Effectively Reasoning about Infinite Sets in Answer Set Programming[⋆]

Victor Marek[1] and Jeffrey B. Remmel[2]

[1] Department of Computer Science, University of Kentucky, Lexington, KY 40506 [⋆⋆]
[2] Departments of Mathematics and Computer Science, University of California at San Diego, La Jolla, CA 92903 [⋆⋆⋆]

Abstract. In answer set programming (ASP), one does not allow the use of function symbols. Disallowing function symbols avoids the problem of having logic programs which have stable models of excessively high complexity. For example, Marek, Nerode, and Remmel showed that there exist finite predicate logic programs which have stable models but which have no hyperarithmetic stable model. Disallowing function symbols also avoids problems with the occurs check that can lead to wrong answers in logic programs. Of course, by eliminating function symbols, one loses a lot of expressive power in the language. In particular, it is difficult to directly reason about infinite sets in ASP.

Blair, Marek, and Remmel [BMR08] developed an extension of logic programming called *set based logic programming*. In the theory of set based logic programming, the atoms represent subsets of a fixed universe X and one is allowed to compose the one-step consequence operator with a monotonic idempotent operator (miop) O so as to ensure that the analogue of stable models are always closed under O. We let $\mathcal{S}_P$ denote the set of fixed points of finite unions of the sets represented by the atoms of P under the miops associated with P. We shall show that if there is a coding scheme which associates to each element $A \in \mathcal{S}_P$ a code $c(A)$ such that there are effective procedures, which given two codes $c(A)$ and $c(B)$ of elements $A, B \in \mathcal{S}_P$, will (i) decide if $A \subseteq B$, (ii) decide if $A \cap B = \emptyset$, and (iii) produce the codes of the closures of $A \cup B$ and of $A \cap B$ under the miop operators associated with P, then we can effectively decide whether an element $A \in \mathcal{S}_P$ is a stable model of P. Thus in such a situation, we can effectively reason about the stable models of P even though $\mathcal{S}_P$ contains infinite sets. Our basic example is the case where all the sets represented by the atoms of P are regular languages but many other examples are possible such as when the sets involved are certain classes of convex sets in $\mathbb{R}^n$.

[⋆] This is an updated and expanded version of [MR09].
[⋆⋆] email:marek@cs.uky.edu.edu
[⋆⋆⋆] email: jremmel@ucsd.edu

1 Introduction

Computer Science for the most part reasons about *finite* sets, relations and functions. There are many examples in computer science where adding symbols for infinite sets or arbitrary function symbols into programming languages results in big jumps in the complexity of models of programs. For example, finding the least model of a finite Horn program with no function symbols can be done in linear time [DG82] while the least model of finite predicate logic Horn program with function symbols can be an arbitrary recursively enumerable set [Sm68]. If we consider logic programs with negation, Marek and Truszczyński [MT93] showed that the question of whether a finite propositional logic program has a stable model is NP-complete. However Marek, Nerode, and Remmel [MNR94] showed that the question of whether a finite predicate logic program with function symbols possesses a stable model is Σ^1_1 complete. Similarly, the stable models of logic programs that contain function symbols can be quite complex. 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 can be exceedingly complex, even in the case where the program has a unique stable model. For example, Apt and Blair [AB90] have shown that arithmetic sets can be defined with stable models of stratified programs and 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 type of results may at first glance appear negative, they had a positive effect 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* such as cmodels [BL02,GLM06], smodels [SNS02], *clasp* [GKN+], ASSAT [LZ02], and `dlv` [LPF+06] had to focus on finite programs that do not admit function symbols (`dlv` allows for use of a very limited class of programs with function symbols). The designers of 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. This development resulted in a class of solvers that found use in combinatorial optimization [MT99,Nie99], hardware verification [KN03], product configuration [SNTS01], and other applications.

Of course, by eliminating function symbols, one loses a lot of expressive power in the language. One of the motivations of this paper was to find ways to extend the ASP formalism to allow one to reason *directly* about infinite sets yet still allow the programs to be processed in an effective manner. This requires a very careful analysis of the complexity issues involved in the formalisms as well as developing various ways to code the infinite sets involved in any given application so that one can process information effectively. Part of our motivation is that with the rise of the Internet, there are now many tools which use the Internet as a virtual data base. While all the information on the Internet at any given point of time is a finite object, it is constantly changing and it would be nearly impossible to characterize the totality of information available in any meaningful way. Thus, for all practical purposes, one can consider the information on the Internet as an infinite set of information items. Hence we need to consider ways in which one can extend various formalisms in computer science to reason about infinite objects.

The main goal of this paper is to show that there are extensions of the ASP formalism where one can effectively reason about infinite languages which are accepted by deterministic finite automata (DFAs). In particular, we shall show that in a recent extension of logic programming due to Blair, Marek, and Remmel [BMR08], one can effectively reason about languages which are accepted by finite automaton. We will also show that under reasonable assumptions, this approach can be lifted to other areas as well.

In [BMR01], Blair, Marek, and Remmel developed an extension of the logic programming paradigm called *spatial logic programming* in which one can directly reason about regions in space and time as might be required in applications like graphics, image compression, or job scheduling. In spatial logic programming, one has some fixed space X be the intended universe of the program rather than having the Herbrand base be the intended underlying universe of the program and one has each atom of the language of the program specify a subset of X, i.e. an element of the set 2^X.

As pointed out in [BMR08], if one reflects for a moment on the basic aspects of logic programming with an Herbrand model interpretation, a slight change in one's point of view shows that it is natural to interpret atoms as subsets of the Herbrand base. In ordinary logic programming, we determine the truth value of an atom p in an Herbrand interpretation I by declaring $I \models p$ if and only if $p \in I$. However, this is equivalent to defining the *sense*, $[\![p]\!]$, of an atom p to be the set $\{p\}$ and declaring that $I \models p$ if and only if $[\![p]\!] \subseteq I$. By this simple move, we have permitted ourselves to interpret the sense of an atom as a subset of a set X rather than the literal atom itself in the case where X is the Herbrand base of the language of the program.

It turns out that if the underlying space X has structure such as a topology or an algebraic structure such as a group or vector space, then a number of other natural options present themselves. That is, Blair, Marek, and Remmel [BMR08] extended the theory of spatial logic programming to what they called *set based logic programming* where one composes the one-step consequence operator of spatial logic programing with a monotonic idempotent operator. For example, if we are dealing with a topological space, one can construct a new one-step consequence operator T by composing the one-step consequence operator for spatial logic programming with an operator that produces the topological closure of a set or the interior of a set. In such a situation, we can ensure that the new one-step consequence operator T *always* produces a closed set or always produces an open set. Similarly, if the underlying space is a vector space, one might construct a new one-step consequence operator T by composing the one-step consequence operator for spatial logic programming with the operator that produces the smallest subspace containing a set, the *span* operator, or with the operator that produces the smallest convex closed set containing a set, the *convex closure* operator. In this way, one can ensure that the new one-step consequence operator T always produces a subspace or always produces a convex closed set. More generally, We say that an operator $O : 2^X \rightarrow 2^X$ is *monotonic* if for all $Y \subseteq Z \subseteq X$, we have $O(Y) \subseteq O(Z)$ and we say that O is *idempotent* for all $Y \subseteq X$, $O(O(Y)) = O(Y)$. Specifically, many familiar operators such as *closure*, *interior*, or the *span* and *convex-closure* operators in vector spaces over the rationals and other fields are monotonic idempotent operators. We call a monotonic idempotent operator a *miop*. We say that a set Y is *closed* with respect to miop O if and only if $Y = O(Y)$. Besides of examples listed above, in the context or regular languages, we discuss a number of monotone idempotent operators in Example 3, Section 4. By composing the one-step consequence operator for spatial logic programs with the operator O, we can ensure that the resulting one-step consequence operator always produces a fixed point of O. We can then think of the operator O as a parameter. This naturally leads us to a situation where we have a natural polymorphism for set based logic programming. That is, one can use the same logic program to produce stable models with different properties depending on how the operator O is chosen.

Moreover, in such a setting, one also has a variety of options for how to interpret negation. In normal logic programming, a model M satisfies $\neg p$ if $p \notin M$. From the spatial logic programming point of view, when p is interpreted as a singleton $\{p\}$, this would be equivalent to saying that M satisfies $\neg p$ if (i) $\{p\} \cap M = \emptyset$, or (equivalently) (ii) $\{p\} \not\subseteq M$. When the sense of p is a set with more than one element, it is easy to see that saying that M satisfies $\neg p$ if $[\![p]\!] \cap M = \emptyset$ (strong negation) is different from saying that M satisfies $\neg p$ if $[\![p]\!] \not\subseteq M$ (weak negation). This leads to two natural interpretations of the negation symbol which are compatible with the basic logic programming paradigm. When the underlying space has a miop cl, one can get even more subsidiary types of negation by taking M to satisfy $\neg p$ if $cl([\![p]\!]) \cap M \subseteq cl(\emptyset)$ (strong negation) or by taking M to satisfy $\neg p$ if $cl([\![p]\!]) \not\subseteq M$ (weak negation).

Blair, Marek, and Remmel [BMR08] showed that set based logic programing provides the foundations and basic techniques for crafting applications in the *answer set paradigm* as described in [MT99,Nie99] and then [GL02,Ba03]. That is, if in a given application, topological, linear algebraic, or similar constructs are either necessary or at least natural, then one can construct an answer set programming paradigm whose models correspond to natural closed structures. The expressive power of miops allows us to capture functions and relations intrinsic to the domain of a spatial logic program, but independent of the program. This permits set based logic programs to seamlessly serve as front-ends to other systems. Miops play the role of back-end, or "behind-the-scenes", procedures and functions.

We let $\mathcal{S}_P$ denote the set of least fixpoints with respect to the miops associated with P containing all finite unions and intersections of sets represented by the atoms of a finite set based logic program P. Here the elements of $\mathcal{S}_P$ may be finite or infinite. The main goal of this paper is find conditions on $\mathcal{S}_P$ which ensure that we can effectively decide whether a given element of $\mathcal{S}_P$ is a stable model of P. We shall show that if there is a way of associating codes $c(A)$ to the elements of $A \in \mathcal{S}_P$ such that there are effective procedures which, given codes $c(A)$ and $c(B)$ for elements of $A, B \in \mathcal{S}_P$, will (i) decide if $A \subseteq B$, (ii) decide if $A \cap B = \emptyset$, and (iii) produce of the codes of closures of $A \cup B$ and $A \cap B$ under miop operators associated with P, then we can effectively decide whether a code $c(A)$ is the code of a stable model of P. Our running example will be the case where P is a finite set-based logic program over a universe $X = \Sigma^*$ where Σ is a finite alphabet and the sets represented by atoms in P are languages contained in X which are accepted by finite automaton and the miops O involved in P preserve regular languages, i.e, if A is an automata such that the language $L(A)$ accepted by A is contained in X, then we can effectively construct an automaton B such that the language $L(B)$ accepted by B equals $O(L(A))$. Then, we shall show that the stable models of P are languages accepted by finite automaton and one can effectively check whether a language accepted by finite automaton is a stable model. Thus in this setting, one can effectively reason about an important class of infinite sets. However, it will be clear from our proofs that the only properties that we use for regular languages coded by their accepting DFAs are that the procedures for (i), (ii), and (iii) are effective.

The outline of this paper is as follows. In Section 2, we shall give the basic definitions of set based logic programming with miops. as developed by Blair, Marek, and Remmel [BMR08]. In Section 3, we shall review that basic properties of languages accepted by finite automata that we shall need. In Section 4, we shall show how the formalisms of finite automata can be seamlessly incorporated into the set based logic programming formalism. Finally, in Section 5, we give conclusions and directions for further research.

2 Set Logic Programs: syntax, miops, and semantics

We review the basic definitions of set based logic programming as introduced by Blair, Marek, and Remmel [BMR08]. The syntax of set based logic programs will essentially be the syntax of DATALOG programs with negation.

Following [BMR08], we define a **set based augmented first-order language** (**set based language**, for short) $\mathcal{L}$ as a triple $(L, X, [\![\cdot]\!])$, where
(1) L is a language for first-order predicate logic (without function symbols other than constants),
(2) X is a nonempty (possibly infinite) set, called the **interpretation space**, and
(3) $[\![\cdot]\!]$ is a mapping from the atoms of L to the power set of X, called the *sense assignment*. If p is an atom, then $[\![p]\!]$ is called the *sense* of p.

In our setting, a **set based logic program** has three components.
1) The language $\mathcal{L}$ which includes the interpretation space and the sense assignment.
2) The IDB (**Intentional Database**): A finite set of program clauses, each of the form $A \leftarrow L_1, \ldots, L_n$, where each L_i is a *literal*, i.e. an atom or the negation

of an atom, and A is an atom.
3) The EDB (**Extensional Database**): A finite set of clauses of the form $A \leftarrow$ where A is an atom.
Given a set based logic program P, the *Herbrand base* of P is the Herbrand base of the smallest set based language over which P is a set based logic program.

We shall assume that the classes of set based logic programs that we consider are always over a language for first-order logic L with no function symbols except constants, and a fixed set X. We let HB_L denote the Herbrand base of L, i.e. the set of atoms of L. We omit the subscript L when the context is clear. Thus we allow clauses whose instances are of the following form:

$$\mathcal{C} = A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m. \tag{1}$$

where A, B_i, and C_j are atoms for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. We let $head(\mathcal{C}) = A$, $Body(\mathcal{C}) = B_1, \ldots, B_n, \neg C_1, \ldots, \neg C_m$, and $PosBody(\mathcal{C}) = \{B_1, \ldots, B_m\}$, and $NegBody(\mathcal{C}) = \{C_1, \ldots, C_m\}$.

We let 2^X be the powerset of X. Given $[\![\cdot]\!] : \mathrm{HB}_L \longrightarrow 2^X$, an *interpretation* I of the set based language $\mathcal{L} = (L, X, [\![\cdot]\!])$ is a subset of X.

2.1 Examples of Monotonic Idempotent Operators

A second component of a set based logic program is one or more monotonic idempotent operators $O : 2^X \to 2^X$ that are associated with the program. Recall that an operator $O : 2^X \to 2^X$ is *monotonic* if for all $Y \subseteq Z \subseteq X$, we have $O(Y) \subseteq O(Z)$ and we say that O is *idempotent* for all $Y \subseteq X$, $O(O(Y)) = O(Y)$. We call a monotonic idempotent operator a *miop* (pronounced "my op"). We say that a set Y is *closed* with respect to miop O if and only if $Y = O(Y)$.

For example, suppose that the interpretation space X is either $\mathbf{R}^n$ or $\mathbf{Q}^n$ where $\mathbf{R}$ is the reals and $\mathbf{Q}$ is the rationals. Then, X is a topological vector space under the usual topology so that we have a number of natural miop operators:

1. $op_{id}(A) = A$, i.e. the identity map is simplest miop operator,
2. $op_c(A) = \bar{A}$ where $\bar{A}$ is the smallest closed set containing A,
3. $op_{int}(A) = int(A)$ where $int(A)$ is the interior of A,
4. $op_{convex}(A) = K(A)$ where $K(A)$ is the convex closure of A, i.e. the smallest set $K \subseteq X$ such that $A \subseteq K$ and whenever $x_1, \ldots, x_n \in K$ and $\alpha_1, \ldots, \alpha_n$ are elements of the underlying field ($\mathbf{R}$ or $\mathbf{Q}$) such that $\sum_{i=1}^n \alpha_i = 1$, then $\sum_{i=1}^n \alpha_i x_i$ is in K, and
5. $op_{subsp}(A) = (A)^*$ where $(A)^*$ is the subspace of X generated by A.

We should note that (5) is a prototypical example if we start with an *algebraic* structure. That is, in such cases, we can let $op_{substr}(A) = (A)^*$ where $(A)^*$ is the substructure of X generated by A. Examples of such miops include the following:

(a) if X is a group, we can let $op_{subgrp}(A) = (A)^*$ where $(A)^*$ is the subgroup of X generated by A,
(b) if X is a ring, we can let $op_{subrg}(A) = (A)^*$ where $(A)^*$ is the subring of X generated by A,

(c) if X is a field, we can let $op_{subfld}(A) = (A)^*$ where $(A)^*$ is the subfield of X generated by A,

(d) if X is a Boolean algebra, we can let $op_{subalg}(A) = (A)^*$ where $(A)^*$ is the subalgebra of X generated by A or we can let $op_{ideal}(A) = Id(A)$ where $Id(A)$ is the ideal of X generated by A, and

(e) if $(X, \leq_X)$ is a partially ordered set, we can let $op_{uideal}(A) = Uid(A)$ where $Uid(A)$ is the upper order ideal of X (that is, the least subset S of X containing A such that whenever $x \in S$ and $x \leq_X y$, then $y \in S$).

2.2 Set based logic programming with miops

Now suppose that we are given a miop $op^+ : 2^X \to 2^X$ and Horn set based logic program P over X. Here we say that a set based logic program is *Horn* if its IDB is Horn. Blair, Marek, and Remmel [BMR08] generalized the one-step consequence-operator of ordinary logic programs with respect to 2-valued logic to set based logic programs relative to a miop operator op^+ as follows. First, for any atom A and $I \subseteq X$, we say that $I \models_{[\![\cdot]\!],op^+} A$ if and only if $op^+([\![A]\!]) \subseteq I$. Then, given a set based logic program P with IDB P, let P' be the set of instances of a clauses in P and let

$$T_{P,op^+}(I) = op^+(I_1 \cup I_2)$$

where $I_1 = \bigcup\{[\![a]\!] \mid a \leftarrow L_1, \ldots, L_n \in P', I \models_{[\![\cdot]\!],op^+} L_i, i = 1, \ldots, n\}$ and $I_2 = \bigcup\{[\![a]\!] \mid a \leftarrow \text{ is an instance of a clause in the EDB of } P\}$.
We then say that a *supported model relative to* op^+ of P is a fixed point of T_{P,op^+}.

We iterate T_{P,op^+} according to the following.

$$\begin{aligned} T_{P,op^+} \uparrow^0 (I) &= I \\ T_{P,op^+} \uparrow^{\alpha+1} (I) &= T_{P,op^+}(T_{P,op^+} \uparrow^\alpha (I)) \\ T_{P,op^+} \uparrow^\lambda (I) &= op^+(\bigcup_{\alpha<\lambda} \{T_{P,op^+} \uparrow^\alpha (I)\}),\ \lambda \text{ limit} \end{aligned}$$

It is easy to see that if P is a Horn spatial logic program and op^+ is a miop, then T_{P,op^+} is monotonic. Blair, Marek, and Remmel [BMR08] proved the following.

Theorem 1. Given a miop op^+, the least model of a Horn set based logic program P exists and is closed under op^+ , is supported relative op^+, and is given by $\mathbf{T}_{P,op^+} \uparrow^\alpha (\emptyset)$ for the least ordinal α at which a fixed point is obtained.

We note, however, that if the Herbrand universe of a set based logic program is infinite (contains infinitely many constants) then, unlike the situation with ordinary Horn programs, T_{P,op^+} will not in general be upward continuous even in the case where $op^+(A) = A$ for all $A \subseteq X$. That is, consider the following example which was given in [BMR08].

Example 1. Assume that op^+ is the identity operator on 2^X. To specify a set based logic program, we must specify the language, EDB and IDB. Let $\mathcal{L} = (L, X, [\![\cdot]\!])$ where L has four unary predicate symbols: p, q, r and s, and countably many constants $e_0, e_1, \ldots,$. X is the set $\mathbf{N} \bigcup \{\mathbf{N}\}$ where $\mathbf{N}$ is the set of natural numbers, $\{0, 1, 2, \ldots\}$. $[\![\cdot]\!]$ is specified by $[\![q(e_n)]\!] = \{0, \ldots, n\}$, $[\![p(e_n)]\!] = \{0, \ldots, n+1\}$, $[\![r(e_n)]\!] = \mathbf{N}$, and $[\![s(e_n)]\!] = \{\mathbf{N}\}$.

The EDB is $q(e_0) \leftarrow$ and the IDB is: $p(X) \leftarrow q(X)$ and $s(e_0) \leftarrow r(e_0)$.

Now, after ω iterations upward from the empty interpretation, $r(e_0)$ becomes satisfied. One more iteration is required to reach an interpretation that satisfies $s(e_0)$, where the least fixed point is attained. $\Box$

Next we consider how we should deal with negation in the setting of miop operators. Suppose that we have a miop operator op^- on the space X. We do not require that op^- is the same as that miop op^+ but it may be. Our goal is to define two different satisfaction relations for negative literals relative to the miop operator op^- which are called strong and weak negation in [BMR08] [3].

For the rest of this paper, we shall think of a set based logic program P as a set of clauses of the form (1) where it may be that either n or m equals 0. We let $horn(P)$ denote the set of all Horn clauses in P and $nohorn(P) = P \setminus horn(P)$.

Definition 1. Suppose that P is a set based logic program over X and op^+ and op^- are miops on X and $a \in \{s, w\}$.

(I) Given any atom A and set $J \subseteq X$, then we say
$J \models^a_{[\![\cdot]\!], op^+, op^-} A$ if and only if $op^+([\![A]\!]) \subseteq J$.

(II)$_s$ (Strong negation) Given any atom A and set $J \subseteq X$, then we say
$J \models^s_{[\![\cdot]\!], op^+, op^-} \neg A$ if and only if $op^-([\![A]\!]) \cap J \subseteq op^-(\emptyset)$.

(II)$_w$ (Weak negation) Given any atom A and set $J \subseteq X$, then we say
$J \models^w_{[\![\cdot]\!], op^+, op^-} \neg A$ if and only if $op^-([\![A]\!]) \not\subseteq J$.

Definition 2. For any given set $J \subseteq X$ we define the *strong Gelfond-Lifschitz transform*, $GL^s_{J,[\![\cdot]\!],op^+,op^-}(P)$, of a program P with respect to miops op^+ and op^- on 2^X, in two steps. First, we consider all clauses in P,

$$\mathcal{C} = A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, C_m \tag{2}$$

where $A, B_1, \ldots, B_n, C_1, \ldots, C_m$ are atoms. If for some i, it is *not* the case that $J \models^s_{[\![\cdot]\!], op^+, op^-} \neg C_i$, then we eliminate clause $\mathcal{C}$. Otherwise we replace $\mathcal{C}$ by the Horn clause

$$A \leftarrow B_1, \ldots, B_n. \tag{3}$$

Then, $GL^s_{J,[\![\cdot]\!],op^+,\mathcal{R}}(P)$ consists of the set of all Horn clauses produced by this two step process.

[3] Lifschitz [Li94] observed that different modalities, thus different operators, can be used to evaluate positive and negative part of bodies of clauses of normal programs.

We define the *weak Gelfond-Lifschitz transform*, $GL^w_{J,[\![\cdot]\!],op^+,op^-}(P)$, of a program P with respect to miops op^+ and op^- on 2^X in a similar manner except that we use $\models^w_{[\![\cdot]\!],op^+,op^-}$ in place of $\models^s_{[\![\cdot]\!],op^+,op^-}$ in the definition.

Note that since $GL^a_{J,[\![\cdot]\!],op^+,op^-}(P)$ is a Horn set based logic program for either $a = s$ or $a = w$, the least model of $GL^a_{J,[\![\cdot]\!],op^+,op^-}(P)$ relative to op^+ is defined. We then define the a-stable model semantics for a set based logic program P over X relative to the miops op^+ and op^- on X for $a \in \{s, w\}$ as follows.

Definition 3. J is an *a-stable* model of P *relative* to op^+ and op^- if and only if J is the least fixed point of $T_{GL^a_{J,[\![\cdot]\!],op^+,op^-}(P),op^+}$.

Next we give a simple example to show that there is a difference between s-stable and w-stable models.

Example 2. Suppose that the space $X = \mathcal{R}^2$ is the real plane. Our program will have two atoms $\{a, b\}, \{c, d\}$ where a, b, c and d are reals. We let $[a, b]$ and $[c, d]$ denote the line segments connecting a to b and c to d respectively. We let the sense of the these atoms be the corresponding subsets, i.e. we let $[\![\{a, b\}]\!] = \{a, b\}$ and $[\![\{c, d\}]\!] = \{c, d\}$. We let $op^+ = op^- = op_{convex}$. The consider the following program $\mathcal{P}$.

(1) $\{a, b\} \leftarrow \neg\{c, d\}$
(2) $\{c, d\} \leftarrow \neg\{a, b\}$

There are four possible candidate for stable models in this case, namely (i) $\emptyset$, (ii) $[a, b]$, (iii) $[c, d]$, and (iv) $op_{convex}\{a, b, c, d\}$. Let us recall that $op_{convex}(X)$ is the convex closure of X which, depending on $a, b, c,$ and d may be either a quadrilateral, triangle, or a line segment.

If we are considering s-stable models where $J \models^s_{[\![\cdot]\!],op^+,op^-} \neg C$ if and only if $op^-(C) \cap J = op^-(\emptyset) = \emptyset$, then the only case where there are s-stable models if $[a, b]$ and $[c, d]$ are disjoint in which (ii) case and (iii) are s-stable models.

If we are considering w-stable models where $J \models^w_{[\![\cdot]\!],op^+,op^-} \neg C$ if and only if $op^-(C) \not\subseteq J$, then there are no w-stable models if $[a, b] = [c, d]$, (ii) is a w-stable model if $[a, b] \not\subseteq [c, d]$, (iii) is w-stable model if $[c, d] \not\subseteq [a, b]$ and (ii) and (iii) are w-stable models if neither $[a, b] \subseteq [c, d]$ nor $[c, d] \subseteq [a, b]$. $\square$

It is still the case that the a-stable models of a set based logic program P form an antichain for $a \in \{s, w\}$. That is, we have the following result.

Theorem 2. *Suppose that P is a set based logic program over X, op^+ and op^- are miops on X, and $a \in \{s, w\}$. If M and N are a-stable models of P and $M \subseteq N$, then $M = N$.*

Proof. It is easy to see that in general if $M \subseteq N$, then

$$GL^a_{N,[\![\cdot]\!],op^+,op^-}(P) \subseteq GL^a_{M,[\![\cdot]\!],op^+,op^-}(P).$$

Hence the least fixed point of $T_{GL^a_{N,[\![\cdot]\!],op^+,op^-}(P),op^+}$ is a subset of the least fixed point of $T_{GL^x_{M,[\![\cdot]\!],op^+,op^-}(P),op^+}$. But if $M \subseteq N$ and M and N are a-stable models, then N equals the least fixed point of $T_{GL^a_{N,[\![\cdot]\!],op^+,op^-}(P),op^+}$ and M equals the least fixed point of $T_{GL^a_{M,[\![\cdot]\!],op^+,op^-}(P),op^+}$ so that $N \subseteq M$. $\square$

3 Languages accepted by finite automaton

In this section, we shall briefly list some of the basic properties of languages accepted by finite automaton that we shall need.

Recall that a deterministic finite automaton (DFA) M is specified by a quintuple $M = (Q, \Sigma, \delta, s, F)$ where

Q is a finite alphabet of state symbols,
Σ is finite alphabet of input symbols,
$\delta : Q \times \Sigma \to Q$ is a transition function,
s in Q is the start state, and
$F \subseteq Q$ is the set of final (accepting) states.

We let $L(M)$ denote the set of all words w accepted by M. A nondeterministic automaton (NFA) $M = (Q, \Sigma, \delta, s, F)$ is specified by similar 5-tuple except that in this case $\delta \subseteq Q \times \Sigma \times Q$. It is well known that for any fixed finite alphabet Σ, the set of languages $L \subseteq \Sigma^*$ accepted by DFAs and the set languages of $L \subseteq \Sigma^*$ accepted by NFA's are the same. Moreover, given any two DFAs M_1 and M_2, there are standard constructions of DFAs M_3, M_4, and M_5 such that

$L(M_3) = L(M_1) \cap L(M_2)$,
$L(M_4) = L(M_1) \cup L(M_2)$, and
$L(M_5) = \Sigma^* - L(M_1)$.

We shall denote these three DFAs by $M_3 = M_1 \cap M_2$, $M_4 = M_1 \cup M_2$, and $M_5 = \bar{M}_1$. We notice that in such setting the automaton accepting the language L is a *code* for L. It is a well-known fact that instead of DFA one can consider a different class of codes for regular languages, namely regular expressions.

A crucial property of DFAs is the pumping lemma of [BPS61].

Lemma 1. *Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA and $p = |Q|$. Then for all words $w \in L(M)$ such that $|w| \geq p$, we can write $w = xyz$ for some $x, y, z \in \Sigma^*$ such that*

1. *$|xy| \leq p$,*
2. *$|y| \geq 1$, and*
3. *$xy^iz \in L(M)$ for all $i \geq 0$.*

One immediate consequence of the pumping lemma is that we can effectively decide whether $L(M)$ is empty or finite. That is, we have the following lemmas.

Lemma 2. *Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. Then, $L(M)$ is empty if and only if for every $w \in \Sigma^*$ such that $|w| < |Q|$, w is not accepted by $L(M)$.*

Lemma 3. *Let $M = (Q, \Sigma, \delta, s, F)$ be a DFA. Then, $L(M)$ is finite if and only if for every $w \in \Sigma^*$ such that $|Q| \leq |w| < 2|Q|$, w is not accepted by $L(M)$.*

Thus the complexity of the decision procedure to decide whether $L(M)$ is empty or finite depends directly on $|Q|$ and $|\Sigma|$. The fact that we can effectively decide if $L(M) = \emptyset$ also means that we can decide for any given DFAs M_1 and M_2 whether

1. $L(M_1) \subseteq L(M_2)$ since $L(M_1) \subseteq L(M_2)$ if and only if $L(M_1 \cap \bar{M_2}) = \emptyset$,
2. $L(M_1) = L(M_2)$ since $L(M_1) = L(M_2)$ if and only if $L(M_1) \subseteq L(M_2)$ and $L(M_2) \subseteq L(M_1)$, and
3. $L(M_1) \cap L(M_2) = \emptyset$ since $L(M_1) \cap L(M_2) = \emptyset$ if and only if $L(M_1 \cap M_2) = \emptyset$.

4 Set Based Logic Programming with Automata

In this section, we shall consider finite set based logic programs P over $\mathcal{L} = (L, X, [\![\cdot]\!])$ where $X = \Sigma^*$ for some finite alphabet Σ. Thus P consists of clauses of the form

$$\mathcal{C} = A \leftarrow B_1, \ldots, B_n, \neg C_1, \ldots, C_m \tag{4}$$

where $A, B_1, \ldots, B_n, C_1, \ldots, C_m$ are atoms. We shall assume that $X = \Sigma^*$ for some finite alphabet Σ and that for any clause of the form (4) in P,

$$[\![A]\!], [\![B_1]\!], \ldots, [\![B_n]\!], [\![C_1]\!], \ldots, [\![C_m]\!]$$

are all accepted by DFAs whose alphabet of symbols is Σ. **For the moment, assume also that op^+ and op^- are the identity operators.** For ease of notation, we shall assume that for any atom A that appears in P, A is a DFA whose over the alphabet Σ and that $[\![A]\!] = L(A)$.

Proposition 1. *For every finite set based program P where $op^+ = op_{id}$, every weak or strong stable model of P is a finite union of the sense assignments of the heads of clauses in P.*

Thus any weak or strong stable model of P must be a finite union of languages in Σ^* which are accepted by DFAs and, hence, the stable model itself is accepted by a DFA since languages accepted by DFAs are closed under union. We claim that if M is a DFA whose alphabet of symbols is Σ, then we can effectively decide whether $L(M)$ is a weak or strong stable model of P.

The first thing to observe is that we can effectively find the weak or strong Gelfond-Lifschitz transform of P. That is, under our assumptions for any atom A and any $a \in \{s, w\}$,

1. $L(M) \models^a_{[\![\cdot]\!], op^+, op^-} A$ if and only if $L(A) \subseteq L(M)$,
2. $L(M) \models^s_{[\![\cdot]\!], op^+, op^-} \neg A$ if and only if $L(A) \cap L(M) = \emptyset$, and
3. $L(M) \models^w_{[\![\cdot]\!], op^+, op^-} \neg A$ if and only if $L(A) \nsubseteq L(M)$.

It follows from the results in Section 3, that we can effectively decide whether $L(M) \models^a_{[\![\cdot]\!],op^+,op^-} A$, $L(M) \models^s_{[\![\cdot]\!],op^+,op^-} \neg A$, and $L(M) \models^w_{[\![\cdot]\!],op^+,op^-} \neg A$. Hence, we can effectively construct $GL^s_{L(M),[\![\cdot]\!],op^+,op^-}(P)$ and $GL^w_{L(M),[\![\cdot]\!],op^+,op^-}(P)$.

Now suppose that Q is a finite Horn set based logic program over $\mathcal{L} = (L, X, [\![\cdot]\!])$ where $X = \Sigma^*$ for some finite alphabet Σ and op^+ and op^- are the identity operators. Moreover, assume that for any atom A which appears in Q, $[\![A]\!]$ is a language accepted by a DFA whose alphabet is Σ. Again, for ease of notation, we shall assume that for any atom A that appears in P, A is a DFA whose alphabet is Σ and that $[\![A]\!] = L(A)$. Then, we claim that we can effectively construct a DFA M such that $L(M)$ is the least model of Q. First, we shall show that for all $n \geq 1$, we can effectively construct a DFA M^n such $T^n_{Q,op^+}(\emptyset) = L(M^n)$. Note that $T_{Q,op^+}(\emptyset)$ is equal to $\bigcup\{L(A) : A \leftarrow \ \in Q\}$. Now if $\{A \leftarrow \ \in Q\}$ is empty, then $T_{Q,op^+}(\emptyset) = \emptyset$ and the least model of Q equals $\emptyset$ so that we simply let M^1 be the one state DFA which has no accepting state. Otherwise, suppose

$$\{A : A \leftarrow \ \in Q\} = \{A^0_1, \ldots, A^0_{n_0}\}.$$

Then, we set $M^1 = A^0_1 \cup \cdots \cup A^0_{n_0}$. Now assume that we have constructed a DFA M^n such that $T^n_{Q,op^+}(\emptyset) = L(M^n)$. Then,

$$T_{Q,op^+}(L(M^n)) = op^+(I_1 \cup I_2)$$

where $I_1 = \bigcup\{[\![A]\!] \mid A \leftarrow B_1, \ldots, B_m \ \in Q, L(M^n) \models_{[\![\cdot]\!],op^+} B_i, i = 1, \ldots, n\}$ and $I_2 = \bigcup\{[\![A]\!] \mid A \leftarrow$ is a clause in the EDB of $Q\}$.

Note that $I_1 \cup I_2$ is finite since Q is finite. Since we can effectively decide whether $L(N) \subseteq L(M^n)$ for any DFA N, we can effectively decide whether $L(M^n) \models_{[\![\cdot]\!],op^+} B_i$ for any atom B_i and hence we can effectively compute I_1 and I_2. Then we simply let $L(M^{n+1})$ be the DFA whose language is the union of all the $L(A)$ such that $A \in I_1 \cup I_2$.

Finally, we can effectively check whether $L(M^{n+1}) = L(M^n)$. Since the least model of Q equals $L(M^n)$ where n is the least integer such that $L(M^{n+1}) = L(M^n)$, we can effectively construct a DFA R such that $L(R)$ is the least model of Q.

It follows that we can effectively construct DFAs M_s and M_w such that $L(M_s)$ is the least model of $GL^s_{L(M),[\![\cdot]\!],op^+,op^-}(P)$ and $L(M_w)$ is the least model of $GL^w_{L(M),[\![\cdot]\!],op^+,op^-}(P)$. Since we can effectively check whether $L(M) = L(M_s)$ and whether $L(M) = L(M_w)$, it follows that we can effectively decide if $L(M)$ is a weak or strong stable model of P.

We can extend our analysis to finite set based logic programs P with miops assuming that the miops for P satisfy the following property.

Definition 4. We say that a miop $op : 2^{\Sigma^*} \to 2^{\Sigma^*}$ is *effectively automata preserving* if for any DFA M whose underlying alphabet of symbols is Σ, we can effectively construct a DFA N whose underlying alphabet of symbols is Σ such that $L(N) = op(L(M))$.

We will now give a number of examples of miops on regular languages.

Example 3. Suppose that $\Sigma = \{0, 1, \ldots, m\}$. Then, the following are effectively automata preserving operators.

1. If N is a DFA whose underlying set of symbols is Σ, then we can define $op : 2^{\Sigma^*} \to 2^{\Sigma^*}$ by setting $op(S) = S \cup L(N)$ for any $S \subseteq \Sigma^*$. Clearly if $S = L(M)$ for some DFA M whose underlying set of symbols is Σ, then $op(L(M)) = L(M \cup N)$ so op is effectively automaton preserving.
2. If N is a DFA whose underlying set of symbols is Σ, then we can define $op : 2^{\Sigma^*} \to 2^{\Sigma^*}$ by setting $op(S) = S \cap L(N)$ for any $S \subseteq \Sigma^*$. Clearly if $S = L(M)$ for some DFA M whose underlying set of symbols is Σ, then $op(L(M) = L(M \cap N)$ so op is effectively automata preserving.
3. If T is any subset of Σ, we can let $op(S) = S(T^*)$. Again op will be an effectively automata preserving miop since if M is DFA whose underlying set of symbols is Σ, then let N be NFA constructed from M by adding loops on all the accepting states labeled with letters from T. It is easy to see that N accepts $L(M)T^*$ and then one can use the standard construction to find a DFA N' such that $L(N') = L(N)$. Note that in the special case where T equals Σ, we can think of op as constructing the upper ideal of S in Σ^* relative to the partial order $\sqsubseteq$ where for any words $u, v \in \Sigma^*$, $u \sqsubseteq v$ if and only if u is prefix of v, i.e. v is of the form uw for some $w \in \Sigma^*$. For any poset $(P, \leq_P)$, we say that a set $U \subseteq P$ is an *upper ideal* in P, if whenever $x \leq_P y$ and $x \in P$, then $y \in P$. Clearly, for the poset $(\Sigma^*, \sqsubseteq)$, $op(S)$ is the upper ideal of $(\Sigma^*, \sqsubseteq)$ generated by S.
4. Let $\mathcal{P} = (\Sigma, \leq)$ be a partially-ordered set. For any $w, w' \in \Sigma^*$, we say that w' is a factor of w if there are words $u, v \in \Sigma^*$ with $w = uw'v$. Define the *generalized factor order* on P^* by letting $u \leq w$ if there is a factor w' of w having the same length as u such that $u \leq w'$, where the comparison of u and w' is done componentwise using the partial order in $\mathcal{P}$. Again we can show that if $op(S)$ is the upper ideal generated by S the generalized factor order relative to P^*, then op is an effectively automata preserving miop. That is, if we start with a DFA $M = (Q, \Sigma, \delta, s, F)$, then we can modify M to an NFA that accepts $op(L(M))$ as follows. Think of M as a digraph with edges labeled by elements of Σ in the usual manner. First, we add a new start state s_0. There are loops from s_0 labeled with all letters in Σ. There is also a λ-transition from s_0 to the old start state s. We then modify the transitions in M so that if there is an edge from state q to q' labeled with symbol r, then we add an edge from q to q' with any symbol s such that $r \leq s$. Finally we add loops to all accepting states such that labeled with all letters in in Σ.
5. If we allow multiple representations of the infinite dimensional vector space V_∞ for the field GF_q where q is prime, then the operator op_{subsp} can be

thought of an automaton preserving miop. Let $\Sigma = \{0, \ldots, q-1\}$. The standard way to represent the elements of V_∞ is to let $\mathbf{0} = 0$ and think of a non-zero element of V_∞ as a finite sequence $\sigma_1 \ldots \sigma_n$ where $\sigma_n \neq 0$. The operations of scalar multiplication and addition are then performed componentwise. In our case, we will let any element $\sigma \in V_\infty$ have multiple representations, namely, σ can be represented by $\sigma 0^n$ for any $n \geq 0$. Then, we let $op_{subsp}(S)$ be the set of all representatives of the subspace of V_∞ generated by S. In what follows, we shall only describe how to construct NFA's that accept the desired languages since the Myhill-Nerode Theorem allows us to construct in a uniform manner, for any NFA M, a DFA D such that $L(M) = L(D)$. First, consider miop op_1 such that $op_1(S)$ is the set of all representations of elements of S. If M is a DFA whose underlying alphabet is Σ, then we can modify M to an NFA N that accepts $op_1(S)$ as follows. First, any state q such that there is an n such that the word 0^n starting at state q ends in an accepting state is an accepting state of N. In particular, every accepting state of M is an accepting state of N. In addition, we add loops labeled with 0 to all the accepting states of N.

Next we let $op_2(S)$ denote the set of all representations of any element which is a scalar multiple of an element of S. We claim op_2 is also an automaton preserving miop. That is, if M is a DFA whose underlying alphabet is Σ, then we can modify M to an NFA $\bar{N}$ that accepts $op_2(S)$ as follows. First, let N be the NFA such that $op_1(L(M)) = L(N)$. The for each $a \in \{0, \ldots, q-1\}$, let aN be the NFA that is constructed from N by replacing each edge labeled with the letter x by an edge labeled ax. Then, it is clear that $L(aN) = \{(a\sigma_1) \ldots (a\sigma_n) : \sigma_1 \ldots \sigma_n \in L(N)\}$ so that $op_2(L(M)) = L(\bar{N})$ where $\bar{N} = 0N \cup 1N \cup \cdots \cup (q-1)N$.

Finally for any $a, b \in \{0, \ldots, q-1\}$, we let $op_{a,b}(S)$ denote the set of all representatives of the form $a\sigma + b\tau$ such that σ, τ are in S and $|\sigma| = |\tau|$. $op_{a,b}$ is not a miop, but nevertheless for any DFA M, we can construct an NFA $R_{a,b}$ such that $L(R_{a,b}) = op_{a,b}(L(M))$. First, let $N = (Q, \Sigma, \delta, s, F)$ be the DFA such that $L(N) = op_2(L(M))$. Then, the set of states of $R_{a,b}$ will be $Q \times Q$, (s,s) will be the start state of $R_{a,b}$, and $F \times F$ will be the set of final states of $R_{a,b}$. Now suppose that there are edges from p_0 to p_1 labeled with α and from q_0 to q_1 labeled with β in N. Then, we will have an edge in $R_{a,b}$ from (p_0, q_0) to (p_1, q_1) labeled with $a\alpha + b\beta$. It is easy to see that $L(R_{a,b}) = op_{a,b}(L(M))$. and hence if we let R be the DFA such that $R = \bigcup_{(a,b) \in \Sigma \times \Sigma} R_{a,b}$, then $S \subseteq L(R) \subseteq op_{subsp}(S)$ and $L(R)$ has the property that if $s_1, s_2 \in S$, then $as_1 + bs_2 \in L(R)$ for any $a, b \in GF_q$. By a similar argument, we can construct for any finite sequence of distinct elements $a_1, \ldots, a_r$ from GF_q, a DFA $U_{a_1,\ldots,a_r}$ such that $L(U_{a_1,\ldots,a_r})$ equals the set of all $a_1 t_1 + \cdots + a_r t_r$ such that $t_1, \ldots t_r \in L(R)$. It then follows that $op_{subsp}(S)$ equal the union of $L(U_{a_1,\ldots,a_r})$ over all possible finite sequence of distinct elements from GF_q and hence is we can construct a DFA U which accepts $op_{subsp}(S)$. $\square$

It is then easy to check that if $op^+ : 2^{\Sigma^*} \to 2^{\Sigma^*}$, then for any Horn set based logic program Q with the properties described above, we can construct a DFA M^n such that $T^n_{Q,op^+}(\emptyset) = L(M^n)$ and, hence, we can effectively construct the least model of Q. Thus we have the following result.

Theorem 3. *Suppose that P is a finite set based logic program over $\mathcal{L} = (L, X, [\![\cdot]\!])$ where $X = \Sigma^*$ for some finite alphabet Σ and $op^+ : 2^{\Sigma^*} \to 2^{\Sigma^*}$ and $op^- : 2^{\Sigma^*} \to 2^{\Sigma^*}$ are effectively automaton preserving miops. Moreover, assume that for any atom A which appears in Q, $[\![A]\!]$ is a language accepted by a DFA whose underlying set of symbols is Σ. Then:*

1. *Every weak (strong) stable model of P is a language accepted by a DFA.*
2. *For any DFA M whose underlying set of symbols is Σ, we can effectively decide whether $L(M)$ is a weak or strong stable model of P.*

Note that under the assumptions of Theorem 3, there are only finitely many possible strong or weak stable models the program P, namely a union of the sense of the head of certain clauses, and these are all recognizable by DFAs. Hence it is decidable whether such a set based logic program has a weak or strong stable model and there is an algorithm to find all such weak or strong stable models.

5 Conclusions

We showed in Theorem 3 that if the senses of the atoms of a finite set based logic program P are all regular languages over some fixed finite alphabet and the miops involved are all automaton preserving miops, then we can effectively decide if P has weak or strong stable model and there is an algorithm to find all weak and strong stable models. In fact, it is not difficult to see that all the operations in searching for either a weak or strong stable model of such programs are effective so that it is possible to extend existing search engines to produce either weak or strong stable models of such programs. However, we suspect that the problem of how to optimize such extensions of existing search engines will be an interesting and challenging research problem. Finite automaton are useful for carrying out a lot of recognition tasks such as search for keywords or ensuring documents or strings have a proper form so that our results show that we can add ASP programming on top of such recognition tasks.

If we examine the proof of Theorem 3, it is clear that we used DFAs as codes for set of regular languages $\mathcal{S}_P$ that arise by taking the closures under op^+ and op^- of finite unions of the languages associated with atoms of P. Here we consider the empty set as the empty union so that the emptyset is in $\mathcal{S}_P$. Then the only properties of such regular languages that were necessary to prove Theorem 3 was that we have effective procedures which, given codes for $A, B \in mathcalS_P$, (i) decide if $A \subseteq B$, (ii) decide $A \cap B = \emptyset$, and (iii) produce the codes of $op^+(A \cup B)$, $op^+(A \cap B)$, $op^-(A \cup B)$ and $op^+(A \cup B)$.

For any finite set based logic program P, we let $\mathcal{S}_P$ denote set of fix points of all finite unions of sets represented by the atoms of a finite set based logic program P of the miops associated with P. If we can associate a code $c(A)$ to each elements of $A \in \mathcal{S}_P$ such that there are effective procedures which, given codes $c(A)$ and $c(B)$ for elements of $A, B \in \mathcal{S}_P$, will (i) decide if $A \subseteq B$, (ii) decide if $A \cap B = \emptyset$, and (iii) produce of the codes of closures of $A \cup B$ and $A \cap B$ under miop operators associated with P, then we can prove the analogue of Theorem 3 for P. We have shown that the case where the code of an atom A is a DFA which accepts A (alernatively a regular expression describing A) then we have such procedures. However, such codes and procedures are available in many other cases. For example, if all sets involved are the convex closures of a finite set of points in $\mathbb{R}^n$ and $op^+ = op_{convex}$ and $op^- = op_{id}$ or if all sets involved are finite dimensional vector spaces over a computable fiel and $op^+ = op^- = op_{subsp}$, then such codes and procedures as described above can be constructed.

Acknowledgments: The first author was partially supported by NASA-JLP contract 1401954. The second author has been partially supported by NSF grant DMS 0654060.

References

[AB90] Apt, K. and Blair, H.A. Arithmetic Classification of Perfect Models of Stratified Programs. *Fundamenta Informaticae* 13:1–17, 1990.

[BL02] Babovich, Y. and Lifschitz, V. *Cmodels*, http://www.cs.utexas.edu/users/tag/cmodels.html, 2002

[Ba03] Baral, C. *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press, 2003.

[BPS61] Bar-Hillel, Y., Perles, M.A. and Shamir, E. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunikationsforschung* 14:143–172, 1961.

[BMR01] Blair. H.A., Marek, V.W. and Remmel, J.B. Spatial Logic Programming, In: *Proceedings SCI 2001*, Orlando, FL, July, 2001.

[BMR08] Blair. H.A., Marek, V.W. and Remmel, J.B. Set Based Logic Programming, *Annals of Mathematics and Artificial Intelligence* 52:81–105, 2008.

[BMS95] Blair. H.A., Marek, V.W. and Schlipf, J.S. The Expressivness of Locally Stratified Programs. *Annals of Mathematics and Artificial Intelligence* 15:209–229, 1995.

[BG02] Blumensath, A. and Grädel, E. Automatic Structures. In: *Proceedings of the* 15^{th} *Symposium on Logic in Computer Science* LICS'00, pages 51-62, 2000.

[Den00] Denecker, M. Extending Classical Logic with Inductive Definitions. In *First International Conference on Computational Logic (CL2000). Lecture Notes in Artificial Intelligence* 1861. Springer-Verlag, pages 703–717, 2000.

[DG82] Dowling, W.F. and Gallier, J.H. Linear-time algorithms for testing satisfiability of propositional Horn formulae. *Journal of Logic Programming* 3:267–284, 1984.

[GKN+] Gebser, M., Kaufmann, B., Neumann, A., and Schaub, T. *clasp* – a Conflict-driven Answer Set Solver. In: *Proceedings of LPNMR 07, Lecture Notes in Artificial Intelligence* 4483, pages 260–265, Springer-Verlag 2007.

[KN03] Heljanko, K. and Niemelä, I. Bounded LTL model checking with stable models. *Theory and Practice of Logic Programming* 3:519–550, 2003.

[GL02] Gelfond, M. and Leone. Logic Programming and Knowledge Representation – A-Prolog perspective. *Artificial Intelligence Journal* 138:3–38, 2002.

[GL88] Gelfond, M. and Lifschitz, V. The stable model semantics for logic programming. In *Proceedings of the International Joint Conference and Symposium on Logic Programming.* MIT Press, 1070–1080, 1988.

[GLM06] Giunchiglia, E., Lierer, Y. and Maratea, M. Answer Set Programming Based on Propositional Satisfiability. *Journal of Automated Reasoning* 36:345–377, 2006.

[KN95] Khoussainov, B. and Nerode, A. Automatic Presentations of Structures, *Springer Lecture Notes in Computer Science* 960, pages 367-392, 1995.

[KNRS07] Khoussainov, B., Nies, A., Rubin S., and Stephan F., Automatic structures: richness and limitations, *Logical Methods of Computer Science* 3(2), 18 pp. (electronic), 2007.

[LPF+06] Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*. 7:499–562, 2006.

[Li94] Lifschitz, V. Minimal belief and negation as failure. *Artificial Intelligence Journal* 70:53–72, 1994.

[LZ02] Lin, F. and Zhao, Y. ASSAT: Computing answer sets of a logic program by SAT solvers. *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02).* AAAI Press, 112–117, 2002.

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

[MR09] Marek, V.W. and Remmel, J.B. Automata and Answer Set Programming. In: *Logical Foundations of Computer Science*, *Springer Lecture Notes in Computer Science* 5407, Springer-Verlag, pages 323-337, 2009.

[MT93] Marek, W. and Truszczyński, M. *Nonmonotonic Logic – Context-Dependent Reasoning*, Springer-Verlag, 1993.

[MT99] Marek, V.W. and Truszczyński, M. Stable Models and an Alternative Logic Programming Paradigm. In: *The Logic Programming Paradigm*, Series Artificial Intelligence, Springer-Verlag, pages 375-398, 1999.

[Nie99] Niemelä, I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25:241–273, 1999.

[SNS02] Simons, P., Niemelä, I. and Soininen, T, Extending and implementing stable semantics of logic programs, *Artificial Intelligence Journal* 138:181–234, 2002.

[SNTS01] Soininen, T., Niemelä, I, Tiihonen, J. and Sulonen, R. Representing Configuration Knowledge with Weight Constraint Rules. In: *Answer Set Programming 2001.*

[Sm68] Smullyan, R. *First-order Logic*, Springer-Verlag, 1968.