Using logic programs to reason about infinite sets

D. Cenzer* V.W. Marek† J.B. Remmel‡

Abstract

Using the ideas from current investigations in Knowledge Representation we study the use of a class of logic programs for reasoning about infinite sets. Those programs assert codes for various infinite sets. Depending on the form of atoms allowed in the bodies of clauses we obtain a variety of completeness results for various classes of arithmetic sets of integers.

1 Introduction

The motivation for this paper comes out of recent progress in logical foundations of Artificial Intelligence and, in particular, the area of Knowledge Representation. In the past few years, there has been significant progress in the theory and practice of Logic Programming. In particular, a whole new area called *Answer Set Programming* (ASP) has arisen which can be viewed as a fusion of Logic Programming with Stable Model Semantics (SLP) and satisfiability (SAT). Answer Set Programming has emerged as both a theoretical and practical basis for the development of new generation of systems that are solidly grounded in the theory of Computer Science and capable of handling practical search problems arising in applications. The new generation of ASP systems such as *smodels*, `dlv`, and ASSAT [NSS99, EL+98, LZ02], which use both the native techniques of Logic Programming and the technology developed in SAT [MM+01, GN02], carry a lot of promise. Moreover, new types of constraints are introduced that allow for a more compact representation of problems. In such systems, the task of the programmer becomes easier because of the effort spent by the back-end processing engines.

The main motivation of this paper is to develop some extensions of the current ASP formalism that allows one to reason about infinite sets. The key idea is to use recursion theoretic techniques to reason about various types of indices of finite, recursive and r.e. sets. In particular, we develop a new extension of Logic Programming, called Extended Set Based (ESB) Logic Programming, which allows constraints expressed in terms of such indices. We shall also briefly analyze the complexity of the question of when such an ESB program has a recursive model.

We will introduce these new types of constraints below. However, first it will be good to recall the basic definitions of answer set programming and some of its recent extensions such as cardinality constraint logic programming [NSS99] and set constraint logic programming [MR03].

A logic programming clause is a construct of the form

$$C = p \leftarrow q_1, \ldots, q_m, not\ r_1, \ldots, not\ r_n$$

*Department of Mathematics, University of Florida, `cenzer@math.ufl.edu`. Corresponding author
†Department of Computer Science, University of Kentucky
‡Department of Mathematics, University of California, San Diego

where $p, q_1, \ldots, q_m, r_1, \ldots, r_n$ are atoms. A logic program is a set of logic programming clauses. The atoms $q_1, \ldots, q_m, not\ r_1, \ldots, not\ r_n$ form the *body* of C and the atom p is its *head*. A model of a clause C is a set of atoms M such that whenever M satisfies the body of C, then M also satisfies the head of C. The clauses C where $n = 0$ are called *Horn* clauses. A program entirely composed of Horn clauses is called a Horn program and a Horn program always has a least model. It is the intended semantics of such program. For programs with bodies containing the negation operator *not* , we will use the stable model semantics. Following [GL88], we define a *stable model* of the program as follows. Assume M is a collection of atoms. The *Gelfond-Lifschitz reduct* of P by M is a Horn program arising from P by eliminating those clauses in P which contain *not* r with $r \in M$. In the remaining clauses, we drop all negative literals from the body. The resulting program $GL_M(P)$ is a Horn program. We call M a stable model of P if M is the least model of $GL_M(P)$. In the case of a Horn program, there is a unique stable model, namely, the least model of P.

It is the general consensus of the Knowledge Representation community that stable models are the intended models of logic programs. Once such a consensus emerged, it was natural for both theoreticians and logicians to study various complexity issues associated with stable models of logic programs. There has been extensive effort of the community to investigate both the theoretical issues associated with stable models and the practical algorithms for processing.

The most general case of stable models of logic programs, those allowing for function symbols in atoms, turned out to be very hard. Recall that Horn programs have a least model which is r.e. in the (code for) program. Starting with [AB90] and continuing with [BMS95] and [MNR94], a number of results showed that the stable models of logic programs that allow function symbols could be exceedingly complex, even in the case when there is a unique stable model. In particular Marek, Nerode and Remmel [MNR94] showed that there exist finite predicate logic programs which have stable models but which have no hyperarithmetic stable model. While these results may at first glance appear negative, they had a positive result in the long run since they forced researchers and designers to limit themselves to cases where programs can be actually processed. The effect was that processing programs (called *solvers*) had to focus on finite programs that do not admit function symbols.

The designers of the solvers have also focused on the issues of both improving processing of the logic programs (i.e. searching for a stable model) and improving the use of logic programs as a programming language. The latter task consists of extending the constructs available to the programmer to make programming easier and more readable. Various researchers discovered that it was possible to introduce meaningful extensions to the logic programming syntax and yet have such extensions be processed in a manner which is entirely analogous to the processing currently employed in case of logic programs proper. For example, let ω denote the set of natural numbers. Then a *cardinality constraint atom* is a constraint of the form kXl where X is a finite set of atoms and k and l are elements of $\omega \cup \{\infty\}$ such that $k \leq |X| \leq l$. The *meaning* of such an atom is that a putative model M satisfies kXl, written $M \models kXl$, if and only if $k \leq |M \cap X| \leq l$. These atoms and related *weight constraint atoms*, where we have some weight function wt on X and a model M satisfies a constraint $l \leq X \leq u$ if and only if $l \leq \sum_{p \in M \cap X} wt(p) \leq u$, are special cases of more general *set constraint atoms* of the form $\langle X, \mathcal{F} \rangle$ where $\mathcal{F} \subseteq 2^X$. Here we say M satisfies $\langle X, \mathcal{F} \rangle$ if and only if $M \cap X \in \mathcal{F}$. Many types of constraints can be expressed in the form $\langle X, \mathcal{F} \rangle$. For instance various constraints used by SQL query language can be so represented. Set constraints have been introduced and investigated in [MR03].

Formally a set-constraint clause (or sc-clause) is an expression of the form

$$\langle X, \mathcal{F} \rangle \leftarrow \langle X_1, \mathcal{F}_1 \rangle, \ldots, \langle X_m, \mathcal{F}_m \rangle$$

It is easy to see that ordinary logic programming can be reduced to set-constraint programming. That is, the meaning of atom a is the same as that of set constraint $\langle\{a\},\{\{a\}\}\rangle$ and the meaning of *not* a is the same as $\langle\{a\},\{\emptyset\}\rangle$. Our definition of stable model is an extension of the version of the Gelfond-Lifschitz transform introduced by Niemelä, Simons and Soininen [NSS99] for cardinality constraint programs and we call it the NSS transform. Again the process of constructing the model is based on some form of "Horn" programs, reduction, and least fixed points of the one-step provability operators for Horn programs. First, a family $\mathcal{F}$ of subsets of X is *upper closed* if $Y \subseteq Z \subseteq X$ and $Y \in \mathcal{F}$ implies $Z \in \mathcal{F}$. We will call an sc-clause *Horn* if

1. the head of that clause is a single atom (recall that atoms are represented as set constraints) and
2. whenever $\langle X_i, \mathcal{F}_i\rangle$ appears in the body, then $\mathcal{F}_i$ is an upper closed family of subsets of X_i.

A set-constraint Horn program P is a set-constraint program which consist entirely of Horn clauses. There is a natural one-step provability operator associated to a sc-Horn program P, $T_P : 2^X \to 2^X$ where X is the underlying set of atoms of the program, defined by $T_P(S)$ equals the set of all p such that there is clause

$$C = p \leftarrow \langle X_1, \mathcal{F}_1\rangle, \ldots, \langle X_m, \mathcal{F}_m\rangle \in P$$

such that S satisfies the body of C. It is easy to see that our definitions ensure that T_P is a monotone operator and hence each sc-Horn program has a least model. It can be computed in a manner analogous to the computation of the least model of a definite Horn program as $T_P \uparrow^\omega (\emptyset)$. The NSS transform $NSS_M(P)$ of the set-constraint program P for a given set of atoms M is defined as follows. First eliminate all clauses with bodies not satisfied by M. Next, for each remaining clause $\langle X, \mathcal{F}\rangle \leftarrow \langle X_1, \mathcal{F}_1\rangle, \ldots, \langle X_m, \mathcal{F}_m\rangle$ and each $p \in M \cap X$ put the clause $p \leftarrow \langle X_1, \overline{\mathcal{F}}_1\rangle, \ldots, \langle X_m, \overline{\mathcal{F}}_m\rangle$ into $NSS_M(P)$. Here $\overline{\mathcal{F}_i}$ is the least family $\mathcal{G}$ containing $\mathcal{F}_i$ and closed upwards. Clearly the resulting program $NSS_M(P)$ is a sc-Horn program and hence has a least model $M^{NSS_M(P)}$. M is a stable model of P if $M = M^{NSS_M(P)}$. It can be shown [NSS99] that this construction corresponds to the same notion of Gelfond-Lifschitz stable models when we restrict ourselves to ordinary logic programs.

In this paper we would like to use the mechanism of set-constraints to reason about *infinite* sets. But how this could be done? Certainly we can not write an entire infinite set. Nevertheless, there are means to *encode* infinite sets by finite means, for instance, various types of *indices* or codes.

So here is the idea. Assume that we have some particular coding scheme for some family of subsets of set X. Let $\mathcal{F}$ be a finite family of such codes. We will write F_e for the set with the code e. Then we can write two types of constraints. One constraint $\langle X, \mathcal{F}\rangle^{\subseteq}$ has the meaning that the putative set of integers M satisfies $\langle X, \mathcal{F}\rangle^{\subseteq}$ if and only if $M \cap X \supseteq F_e$ for some $e \in \mathcal{F}$. Similarly, we shall also consider constraints of the form $\langle X, \mathcal{F}\rangle^{=}$ where we say that M satisfies $\langle X, \mathcal{F}\rangle^{=}$ if and only if $M \cap X = F_e$ for some $e \in \mathcal{F}$. Observe that the constraints of the form $\langle X, \mathcal{F}\rangle^{\subseteq}$ behave like atoms p in that they are preserved when the set grows while the constraints of the form $\langle X, \mathcal{F}\rangle^{=}$ behave more like constraints *not* p in that they are not always preserved as the set grows. Now, it is clear that once we introduce these type of constraint schemes, we can consider various coding schemes for the set of indices. For example, in this paper, we will consider three such schemes: explicit indices of finite sets, recursive indices of recursive sets and r.e. indices of r.e. sets.

We shall then define an extended set-based clause C to be a clause of the form

$$\langle X, \mathcal{A}\rangle^* \leftarrow \langle Y_1, \mathcal{B}_1\rangle^{\subseteq}, \ldots, \langle Y_k, \mathcal{B}_k\rangle^{\subseteq}, \langle Z_1, \mathcal{C}_1\rangle^{=}, \ldots, \langle Z_l, \mathcal{C}_l\rangle^{=}. \quad (1)$$

where $*$ is either $=$ or $\subseteq$ and define an *extended set based program* (ESB) P to be a set of extended set based clauses.

2 ESB Constraints, Clauses and Programs

In this section, we shall give the formal definitions of ESB constraints, clauses, programs and define the analogue of Horn programs and stable models for ESB programs. To describe our constraints, we first need to describe three different types of indices for subsets of the natural numbers:
(1) **Explicit indices of finite sets**. For each finite set $F \subseteq \omega$, we shall define the explicit index of F as follows. The explicit index of the empty set is 0 and the explicit index of $\{x_1 < \ldots < x_m\}$ is $2^{x_1} + \cdots + 2^{x_n}$. We shall let F_n denote the finite set whose index is n.
(2) **Recursive indices of recursive sets**. Let $\phi_0, \phi_1, \ldots$, be an effective list of all partial recursive functions. By a recursive index of a recursive set R, we mean an e such that ϕ_e is the characteristic function of R. If ϕ_e is a total $\{0,1\}$-valued function, then R_e will denote the set $\{x \in \omega : \phi_e(x) = 1\}$.
(3) **R.e. indices of r.e. sets**. By a r.e. index of a r.e. set W, we mean an e such that W equals the domain of ϕ_e, that is, $W_e = \{x \in \omega : \phi_e(x) \text{ converges}\}$.
No matter what type of indices we use, we shall always consider two types of constraints based on X and a set of indices $\mathcal{F}$, namely, $\langle X, \mathcal{F}\rangle^=$ and $\langle X, \mathcal{F}\rangle^\subseteq$. For any subset $M \subseteq \omega$, we shall say that M *is a model of* $\langle X, \mathcal{F}\rangle^=$, written $M \models \langle X, \mathcal{F}\rangle^=$, if there exists an $e \in \mathcal{F}$ such that $M \cap X$ equals that set with index e. Similarly, we shall say that M *is a model of* $\langle X, \mathcal{F}\rangle^\subseteq$, written $M \models \langle X, \mathcal{F}\rangle^\subseteq$, if there exists an $e \in \mathcal{F}$ such that $M \cap X$ contains the set with index e.

Fix some recursive pairing function $[,] : \omega \times \omega \to \omega$. For any sequence $a_1, \ldots, a_n$, with $n \geq 2$, we define the code $c(a_1, \ldots, a_n)$ by the usual inductive procedure of defining $c(a_1, a_2) = [a_1, a_2]$ and $c(a_1, \ldots, a_n) = [a_1, c(a_2, \ldots, a_n)]$ if $n \geq 3$. The explicit index of the sequence $\vec{s} = (a_1, \ldots, a_n)$, $ind(a_1, \ldots, a_n)$ is defined by induction. If $n = 2$, then $ind(a_1, a_2) = [2, [a_1, a_2]]$ and if $n \geq 3$, then $ind(a_1, \ldots, a_n) = [n, c(a_1, \ldots, a_n)]$. In this paper, we shall consider three different types of constraints.
(A) **finite constraints**: Here we assume that we are given an explicit index x of a finite set X and a finite family $\mathcal{F}$ of explicit indices of finite subsets of X. Throughout this paper we shall identify the finite constraints $\langle X, \mathcal{F}\rangle^=$ and $\langle X, \mathcal{F}\rangle^\subseteq$ with their codes, $ind(0, 0, x, n)$ and $ind(0, 1, x, n)$ respectively where $\mathcal{F} = F_n$. Here the first coordinate 0 tells that the constraint is finite, the second coordinate is 0 or 1 depending on whether the constraint is $\langle X, \mathcal{F}\rangle^=$ or $\langle X, \mathcal{F}\rangle^\subseteq$, and the third and fourth coordinates code X and $\mathcal{F}$ respectively.
(B) **recursive constraints**: Here we assume that we are given a recursive index x of a recursive set X and a finite family $\mathcal{R}$ of recursive indices of recursive subsets of X. Again we shall identify the recursive constraints $\langle X, \mathcal{R}\rangle^=$ and $\langle X, \mathcal{R}\rangle^\subseteq$ with their codes, $ind(1, 0, x, n)$ and $ind(1, 1, x, n)$ respectively, where $\mathcal{R} = F_n$. Here the first coordinate 1 tells that the constraint is recursive, the second coordinate is 0 or 1 depending on whether the constraint is $\langle X, \mathcal{R}\rangle^=$ or $\langle X, \mathcal{R}\rangle^\subseteq$, and the third and fourth coordinates code X and $\mathcal{R}$ respectively.
(C) **r.e. constraints**: Here we are given a r.e. index x of a r.e. set X and a finite family $\mathcal{W}$ of r.e. indices of r.e. subsets of X. Again we shall identify the finite constraints $\langle X, \mathcal{W}\rangle^=$ and $\langle X, \mathcal{W}\rangle^\subseteq$ with their codes, $ind(2, 0, x, n)$ and $ind(2, 1, x, n)$ respectively, where $\mathcal{W} = F_n$. Here the first coordinate 2 tells that the constraint is r.e., the second coordinate is 0 or 1 depending on whether the constraint is $\langle X, \mathcal{W}\rangle^=$ or $\langle X, \mathcal{W}\rangle^\subseteq$, and the third and fourth coordinates code X and $\mathcal{W}$ respectively.

Next we define an extended set-based clause C to be a clause of the form

$$\langle X, \mathcal{A}\rangle^* \leftarrow \langle Y_1, \mathcal{B}_1\rangle^\subseteq, \ldots, \langle Y_k, \mathcal{B}_k\rangle^\subseteq, \langle Z_1, \mathcal{C}_1\rangle^=, \ldots, \langle Z_l, \mathcal{C}_l\rangle^=. \tag{2}$$

where $*$ is either $=$ or $\subseteq$. We shall refer to $\langle X, \mathcal{A}\rangle^*$ as the head of C, written $head(C)$, and $\langle Y_1, \mathcal{B}_1\rangle^\subseteq, \ldots, \langle Y_k, \mathcal{B}_k\rangle^\subseteq, \langle Z_1, \mathcal{C}_1\rangle^=, \ldots, \langle Z_l, \mathcal{C}_l\rangle^=$ as the body of C, written $body(C)$. Here either k or l may be 0. M is said to be a model of C if either M does not model every constraint in $body(C)$ or $M \models head(C)$.

Again we shall talk about three different types of clauses.
(a) **finite clauses**: These are clauses in which all of the constraints are finite constraints.
(b) **recursive clauses**: These are clauses where all the constraints appearing in the clause are finite or recursive constraints and at least one constraint is a recursive constraint.
(c) **r.e. clauses**: These are clauses where all the constraints appearing in the clause are finite, recursive or r.e. constraints and there is at least one r.e. constraint.

An extended set-based (ESB) program P is a set of clauses of the form of (2). We say that an ESB program P is recursive, if the set of codes of the clauses in P form a recursive set where the code of a clause C of the form of (1) is $ind(c, e_1, \ldots, e_k, f_1, \ldots, f_l)$ where c is the code of $\langle X, \mathcal{A}\rangle^*$, e_i is the code of $\langle Y_i, \mathcal{B}_i\rangle^{\subseteq}$ for $i = 1, \ldots, k$ and f_j is the code of $\langle Z_j, \mathcal{C}_j\rangle^{=}$ for $j = 1, \ldots, l$.

Given a program P, we let $Fin(P)$ ($Rec(P)$, $RE(P)$) denote the set of all finite (recursive, r.e.) clauses in P. It is easy to see from our coding of clauses that if P is a recursive ESB program, then $Fin(P)$, $Rec(P)$ and $RE(P)$ are also recursive ESB programs.

We will say that a program P is *recursive with finite constraints* if P is a recursive program such that $P = Fin(P)$. Similarly we say that a program P is *recursive with recursive constraints* if $P = Fin(P) \cup Rec(P)$ and $Rec(P) \neq \emptyset$ and a program P is *recursive with r.e. constraints* if P is a recursive program and $RE(P) \neq \emptyset$. Finally we say that P is *weakly finite with recursive constraints* if P is a recursive program with recursive constraints and the set of heads of clauses in $Rec(P)$ is finite and P is *weakly finite with r.e. constraints* if P is a recursive program with r.e. constraints and the set of heads of clauses in $Rec(P) \cup RE(P)$ is finite.

Next we define the analogue of Horn programs for ESB programs. A Horn program P is a set of clauses of the form

$$\langle X, \mathcal{A}\rangle^{\subseteq} \leftarrow \langle Y_1, \mathcal{B}_1\rangle^{\subseteq}, \ldots, \langle Y_k, \mathcal{B}_k\rangle^{\subseteq}. \tag{3}$$

where $\mathcal{A}$ is a singleton. We define the *one-step provability operator*, $T_P : 2^\omega \to 2^\omega$ so that for any $S \subseteq \omega$, $T_P(S)$ is the union of the set of all D_e such that there exists a clause $C \in P$ such $S \models body(C)$, $head(C) = \langle X, \mathcal{A}\rangle^{\subseteq}$ and $A = \{e\}$ where $D_e = F_e$ if C is a finite clause, $D_e = R_e$ is C is a recursive clause, and D_e is W_e if C is an r.e. clause. It is easy to see that T_P is a monotone operator and hence there is a least fixed point which we denote by M_P. Moreover it is easy to check that M_P is a model of P.

If P is an ESB Horn program in which the body of every clause consists of *finite* constraints, then one can easily show we reach the least fixed point of T_P in ω-steps, that is, $M_P = T_P \uparrow^\omega (\emptyset)$. However, if we allow clauses whose bodies contain either recursive or r.e. constraints of the form $\langle X, \mathcal{G}\rangle^*$ where X is infinite and $*$ is either $=$ or $\subseteq$, then we can no longer guarantee that we reach the least fixed point of T_P in ω steps. To this end consider the following example.

Example 2.1 Let e_n be the explicit index of the set $\{n\}$ for all $n \geq 0$, let w be a recursive index of ω and f be a recursive index of the set of even numbers E. Consider the following program.

$$\begin{aligned}
\langle\{0\}, \{e_0\}\rangle^{\subseteq} &\leftarrow \\
\langle\{2x+2\}, \{e_{2x+2}\}\rangle^{\subseteq} &\leftarrow \langle\{2x\}, \{e_{2x}\}\rangle^{\subseteq} \quad \text{(for every even number } x\text{)} \\
\langle\omega, \{w\}\rangle^{\subseteq} &\leftarrow \langle E, \{f\}\rangle^{\subseteq}
\end{aligned}$$

Clearly ω is the least model of P but it takes $\omega + 1$ steps to reach the fixed point. That is, it is easy to check that $T_P \uparrow^\omega = E$ and that $T_P \uparrow^{\omega+1} = \omega$

Theorem 2.1 (a) *If P is a recursive ESB Horn Program with finite constraints, then the least fixed point of the one step provability T_P is r.e..*

(b) *If P is a weakly finite ESB Horn program with recursive constraints such that $Fin(P)$ is recursive, then the least fixed point of the one step provability operator T_P is r.e..*

(c) *If P is a weakly finite program with r.e. constraints such that $Fin(P)$ is recursive, then the least fixed point of the one step provability operator T_P is r.e..*

Proof. Part (a) is essentially the usual proof that the least fixed point of a recursive Horn program is r.e..

For part (b), we note that we can construct the least fixed point as follows.
Step 1) First take $Fin(P)$ and construct the least fixed point which we will call U_0. Since $Fin(P)$ is recursive, U_0 is r.e.. Next consider the set $T_1 = U_0 \cup S_0$ where S_0 is the union of the set of all R_e such that there exists a clause $C \in Rec(P)$ such that $U_0 \models body(C)$ and $head(C) = \langle X, \mathcal{R} \rangle$ where $\mathcal{R} = \{e\}$. Even though we cannot find S_0 recursively, our hypothesis ensures that S_0 is a finite union of recursive sets and hence is a recursive set. Thus T_1 is an r.e. set. Now if $S_0 = \emptyset$, then stop; T_1 is the least fixed point of P. Otherwise go onto step (2).

Step (n+1) Consider the set $U_n = T_{Fin(P)} \uparrow^\omega (T_n)$. It is easy to see that that since T_n is r.e., U_n is r.e.. Next consider the set $T_{n+1} = U_n \cup S_n$ where S_n is the union of the set of all R_e such that there exists a clause $C \in Rec(P)$ such that $U_n \models body(C)$ and $head(C) = \langle X, \mathcal{R} \rangle$ where $\mathcal{R} = \{e\}$. Even though we cannot find S_n recursively, our hypothesis ensures that S_n is a finite union of recursive sets and hence is a recursive set. Thus T_{n+1} is an r.e. set. Now if $S_n = \emptyset$, then stop; T_{n+1} is the least fixed point of P. Otherwise go onto step (n+2).

Since the set of all $head(C)$ such $C \in Rec(P)$ is finite, it easily follows that this process must stop after a finite number of steps and hence the least model of P is r.e..

The proof of part (c) is similar to the proof of part (b). □

We note that there is an alternative way to obtain the least model of weakly finite ESB program with recursive or r.e. constraints. Namely, if P is a recursive weakly finite program with either recursive or r.e. constraints, let $\mathcal{H}(P)$ denote the set of all $head(C)$ where $C \in Rec(P) \cup RE(P)$. By definition, $\mathcal{H}(\mathcal{P})$ is a finite set consisting of constraints of the form $C_{X,e} = \langle X, \mathcal{A} \rangle^{\subseteq}$ where $A = \{e\}$. In such a situation, we let $S_{C_{X,e}} = R_e$ if e is a recursive index and $S_{C_{X,e}} = W_e$ if e is an r.e. index. If $S \subseteq \mathcal{H}(P)$, then let $U_S = \bigcup_{C_{X,e} \in S} S_{C_{X,e}}$. Then the least model M of P is of the form

$$M = T_{Fin(P)} \uparrow^\omega (U_S)$$

for some $S \subseteq \mathcal{H}(P)$.

The hypothesis that the P is a weakly finite Horn program with recursive or r.e. constraints is absolutely necessary for the proof of Theorem 2.1 as our next example will show.

Example 2.2 Suppose that we are given a sequence of pairwise disjoint infinite recursive sets $Y, X_0, A_0, X_1, A_1, \ldots$. Let $Y = \{y_0 < y_1 < \ldots\}$, $X_e = \{x_{0,e} < x_{1,e} < \ldots\}$ for each $e \in \omega$ and $A_e = \{a_{0,e} < a_{1,e} < \ldots\}$ for each $e \in \omega$. For all $k \geq 0$, we shall let $X_{e,\geq k} = \{x_{k,e} < x_{k+1,e} < \cdots\}$.

Given an atom a, the finite set constraint $\langle \{a\}, \{n\} \rangle^{\subseteq}$ where $F_n = \{a\}$ is satisfied by a model M iff $a \in M$ so that the set constraint $\langle \{a\}, \{n\} \rangle^{\subseteq}$ acts like a atom in a normal logic program. Thus in the following program, we shall abbreviate that finite set constraint $\langle \{a\}, \{n\} \rangle^{\subseteq}$ by the atom a.

Let W_e^s denote the finite set of elements z less than or equal to s such that $\phi_e(z)$ converges in s or fewer steps. Now consider the following program.

(1) $x_{n,e} \leftarrow a_{[n,s],e}$ for all n such that $n \in W_e^s - W_e^{s-1}$
(2) $a_{[n,s],e} \leftarrow$ for all n such that $n \in W_e^s - W_e^{s-1}$
(3) $y_e \leftarrow \langle X_e, \{n_k\}\rangle^{\subseteq}$ for all $k \geq 0$ where n_k is a recursive index of $X_{e,\geq k}$.

It is now easy to see that P is a recursive ESB Horn program such that the least model of P equals $\{y_e : W_e \text{ is cofinite}\} \cup \{x_{n,e} : n \in W_e\}$. However it is known [So87] that the set $\{e : W_e \text{ is cofinite}\}$ is a complete Σ_3^0 set so that M is a complete Σ_3^0 and hence is certainly not r.e.

Finally, we can define the analogue of a stable model for ESB programs. Given a model M and an ESB program P, we define the analogue of the GL-transform by saying that $GL_M(C)$, where $C \in P$ is a clause of the form (1), is *nil* if M does not satisfy the body of C and is

$$\langle X, \mathcal{F}'\rangle^{\subseteq} \leftarrow \langle Y_1, \mathcal{B}_1\rangle^{\subseteq}, \ldots, \langle Y_k, \mathcal{B}_k\rangle^{\subseteq}, \langle Z_1, \mathcal{C}_1\rangle^{\subseteq}, \ldots, \langle Z_l, \mathcal{C}_l\rangle^{\subseteq}. \tag{4}$$

if M does satisfy the body of C where $\mathcal{F}' = \{e\}$ and e is an explicit (recursive, r.e.) index of $X \cap M$ if $\langle X, A\rangle > *$ is a finite (recursive, r.e.) constraint. Then $GL_M(P) = \{GL_M(C) : C \in P\}$ will be a Horn program. We then say that M is a *stable model of* P if M is a model of P and M equals the least model of $GL_M(P)$.

3 Complexity of least models of ESB Horn programs

There are many complexity issues that need to be explored with respect of ESB programs. In this section, we shall content ourselves with stating a few results about the complexity of ESB Horn programs. The arithmetic hierarchy is defined as usual so that the recursive sets are both Σ_0^0 and Π_0^0, a Σ_{n+1}^0 set is obtained by existential number quantification over a Π_n^0 set, and a Π_{n+1}^0 set is the complement of a Σ_{n+1}^0 set. In particular, a set of natural numbers is Σ_1^0 if and only if it is an r.e. set.

Theorem 3.1 *For any arithmetic set A, there is a recursive ESB Horn program P_A with least fixed point is of degree A.*

Theorem 3.2 *Let M be a recursive set.*

(a) If P is a recursive ESB Horn program with finite constraints, then the predicate 'M is the least model of P' is Π_2^0.

(b) If P is a recursive weakly finite ESB Horn program with recursive constraints, then the predicate 'M is the least model of P' is Σ_3^0.

(c) If P is a recursive weakly finite ESB Horn program with r.e. constraints, then the predicate 'M is the least model of P' is Σ_4^0 and in fact a Boolean combination of Σ_3^0 predicates.

It follows from Theorem 3.2 that the predicate that a recursive weakly finite ESB Horn program P with recursive constraints has recursive least model is Σ_3^0. In fact, we can prove the following.

Theorem 3.3 *Let P be a recursive weakly finite ESB Horn program with recursive constraints, then the predicate 'P has a recursive stable model' is Σ_3^0-complete.*

It turns our that there is a difference between recursive weakly finite ESB programs with r.e. constraints and recursive weakly finite ESB programs with recursive constraints. To give a more refined analysis for recursive weakly finite programs with r.e. constraints, let us say that a recursive Horn program P with r.e. constraints is **n-weakly finite** if there are n different heads of clauses from $P - Fin(P)$. Next we define the difference of two Σ^0_3 sets, i.e. the conjunction of Σ^0_3 set with a Π^0_3 set, to be 1-Σ^0_3 set. Then by induction, we say that a set C is n-Σ^0_3 if $C = A - B$ where A is Σ^0_3 and B is a $(n-1)$-Σ^0_3 set. We can then prove the following.

Theorem 3.4 *Let M be a recursive set and let P be a recursive n-weakly-finite ESB Horn program with r.e. constraints. Then the predicate 'M is the least model of P' is a $(2^{n+1}-1)$-Σ^0_3 predicate.*

Finally, we can give a completeness result for the case of $n = 1$.

Theorem 3.5 *If P is a recursive 1-weakly-finite ESB Horn program with r.e. constraints, then the predicate 'P has a recursive stable model' is $3 - \Sigma^0_3$-complete.*

References

[AB90] K. Apt and H.A. Blair. Arithmetical classification of perfect models of stratified programs. *Fundamenta Informaticae*, 12:1 – 17, 1990.

[BMS95] H.A. Blair, V.W. Marek, and J. Schlipf. The expressiveness of locally stratified programs, *Annals of Mathematics and Artificial Intelligence* 15(2):209–229, 1995.

[EL+98] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR System dlv: Progress Report, Comparisons, and Benchmarks. In *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 406–417, 1998.

[GL88] M. Gelfond and V. Lifschitz. The stable semantics for logic programs. In R. Kowalski and K. Bowen, editors, ICLP88, pages 1070–1080, 1988.

[GN02] E. Goldberg, Y. Novikov. BerkMin: a Fast and Robust SAT-Solver. DATE-2002, pages 142–149, 2002.

[LZ02] F. Lin and Y. Zhao, ASSAT: Computing answer sets of a logic program by SAT solvers, AAAI-2002, pages 112–117, 2002.

[MNR94] W. Marek, A. Nerode, and J. B. Remmel. The stable models of predicate logic programs. *Journal of Logic Programming*, 21(3):129–154, 1994.

[MR03] V.W. Marek and J.B. Remmel, Set Constraints in Logic Programming, LPNMR03, *Accepted for publication.*

[MM+01] M.W. Moskewicz, C.F. Magidan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver, SAT 2001, 2001.

[NSS99] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. LPNMR99, Springer LN in Computer Science 1730, pages 317–331, 1999.

[So87] R.I. Soare. *Recursively Enumerable Sets and Degrees.* Springer Verlag, 1987.