers positively to* relation. A *negatively depends on* B if there are atoms A' and B' such that A depends on A' or is A', B' depends on B or is B, and A' refers negatively to B'. We say that the pair (A, B) is in the *negative dependency* relation if A negatively depends on B. Atom A negatively depends *directly* on atom B if there is an atoms A' such that A depends on A' or is A' and A' refers negatively to B. We say that the pair (A, B) is in the *direct negative dependency* relation if A negatively depends directly on B.

Note that the direct negative dependency relation is well-founded if, and only if, the negative dependency relation is well-founded.

The following basic lemma relates well-foundedness of negative dependency to local stratification. For our purposes in this paper we can take the lemma as a definition of *local stratification*. The interested reader should consult [Pr88] for the historically prior definition.

Lemma 1.1: Normal program P is locally stratified if, and only if, the negative dependency relation is of P well-founded. ■

The recursion-theoretic complexity of the decision problem of the well-foundedness of the ground negative dependency relation is given precisely by the following theorem.

Theorem 1.1: The set of (Gödel numbers of) normal logic programs P which have well-founded negative dependency relations is Π^1_1-complete.

We first briefly describe the method of proof. We augment finite-register machines to allow for "infinitely branching" nondeterministic computations by permitting the following non-deterministic *choice* instruction:

```
    X := choice
```

to be used. Call the finite-register machine programs determined by the above *choice instruction* together with the standard finite-register machine instructions *nondeterministic* finite-register machine programs. Operationally, the choice instruction nondeterministically assigns a natural number to `X` in one step. This introduces infinitely branching nondeterministic computations that cannot be simulated in any suitable sense by finitely branching nondeterministic computations because such finitely branching nondeterministic processes used to select an arbitrarily large data object do not halt on at least one computation path. In particular, the class of nondeterministic finite-register programs that halt independently of the choices made during the computation and independently of the *initial state, i.e.* the initial values of the registers and initial value of the program counter, is Π^1_1-complete. There are a variety of ways of showing this. Now translate the nondeterministic finite-register programs into (bi-Horn) definite clause programs, by the method of Shepherdson in [Sh91]. (The translation of choice instructions into definite clauses is described below.) Now copy each definite clause program obtained to a program with a negation inserted in the literal in the body. Let R be a finite-register machine program, and let P_R be the resulting copy of the translation of R into a normal program. Then the ground negative dependency relation of P_R is well-founded iff R halts when started in any state independently of the choices made during R's computation. Since the class of register machine programs with this property is Π^1_1-complete, so is the class of logic programs with well-founded negative dependency relations.

2 Bi-Horn Programs as Finite Register Machines

We review finite-register machines and their translations into definite clause programs. The translation is that of Shepherdson's [Sh91], but extended from 2 registers to r registers.

Definition 2.1: An r-register program M is a sequence of n instructions and r natural number variables $\mathtt{X}_1, \ldots, \mathtt{X}_r$ where each instruction has one of the following forms:

i) `X`$_k$ `:= X`$_k$ `+1`
ii) `if X`$_k \neq$ `0 then (X`$_k$ `:= X`$_k$`-1; goto` j`)`

where $j \in \{1, \ldots, n+1\}$ and $k \in \{1, \ldots, r\}$.
A *nondeterministic* r-register machine program M is a sequence of n instructions and r natural number variables $\mathtt{X}_1, \ldots, \mathtt{X}_r$ where each instruction has one of the forms above, or has the form
iii) `X`$_k$ `:= choice`
Instructions of type (i) we call *increment-*`X` instructions, and instructions of type (ii) we call *conditional decrement-*`X` instructions. Instructions of type (iii) we call *choice* instructions. Hereafter we refer to r-register machine programs, both deterministic and nondeterministic, simply as *r-register machines.*

Informally, the intended operational meanings of the above instructions are sufficiently clear. It is understood here that control passes from the i^{th} instruction to the $(i+1)^{\text{st}}$ instruction in the sequence unless a "`goto` j instruction" is executed during execution of the i^{th} instruction, in which case control passes to the j^{th} instruction. These operational notions will be formalized below in the definition of M's *transition relation.* Note that every r-register machine is a *nondeterministic* r-register machine.

Definition 2.2: Let f be a unary partial recursive function on the natural numbers. 2-register machine M *computes* f if for every natural number a: M, when started at instruction 1 with $\mathtt{X}_1 = 2^a$, $\mathtt{X}_2 = 0$ halts (by passing control to the nonexistent $(n+1)^{\text{st}}$ instruction) with $\mathtt{X}_1 = 2^{f(a)}$ and $\mathtt{X}_2 = 0$ if $f(a)$ is defined, and does not halt if $f(a)$ is undefined.

Basic fact: Every unary partial recursive function is computable by some 2-register machine, (*cf.* [Sh91, SS63]).

One can give a 3-register machine which, when started at instruction 1 with $\mathtt{X}_1 = a$, $\mathtt{X}_2 = \mathtt{X}_3 = 0$, eventually halts with $\mathtt{X}_1 = 2^a$, $\mathtt{X}_2 = \mathtt{X}_3 = 0$. Similarly, one can give a 3-register machine which, when started at instruction 1 with $\mathtt{X}_1 = 2^a, \mathtt{X}_2 = \mathtt{X}_3 = 0$, eventually halts with $\mathtt{X}_1 = a$, $\mathtt{X}_2 = \mathtt{X}_3 = 0$. Both of these remarks follow from the fact that one can give 2-register machine programs that map $(\mathtt{X}_1, \mathtt{X}_2) = (x, 0)$ to $(\mathtt{X}_1, \mathtt{X}_2) = (2x, 0)$ and $(\mathtt{X}_1, \mathtt{X}_2) = (2x, 0)$ to $(\mathtt{X}_1, \mathtt{X}_2) = (x, 0)$, respectively. These programs can then be augmented with a third register to keep a count of how many times these actions are performed. It follows that with three registers each unary partial recursive function is computable without having to encode the input and output as an exponent of 2. Similarly, with four registers, each binary partial recursive function is computable with the inputs stored in e.g. the first two registers. (It is of interest to note that a bijective pairing function can then be implemented with four registers.) This turns out to be an enormous convenience for our purposes.

We assume from here on that logic programs are written over a first-order language whose Herbrand universe is generated by the constant symbol `0` and unary function symbol `s`. We adopt the following syntactic abbreviations. $\mathtt{s}^0\mathtt{(0)}$ stands for `0` and $\mathtt{s}^{n+1}\mathtt{(0)}$ stands for $\mathtt{s(s}^n\mathtt{(0))}$.

Definition 2.3: The *translation* of r-register machine M into definite clause program P is obtained by translating each of the machine's instructions as follows.
If the $i$th instruction has the form

$$\mathtt{X}_k := \mathtt{X}_k + 1$$

then translate this instruction into the clause

$$\ell_i(\mathtt{X}_1, \ldots, \mathtt{X}_k, \ldots, \mathtt{X}_r) \leftarrow \ell_{i+1}(\mathtt{X}_1, \ldots, \mathtt{s}(\mathtt{X}_k), \ldots, \mathtt{X}_r).$$

If the $i$th instruction has the form

$$\mathtt{if}\ \mathtt{X}_k \neq 0\,\mathtt{then}\ (\mathtt{X}_k\ \mathtt{:=}\ \mathtt{X}_k\mathtt{-1;}\ \mathtt{goto}\,\mathtt{j})$$

then translate this instruction into the two clauses

$$\begin{aligned} \ell_i(\mathtt{X}_1, \ldots, \mathtt{s}(\mathtt{X}_k), \ldots, \mathtt{X}_r) &\leftarrow \ell_j(\mathtt{X}_1, \ldots, \mathtt{X}_k, \ldots, \mathtt{X}_r) \\ \ell_i(\mathtt{X}_1, \ldots, \mathtt{X}_{k-1}, \mathtt{0}, \mathtt{X}_{k+1} \ldots, \mathtt{X}_r) &\leftarrow \ell_{i+1}(\mathtt{X}_1, \ldots, \mathtt{X}_{k-1}, \mathtt{0}, \mathtt{X}_{k+1} \ldots, \mathtt{X}_r). \end{aligned}$$

If the $i$th instruction is a choice instruction $\mathtt{X}_k$ `:= choice` then this translates into

$$\ell_i(\mathtt{X}_1, \ldots, \mathtt{X}_k, \ldots, \mathtt{X}_r) \leftarrow \ell_{i+1}(\mathtt{X}_1, \ldots, \mathtt{Z}, \ldots, \mathtt{X}_r)$$

where `Z` is distinct from the variables $\mathtt{X}_1, \ldots, \mathtt{X}_k, \ldots, \mathtt{X}_r$.
Finally, add the unit clause

$$\ell_{n+1}(\mathtt{X}_1, \ldots, \mathtt{X}_r).$$

where n is the number of instructions in M. P is the set of all clauses obtained by the above procedure.

Observe that every clause in the translation of M has exactly one atom in its body, except for the final unit clause. Observe also that the only clauses that have a variable occurring in their body not occurring in their head (*i.e.* a *local* variable) are clauses which result from translating choice instructions.

Definition 2.4: A *state* of a nondeterministic r-register machine M is an $(r+1)$-tuple of natural numbers $\langle i, x_1, \ldots, x_r\rangle$ such that $1 \leq i \leq n+1$ where n is the number of instructions in M. The *transition relation* $\vdash_M$ is a binary relation on states of M satisfying the following.

$\langle i, x_1, \ldots, x_k, \ldots, x_r\rangle \vdash_M \langle i, x'_1, \ldots, x'_k, \ldots, x'_r\rangle$ iff $i \leq n$, $1 \leq k \leq r$ and one of the following conditions holds:

i) $i' = i+1, x'_k = x_k + 1, x'_j = x_j$ for all j such that $1 \leq j \leq r$ and $j \neq k$ and the i^{th} instruction of M is an increment-$\mathtt{X}_k$ instruction.

ii) $i' = i+1, x'_k = x_k = 0, x'_j = x_j$ for all j such that $1 \leq j \leq r$ and $j \neq k$ and the i^{th} instruction of M is a conditional decrement-$\mathtt{X}_k$ instruction.

iii) $x'_k = x_k - 1 \geq 0, x'_j = x_j$ for all j such that $1 \leq j \leq r$ and $j \neq k$ and the i^{th} instruction of M is a conditional decrement-$\mathtt{X}_k$ instruction.

iv) $i' = i+1, x'_j = x_j$ for all j such that $1 \leq j \leq r$ and $j \neq k$ and the i^{th} instruction of M is a choice instruction that sets the value of $\mathtt{X}_k$.

It should be immediately clear that the following proposition holds.

Proposition 2.1:

$$\langle i, x_1, \ldots, x_r \rangle \vdash_M \langle i', x'_1, \ldots, x'_r \rangle$$

iff

$$\ell_i(\mathtt{s}^{x_1}(0), \ldots, \mathtt{s}^{x_r}(0)) \text{ refers to } \ell_{i'}(\mathtt{s}^{x'_1}(0), \ldots, \mathtt{s}^{x'_r}(0))$$

where P is the logic program translation of M. ■

Definition 2.5: A *computation* of a nondeterministic r-register machine M is a sequence of states $\{\sigma_i\}_{0 \leq i < \alpha}$ where $1 \leq \alpha \leq \omega$ such that $\sigma_{i-1} \vdash_M \sigma_i$ for every i in the range $1 \leq i < \alpha$. The computation is *finite* if α is finite; otherwise it is *infinite*. We say that the computation *starts* at σ_0. The computation *halts* if α is finite and there is no state σ such that $\sigma_{\alpha-1} \vdash_M \sigma$. M *always halts* when started in state σ_0 if every computation of M that starts in state σ_0 halts. A state σ is *reachable* from a state σ_0 if there is a computation that starts in state σ_0 in which σ occurs. (Notice that a state is reachable from itself.) Lastly, a state of the form $\langle 1, x, 0 \ldots, 0 \rangle$ is called an *initial* state. (Note that a computation need not begin with an initial state.)

Corollary 2.1: The following are equivalent:
(i) Every computation of M halts.
(ii) $\vdash_M$ is well-founded.
(iii) The *refers to* relation of the translation P of M is well-founded. ■

The problem of whether an r-register machine always halts when started in any of its states we call the *strong* halting problem. The following corollary shows that the well-foundedness of the negative dependency relation, indeed the dependency relation itself, is at least as complex as the strong halting problem for infinitely branching nondeterministic r-register machines.

Corollary 2.2: The class of (Gödel numbers of) nondeterministic r-register machines for which $\vdash_M$ is well-founded is one-one reducible (*cf.* [Ro67]) to the class of normal logic programs P whose negative dependency relation is well-founded.

Proof: Let X be the class of nondeterministic r-register machines M for which $\vdash_M$ is well-founded, and let Y be the class of definite clause logic programs whose *refers to* relation is well-founded. Corollary 2.1 shows that the translation of nondeterministic r-register machines into definite clause logic programs is a one-one reducibility of class X into class Y. Let Z be the class of normal logic programs whose *negative dependency* relation is well-founded. Obtain the one-one reducibility of class X into class Z by translating a given r-register machine M into definite clause program P, and then replace each clause $A \leftarrow \dot{B}$ in P by $A \leftarrow \neg B$. ■

Proposition 2.2: The class of (Gödel numbers of) of normal logic programs P for whose *negative dependency* relation is well-founded is Π^1_1.

Proof: We need only show here that the class of normal logic programs P whose *negative dependency* relation is well-founded is Π^1_1, *i.e.* that Π^1_1 is an "upper bound" on the complexity of this class of normal programs. The *negative dependency* relation is a recursively enumerable relation on the Herbrand base of P. The proposition then follows from the fact that the class of (indices) of well-founded r.e. binary relations is Π^1_1. (In fact Π^1_1-complete.) ■

3 Completeness

In this section we will, in effect, equip our nondeterministic finite register machines with a (0-jump) oracle.

The two preceding propositions, together with the corollaries to the first, would be sufficient to prove the main result of this section if we knew that the set of (Gödel numbers of) nondeterministic r-register machines with well-founded transition relations was Π^1_1-complete. It is relatively easy to establish that the set of nondeterministic r-register machines which always halt, *when started in an initial state* is Π^1_1-complete, but we must show, for our present purposes, that either the set of machines which always halt, even when started from a state not reachable from an initial state is itself Π^1_1-complete, or else further modify our nondeterministic r-register machines to circumvent this difficulty. We take the latter approach.

To complete the demonstration of the main result we shall proceed through the following steps. First, we define an alternative (recursively enumerable) transition relation $\vdash'_M$ which has the property that if M is in state σ and σ is not reachable from an initial state, then M halts *i.e.* there is no transition from σ to another state. This immediately yields a modified notion of a computation of a nondeterministic r-register machine. We then show how to translate nondeterministic r-register machines into normal logic programs so that with respect to this modified notion of computation, corollary 2.2 and proposition 2.2 continue to hold. What is gained is that now it becomes a much more simple matter to prove that the class of nondeterministic r-register machines which always halt, independently of the state they start in, is Π^1_1-complete. The main result then follows at once from corollary 2.2 and proposition 2.2.

Definition 3.1: Let M be a nondeterministic r-register machine. The *enhanced transition relation* $\vdash'_M$ is given by

$$\sigma \vdash'_M \tau \text{ iff } \sigma \vdash_M \tau \text{ and } \sigma \text{ is reachable from some initial state.}$$

Next, we show how to translate nondeterministic r-register machines into normal logic programs to reflect for our present purposes the enhanced transition relation.

Definition 3.2: The *enhanced* translation of nondeterministic r-register machine M into normal logic program P_M is obtained as follows. First, let $\hat{P}_M$ be the translation of M into a normal logic program as given by definition 2.3. To obtain *definite clause* program Q_M from $\hat{P}_M$ include in Q_M a clause

$$\ell'_{i'}(\mathtt{u}'_1, \ldots, \mathtt{u}'_r, \mathtt{I}, \mathtt{X}_1, \ldots, \mathtt{X}_r) \quad\leftarrow\quad \ell'_i(\mathtt{u}_1, \ldots, \mathtt{u}_r, \mathtt{I}, \mathtt{X}_1, \ldots, \mathtt{X}_r)$$

for each clause

$$\ell_i(\mathtt{u}_1, \ldots, \mathtt{u}_r) \quad\leftarrow\quad \ell_{i'}(\mathtt{u}'_1, \ldots, \mathtt{u}'_r)$$

in $\hat{P}_M$. Also include in Q_M the clauses

$$\begin{array}{lcl} \ell'_1(\mathtt{U}_1, \mathtt{0}, \ldots, \mathtt{0}, \mathtt{s}^1(\mathtt{0}), \mathtt{X}_1, \ldots, \mathtt{X}_r) & \leftarrow & \ell_1(\mathtt{X}_1, \ldots, \mathtt{X}_r) \\ & \vdots & \\ \ell'_1(\mathtt{U}_1, \mathtt{0}, \ldots, \mathtt{0}, \mathtt{s}^{n+1}(\mathtt{0}), \mathtt{X}_1, \ldots, \mathtt{X}_r) & \leftarrow & \ell_{n+1}(\mathtt{X}_1, \ldots, \mathtt{X}_r) \end{array}$$

We assume the predicate symbols $\ell'_1, \ldots, \ell'_{n+1}$ are all distinct and distinct from each of the distinct predicate symbols $\ell_1, \ldots, \ell_{n+1}$.
Now, to obtain the normal program P_M from Q_M include the clauses of Q_M together with a clause

$$\ell_i(\mathtt{u}_1, \ldots, \mathtt{u}_r) \quad\leftarrow\quad \neg\ell'_i(\mathtt{u}'_1, \ldots, \mathtt{u}'_r, \mathtt{s}^{i'}(\mathtt{0}), \mathtt{u}'_1, \ldots, \mathtt{u}'_r)$$

for each clause

$$\ell_i(\mathtt{u}_1, \ldots, \mathtt{u}_r) \quad\leftarrow\quad \ell_{i'}(\mathtt{u}'_1, \ldots, \mathtt{u}'_r)$$

in $\hat{P}_M$. This completes the definition of the *enhanced translation.*

Some notation: If ρ is a binary relation, we denote the transitive closure of ρ by ρ^+.

Proposition 3.1: Let M be a nondeterministic r-register machine, $\vdash'_M$ the enhanced transition relation of M, and P_M the enhanced translation of M into a normal program. Then with respect to P_M,

$$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{s}^j(\mathtt{0}), \mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}))$$

negatively depends directly on

$$\ell'_k(\mathtt{s}^{x'_1}(\mathtt{0}), \ldots, \mathtt{s}^{x'_r}(\mathtt{0}), \mathtt{s}^k(\mathtt{0}), \mathtt{s}^{x'_1}(\mathtt{0}), \ldots, \mathtt{s}^{x'_r}(\mathtt{0}))$$

if, and only if

$$\langle j, x_1 \ldots, x_r\rangle \vdash'_M \langle k, x'_1, \ldots, x'_r\rangle .$$

Proof: Let n be the number of instructions in M. There is no state σ such that $\langle n+1, x_1, \ldots, x_r\rangle \vdash'_M \sigma$, for any natural numbers $x_1, \ldots, x_r$. Similarly, there is no atom A such that $\ell_{n+1}(\mathtt{x}_1, \ldots, \mathtt{x}_r)$ refers to A, for any terms $\mathtt{x}_1, \ldots, \mathtt{x}_r$; a fiortiori, there is no atom A such that $\ell_{n+1}(\mathtt{x}_1\mathtt{a}, \ldots, \mathtt{x}_r)$ refers negatively to A, with respect to either program $\hat{P}_M$ or program P_M.

Suppose that $\langle j, x_1, \ldots, x_r\rangle$ is a state of M not reachable from an initial state; *i.e.* there is no initial state σ such that $\sigma \vdash^+_M \langle j, x_1, \ldots, x_r\rangle$. Then with respect to $\hat{P}_M$

$$\ell_1(\mathtt{s}^{x'}(\mathtt{0}), \mathtt{0}, \ldots, \mathtt{0}) \quad \text{does not depend on} \quad \ell_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}))$$

for any natural number x'. In particular $\langle j, x_1, \ldots, x_r\rangle$ *is not an initial state.* Hence with respect to Q_M

$$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{i}'', \mathtt{t}_1, \ldots, \mathtt{t}_r)$$

does not depend on

$$\ell'_1(\mathtt{s}^{x'}(\mathtt{0}), \mathtt{0}, \ldots, \mathtt{0}, \mathtt{i}'', \mathtt{t}_1, \ldots, \mathtt{t}_r),$$

for any natural number x' and *any* ground terms $\mathtt{i}''$, $\mathtt{t}_1, \ldots, \mathtt{t}_r$.

Thus, also with respect to Q_M,

$$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{s}^{i''}(\mathtt{0}), \mathtt{t}_1, \ldots, \mathtt{t}_r) \quad \text{does not depend on} \quad \ell''_i(\mathtt{t}_1, \ldots, \mathtt{t}_r)$$

for any $1 \leq i'' \leq n+1$ and *any* ground terms $\mathtt{t}''_1, \ldots, \mathtt{t}''_r$.

Now suppose that $\langle j, x_1, \ldots, x_r\rangle$ is a state of M reachable from an initial state; *i.e.* there is an initial state $\sigma = \langle 1, x', 0, \ldots, 0\rangle$ such that $\sigma \vdash^+_M \langle j, x_1, \ldots, x_r\rangle$. Then with respect to $\hat{P}_M$

$$\ell_1(\mathtt{s}^{x'}(\mathtt{0}), \mathtt{0}, \ldots, \mathtt{0}) \quad \text{depends on} \quad \ell_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0})).$$

Hence, with respect to Q_M

$$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{s}^j(\mathtt{0}), \mathtt{s}^{x_1}(\mathtt{0}), \ldots, \mathtt{s}^{x_r}(\mathtt{0}))$$

depends on

$$\ell'_1(\mathtt{s}^{x'}(\mathtt{0}),\mathtt{0},\ \ldots,\ \mathtt{0},\mathtt{s}^{j}(\mathtt{0}),\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))\,.$$

It follows that with respect to Q_M,

$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots\ ,\mathtt{s}^{x_r}(\mathtt{0}),\mathtt{s}^{j}(\mathtt{0}),\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))$

depends on

$\ell_j(\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))\,.$

We have just shown that with respect to Q_M

$\langle j, x_1,\ \ldots,\ x_r\rangle$ is a state of M reachable from an initial state

if, and only if,

$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}),\mathtt{s}^{j}(\mathtt{0}),\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))$

depends on

$\ell_j(\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))\,.$

Now, with respect to P_M,

$\ell_j(\mathtt{s}^{x_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))$

refers negatively to

$\ell'_k(\mathtt{s}^{x'_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x'_r}(\mathtt{0}),\mathtt{s}^{k}(\mathtt{0}),\mathtt{s}^{x'_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x'_r}(\mathtt{0}))$

if, and only if,

$\langle j, x_1,\ \ldots,\ x_r\rangle \ \vdash_M\ \langle k, x'_1,\ \ldots,\ x'_r\rangle\,.$

With respect to either Q_M or P_M,

if

$$\ell'_k(\mathtt{s}^{x'_1}(\mathtt{0}),\ \ldots\mathtt{x}\ \mathtt{s}^{x'_r}(\mathtt{0}),\mathtt{s}^{k}(\mathtt{0}),\mathtt{s}^{x'_1}(\mathtt{0}),\ \ldots,\ \mathtt{s}^{x'_r}(\mathtt{0}))$$

depends, for some m, positively on

$$\ell'_m(\mathtt{t}_1,\ \ldots,\ \mathtt{t}_r,\mathtt{v},\mathtt{w}_1,\ \ldots,\ \mathtt{w}_r)$$

then

$$\mathtt{v} \text{ is } \mathtt{s}^{k}(\mathtt{0}), \text{ and } \mathtt{w}_i \text{ is } \mathtt{s}^{x'_i}(\mathtt{0}) \text{ for } i = 1,\ \ldots,\ r.$$

Thus, with respect to P_M,

$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}),\ldots,\ \mathtt{s}^{x_r}(\mathtt{0}),\mathtt{s}^{j}(\mathtt{0}),\mathtt{s}^{x_1}(\mathtt{0}),\ldots,\ \mathtt{s}^{x_r}(\mathtt{0}))$

negatively depends directly on

$\ell'_k(\mathtt{s}^{x_1}(\mathtt{0}),\ldots,\ \mathtt{s}^{x_r}(\mathtt{0}),\mathtt{s}^{k}(\mathtt{0}),\mathtt{s}^{x'_1}(\mathtt{0}),\ldots,\ \mathtt{s}^{x'_r}(\mathtt{0}))$

if, and only if,

$\ell'_j(\mathtt{s}^{x_1}(\mathtt{0}), \dots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{s}^{j}(\mathtt{0}), \mathtt{s}^{x_1}(\mathtt{0}), \dots, \mathtt{s}^{x_r}(\mathtt{0}))$

depends on $\ell'_1(\mathtt{t}_1, \dots, \mathtt{t}_r, \mathtt{s}^{j}(\mathtt{0}), \mathtt{s}^{x_1}(\mathtt{0}), \dots, \mathtt{s}^{x_r}(\mathtt{0}))$, for some terms $\mathtt{t}_1, \dots, \mathtt{t}_r$

which refers to

$\ell_j(\mathtt{s}^{x_1}(\mathtt{0}), \dots, \mathtt{s}^{x_r}(\mathtt{0}))$

which negatively refers to

$\ell'_k(\mathtt{s}^{x_1}(\mathtt{0}), \dots, \mathtt{s}^{x_r}(\mathtt{0}), \mathtt{s}^{k}(\mathtt{0}), \mathtt{s}^{x'_1}(\mathtt{0}), \dots, \mathtt{s}^{x'_r}(\mathtt{0}))$

if, and only if,

$\langle j, x_1, \dots, x_r\rangle \vdash_M \langle k, x'_1, \dots, x'_r\rangle$
and $\langle j, x_1, \dots, x_r\rangle$ is reachable from some initial state

if, and only if,

$\langle j, x_1, \dots, x_r\rangle \vdash'_M \langle k, x'_1, \dots, x'_r\rangle$,

which proves the proposition. ■

Corollary 3.1: The following three statements are equivalent.

1. M always halts when started in an initial state
2. $\vdash'_M$ is well-founded
3. the negative dependency relation with respect to P_M is well-founded.

■

Proposition 3.2: The set of (Gödel numbers of) nondeterministic 4-register machines that always halt when started in an initial state is Π^1_1-complete.

Proof: Let α vary over countably infinite sequences of states of 4-register machines. If M is a nondeterministic 4-register machine then

$$\forall\, \alpha\, [\alpha(0) \text{ is an initial state } \Rightarrow\ \exists\, k \in \mathbb{N}\, [\alpha(k) \not\vdash_{\mathrm{M}} \alpha(k+1)]]$$

holds f, and only if, M always halts when started in an initial state. $\vdash_{\mathrm{M}}$ is decidable; thus the class of 4-register machines that always halt when started in an initial state is Π^1_1.

Let W_z be the recursively enumerable set with index z as defined in *e.g.* [Ro67]. Let $X \subseteq \mathbb{N}$ and let

$$\mathrm{Dom}(X) = \{y \mid W_y \subseteq X\}\,.$$

For $X \subseteq \mathbb{N}$ inductively define

$$\begin{aligned} \mathrm{Dom} \uparrow 0(X) &= X \\ \mathrm{Dom} \uparrow \delta(X) &= \textstyle\bigcup_{\gamma<\delta} \mathrm{Dom}(\mathrm{Dom} \uparrow \gamma(X)) . \end{aligned}$$

where γ and δ are ordinals.
By recursion-theoretic techniques once can show (*cf.* [Ro67, Bl82]) that the set of natural numbers

$$\mathrm{Dom} \uparrow \omega_1^{\mathrm{ck}}(\emptyset)$$

is Π_1^1-complete, where ω_1^{ck} is the least non-constructible ordinal.

Note that

$$n \in \mathrm{Dom} \uparrow \omega_1^{\mathrm{ck}}(\emptyset) \text{ iff } W_n \subseteq \mathrm{Dom} \uparrow \omega_1^{\mathrm{ck}}(\emptyset) .$$

Thus procedure W, given below, is guaranteed to halt, independently of the values chosen for natural number variable y during the computation, if, and only if, $x \in \mathrm{Dom} \uparrow \omega_1^{\mathrm{ck}}(\emptyset)$.

Let τ be a bijective pairing function on $\mathbb{N}$; e.g.

$$\tau(x,y) = \frac{1}{2}((x+y)^2 + 3x + y)$$

and let the inverses $(\cdot)_0$ and $(\cdot)_1$ of τ be defined by

$$(\tau(x,y))_0 = x \quad \text{and} \quad (\tau(x,y))_1 = y .$$

Kleene's well-known recursive relation T gives

$$n \in W_z \text{ iff } \exists y T(z,n,y)$$

Procedure W is then

```
begin /* procedure W */
input $x$;
choose arbitrary $y$;
while $T(x, (y)_0, (y)_1)$ do
$x := (y)_0$; choose arbitrary $y$ od
end.
```

The set of initial values input to the variable x in procedure W such that the procedure is guarenteed to halt, independently of the values chosen for y during the computation, is Π_1^1-complete.

Using the fact, discussed in section 2, that every binary partial recursive function can be computed by a four register machine program without having to encode the inputs as exponents, it easy to give a nondeterministic 4-register machine program M_W to implement procedure W. To complete the proof observe that for each $x \in \mathbb{N}$, we can specify a 4-register machine program $M_{W,x}$ that, when started with initial state $\langle 1, z, 0, 0, 0 \rangle$, replaces z by x and passes control to M_W. ■

Theorem 1.1 now follows from the previous corollary and proposition.

4 Conclusion

One can show that the perfect model of a stratified or locally stratified program is stable (*cf.* [GL88]), and that such programs have unique stable models. Marek, Nerode and Remmel [MNR90] proved that the class of stable models of a logic program is Π^0_2, from which it follows that a locally stratified program's unique stable model is necessarily Δ^1_1. Although beyond the scope of this paper, one can use techniques based on finite register machines similar to ones used here to show that every Δ^1_1 set of natural numbers is encodable, in a very direct way, as the true instances of a predicate in the unique stable model of a locally stratified program, and finally one can give a logic program which is not locally stratifiable that has stable models none of which are Δ^1_1, [BMS91]. Thus, the locally stratified logic programs index the hyperarithmetic sets.

Finally, we conjecture that the class of infinitely branching nondeterministic 2-register machines which always halt when started in an initial state is itself Π^1_1-complete; the proof of proposition 3.2 used four registers. Note that the number of registers used in that proof could be reduced to three if a pairing function could be implemented with three registers.

References

[ABW88] Apt, K. R., Blair, H. A., & Walker, A. "Towards a Theory of Declarative Knowledge," in *Foundations of Deductive Databases and Logic Programming,* Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988, pp. 89–148.

[AB90] Apt, K.R. and Blair, H.A., "Arithmetic Classification of Perfect Models of Stratified Programs", *Fundamenta Informaticae*, XIII, 1990, pp. 1–17.

[Bl82] Blair, H. A. "The Recursion-Theoretic Complexity of the Semantics of Predicate Logic as a Programming Language." *Information and Control*, July-August, 1982, pp. 25–47.

[BMS91] Blair, H., Marek, W. and Schlipf, J. Private communication.

[GL88] Gelfond, M. and Lifschitz, V. "The Stable Model Semantics for Logic Programming," *Proc. ICLP/SLP-5*, 1988, pp.1070–1080.

[Ko87] Kolaitis, P.G. "The Expressive Power of Stratified Logic Programs." Manuscript, Nov. 1987.

[MNR90] Marek, W., Nerode, A. and Remmel, J. *A Theory of Nonmonotonic Rule Systems*, MSI Technical Report 90-31, Mathematical Sciences Institute, Cornell University.

[Pr88] Przymusinski, T. "On the Declarative Semantics of Deductive Databases and Logic Programs," in *Foundations of Deductive Databases and Logic Programming,* Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988,

[Ro67] Rogers, H. *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

[Sh91] Shepherdson, J. C. *Unsolvable Problems for SLDNF-Resolution*, *J. of Logic Programming*, 10(1), Jan. 1991, pp. 19–22.

[SS63] Shepherdson, J. C. and Sturgis, H. E. "Computability of Recursive Functions," *JACM*, **10**, 217–255, 1963.

[VG88] Van Gelder, A. "Negation as Failure Using Tight Derivations for General Logic Programs," in *Foundations of Deductive Databases and Logic Programming,* Jack Minker, ed. Morgan-Kaufmann, Los Altos, CA. 1988, pp. 149–176.