Computing stable models: worst-case performance estimates

ZBIGNIEW LONC
Faculty of Mathematics and Information Science
Warsaw University of Technology
00-661 Warsaw, Poland
(*e-mail:* zblonc@alpha.mini.pw.edu.pl)

MIROSŁAW TRUSZCZYŃSKI
Department of Computer Science
University of Kentucky
Lexington, KY 40506-0046, USA
(*e-mail:* mirek@cs.uky.edu)

Abstract

We study algorithms for computing stable models of logic programs and derive estimates on their worst-case performance that are asymptotically better than the trivial bound of $O(m2^n)$, where m is the size of an input program and n is the number of its atoms. For instance, for programs whose clauses consist of at most two literals (counting the head) we design an algorithm to compute stable models that works in time $O(m \times 1.44225^n)$. We present similar results for several broader classes of programs. Finally, we study the applicability of the techniques developed in the paper to the analysis of the performance of *smodels*.

KEYWORDS: logic programs, stable models, computing stable models, worst-case bounds

1 Introduction

The stable-model semantics was introduced by Gelfond and Lifschitz (Gelfond and Lifschitz 1988) to provide an interpretation for the negation operator in logic programming. In this paper, we study algorithms to compute stable models of propositional logic programs. Our main goal is to design algorithms for which one can derive non-trivial worst-case performance bounds.

Computing stable models is important. It allows us to use logic programming, with the negation operator interpreted by the stable model semantics, as a *computational* knowledge representation tool and as a declarative programming system (Marek and Truszczyński 1999; Niemelä 1999). In most cases, when designing algorithms for computing stable models we restrict the syntax to that of DATALOG *with negation* (or DATALOG$^\neg$, for short), by eliminating function symbols from the language. When function symbols are allowed, models can be infinite and highly complex, and the general problem of existence of a stable model of a finite logic

program is not even semi-decidable (Marek et al. 1994)[1]. On the other hand, when function symbols are not used, stable models are guaranteed to be finite and can be computed. Some recent results on the algorithmic aspects of the problem to compute stable models of logic programs can be found in (Lonc and Truszczyński 2001; Truszczyński 2001; Lonc and Truszczyński 2001).

To compute stable models of *finite* DATALOG$^\neg$ programs we usually proceed in two steps. In the first step, we *ground* an input program P and produce a *finite* propositional program with the same stable models as P (the finiteness of the resulting *ground* program is ensured by the finiteness of P and the absence of function symbols). In the second step, we compute stable models of the ground program by applying search. This general approach is used in *smodels* (Niemelä and Simons 2000) and *dlv* (Eiter et al. 2000), two most advanced systems to process DATALOG$^\neg$ programs[2].

It is this second step of the process — computing stable models of propositional logic programs (in particular, programs obtained by grounding DATALOG$^\neg$ programs) — that is of central interest to us in the present paper. Stable models of a propositional logic program P can be computed by a trivial brute-force algorithm that generates all subsets of the set of atoms of P and, for each of these subsets, checks the stability condition. This algorithm can be implemented to run in time $O(m2^n)$, where m is the size of P and n is the number of atoms in P (we will use m and n in this meaning consistently throughout the paper). The algorithms used in *smodels* and *dlv* refine this brute-force algorithm by employing effective search-space pruning techniques. Experiments show that their performance is significantly better than that of the brute-force algorithm. However, at present, no non-trivial upper bound on their worst-case running time is known. In fact, *no* algorithms for computing stable models are known whose worst-case performance would be *provably* better than that of the brute-force algorithm. Our main goal here is to design such algorithms.

To this end, we propose a template for an algorithm to compute stable models of propositional programs. This template involves an auxiliary procedure whose particular instantiation determines the specific algorithm and its running time. We propose concrete implementations of this procedure and show that the resulting algorithms for computing stable models are asymptotically better than the straightforward algorithm described above. The performance analysis of our algorithms is closely related to the question of how many stable models logic programs may have. The template proposed in the paper can be instantiated to algorithms such as *smodels* but also to algorithms that search for stable models over search trees that are not, in general, binary.

Our main results concern propositional logic programs, called *t-programs*, in which the number of literals in *normal* clauses (we give the definition later), in-

[1] We note, however, that some progress on automated reasoning with logic programs that allow function symbols have recently been obtained by Bonatti (Bonatti 2001; Bonatti 2002).

[2] In fact, *dlv* is designed to process programs from a broader class of *disjunctive* DATALOG$^\neg$ programs.

cluding the head, is bounded by a constant t. Despite their restricted syntax t-programs are of interest. Many logic programs that were proposed as encodings of problems in planning, model checking and combinatorics become propositional 2- or 3-programs after grounding. For instance, the ground versions of the programs given in (Marek and Truszczyński 1999) as encodings of the propositional satisfiability and the Hamilton cycle problems are 3-programs. Similarly, the ground versions of the programs given in (Niemelä 1999) as encodings of the pigeonhole, the n-queens, the graph k-colorability and the Schur number problems are 2-programs. In each case some straightforward program simplifications must be applied (all these simplifications are implemented in *lparse*, a grounding program developed for *smodels* (Syrjänen 1999)). In general, programs obtained by grounding finite DATALOG$^{\neg}$ programs are t-programs, for some fixed and usually small t that depends only on the problem specification and not on a particular problem instance.

In the paper, for every $t \geq 2$, we construct an algorithm that computes all stable models of a t-program P in time $O(m\alpha_t^n)$, where α_t is a constant such that $\alpha_t < 2-1/2^t$.

For 2-programs we obtain stronger results. We construct an algorithm that computes all stable models of a 2-program in time $O(m3^{n/3}) = O(m \times 1.44225^n)$. We note that $1.44225 < \alpha_2 \approx 1.61803$. Thus, this algorithm is indeed a significant improvement over the algorithm following from general considerations discussed above. We design an even faster algorithm for a subclass of 2-programs consisting of programs that are purely negative and do not contain *dual* clauses. This algorithm runs in time $O(m \times 1.23651^n)$.

We obtain significant improvements over the general results concerning t-programs also in the case when $t = 3$. Namely, we describe an algorithm that computes all stable models of a 3-program P in time $O(m \times 1.70711^n)$. In contrast, since $\alpha_3 \approx 1.83931$, the algorithm implied by the general considerations runs in time $O(m \times 1.83931^n)$.

For programs in all the classes mentioned above, we obtain upper bounds on the maximum number of stable models a program in a given class may have. As mentioned earlier, these bounds are related to the worst-case performance estimates for algorithms we proposed.

In the paper we also consider a general case where no bounds on the length of a clause are imposed. We describe an algorithm to compute all stable models of such programs. Its worst-case complexity is better by the factor of $\sqrt{n}$ than that of the brute-force algorithm.

Finally, in the paper we discuss some lower bounds for the worst-case estimate of the time needed to compute *all* stable models. Since computing stable models requires that all of them be output, the sum of the cardinalities of all stable models of a program provides a lower bound on the worst-case performance estimate. Thus, to obtain a good bound, one needs to construct programs with as many as possible "large" stable models. We present two such constructions in the paper. In some cases, they demonstrate optimality of our algorithms in the sense that the exponential factor in the formula cannot be improved.

It is well known that, by introducing new atoms, every logic program P can be

transformed in polynomial time into a 3-program P' that is, essentially, equivalent to P. Specifically, every stable model of P with the set of atoms At is of the form $M' \cap At$, for some stable model M' of P' and, for every stable model M' of P', the set $M' \cap At$ is a stable model of P. This observation might suggest that in order to design fast algorithms to compute stable models, it is enough to focus on the class of 3-programs. It is not the case. In the worst case, the number of new atoms that need to be introduced is of the order of the size of the original program P. Consequently, an algorithm to compute stable models that can be obtained by combining the reduction described above with an algorithm to compute stable models of 3-programs runs in time $O(m \times 2^m)$ and is asymptotically slower than the brute-force approach outlined earlier. Thus, it is necessary to study algorithms for computing stable models designed explicitly for particular classes of programs.

2 Preliminaries

For a detailed account of logic programming and stable model semantics we refer the reader to (Gelfond and Lifschitz 1988; Apt 1990; Marek and Truszczyński 1993). In the paper, we consider only the propositional case. The language is determined by some (infinite) set At of atoms. A *literal* is an atom or any expression of the form $\mathbf{not}(a)$, where a is an atom. Literals b and $\mathbf{not}(b)$, where b is an atom, are *dual* to each other. For a literal β, we denote its dual by $\mathbf{not}(\beta)$.

A *clause* is an expression c of the form $p \leftarrow B$ or $\leftarrow B$, where p is an atom and B is a list of literals. The atom p (if present) is called the *head* of c and is denoted by $h(c)$. The set of literals B is called the *body* of c. The set of all atoms appearing as non-negated literals in B is the *positive body* of c, $b^+(c)$, in symbols. The set of atoms appearing in negated literals of B is the *negative body* of c, $b^-(c)$, in symbols. We emphasize that while the body of a rule consists of *literals*, positive and negative bodies are defined to consist of *atoms*. We assume that no literals in B are repeated.

A clause of the form $p \leftarrow B$ is called a *normal* clause. A clause of the form $\leftarrow B$ is called a *constraint*. A clause whose positive body is empty is called a *purely negative* clause. A normal clause whose negative body is empty is called a *Horn* clause (thus, in this paper, we require that a Horn clause have a *non-empty* head).

A *logic program* is a collection of clauses. If every clause of P is normal, P is a *normal logic program*. If every clause in P is *purely negative*, P is a *purely negative* program. If every clause in a logic program P is a Horn clause, P is a *Horn* program. Finally, a logic program P is a *t-program* if every normal clause in P has no more than t literals (counting the head). We note that the definition of t-programs does not impose any restrictions on the length of constraints.

For a logic program P, by $At(P)$ we denote the set of all atoms appearing in P. We define

$$Lit(P) = At(P) \cup \{\mathbf{not}(a) \colon a \in At(P)\}.$$

A set of atoms M is a *model* of a clause of the form $p \leftarrow B$ if: (i) $p \in M$, or (ii) $b \notin M$, for some atom $b \in B$, or (iii) $b \in M$, for some atom b such that $\mathbf{not}(b) \in B$.

Similarly, M is a *model* of a clause of the form $\leftarrow B$ if: (i) $b \notin M$, for some atom $b \in B$, or (ii) $b \in M$, for some atom b such that $\mathbf{not}(b) \in B$. A set of atoms M is a *model* of a logic program P if M is a model of every clause in P. It is well known that every Horn program P has a least model (with respect to set inclusion). We will denote this model by $lm(P)$.

Given a normal logic program P and a set of atoms $M \subseteq At(P)$, we define the *reduct* of P with respect to M (P^M, in symbols) to be the logic program obtained from P by

1. removing from P each clause c such that $M \cap b^-(c) \neq \emptyset$ (we call such clauses *blocked by* M),
2. removing all negated atoms from the bodies of all the rules that remain (that is, those rules that are not blocked by M).

Since we assumed that the program P is normal, the reduct P^M is a Horn program. Thus, it has a least model. We say that M is a *stable model* of P if $M = lm(P^M)$. Both the notion of the reduct and that of a stable model were introduced in (Gelfond and Lifschitz 1988).

The concept of a stable model can be extended to the case of arbitrary logic programs (not necessarily normal). Let P be such a program. We define a set of atoms M to be a *stable model* of P if (1) M is a stable model of the program consisting of all normal clauses in P, and if (2) M is a model of the program consisting of all constraints of P.

A clause c is a *tautology* if it is normal and $h(c) \in b^+(c)$, or if $b^+(c) \cap b^-(c) \neq \emptyset$. A clause c is a *virtual constraint* if it is normal and $h(c) \in b^-(c)$. We have the following result (Dix 1995).

Proposition 1
Let P be a logic program and let P' be the subprogram of P obtained by removing from P all tautologies, constraints and virtual constraints. A set of atoms M is a stable model of P if and only if M is a stable model of P' and satisfies all constraints and virtual constraints of P.

Thanks to this proposition, when designing algorithms for computing stable models we may restrict attention to normal programs without tautologies, constraints and virtual constraints. Indeed, to compute stable models of P we simply compute stable models for P' and, for each of them, check whether it satisfies all constraints and virtual constraints of P. Those that "pass" the check are stable models of P. Moreover, every stable model of P can be found in this way.

For a set of literals $L \subseteq Lit(P)$, we define:

$$L^+ = \{a \in At(P) \colon a \in L\} \quad \text{and} \quad L^- = \{a \in At(P) \colon \mathbf{not}(a) \in L\}.$$

We also define $L^0 = L^+ \cup L^-$. A set of literals L is *consistent* if $L^+ \cap L^- = \emptyset$. A set of atoms $M \subseteq At(P)$ is *consistent* with a set of literals $L \subseteq Lit(P)$, if $L^+ \subseteq M$ and $L^- \cap M = \emptyset$.

To characterize stable models of a program P that are consistent with a set of literals $L \subseteq Lit(P)$, we denote by $[P]_L$ the program obtained from P by removing:

1. every clause c such that $b^+(c) \cap L^- \neq \emptyset$
2. every clause c such that $b^-(c) \cap L^+ \neq \emptyset$
3. every clause c such that $h(c) \in L^0$
4. every occurrence of a literal in L from the bodies of the remaining clauses.

The program $[P]_L$ contains all information necessary to reconstruct stable models of P that are consistent with L. The following result was obtained in (Dix 1995) (we refer also to (Subrahmanian et al. 1995; Cholewiński and Truszczyński 1999)).

Proposition 2
Let P be a logic program and L be a set of literals of P. If M is a stable model of P consistent with L, then $M \setminus L^+$ is a stable model of $[P]_L$.

Thus, to compute all stable models of P that are consistent with L, one can first check if L is consistent. If not, there are no stable models consistent with L. Otherwise, one can compute all stable models of $[P]_L$, for each such model M' check whether $M = M' \cup L^+$ is a stable model of P and, if so, output M. This approach is the basis of the algorithm to compute stable models that we present in the following section.

3 A high-level view of stable model computation

We will now describe an algorithm *stable*(P, L) that, given a *normal* program P and a set of literals L, outputs all stable models of P that are consistent with L. The key concept we need is that of a complete collection. Let P be a logic program. A non-empty collection $\mathcal{A}$ of non-empty subsets of $Lit(P)$ is *complete* for P if every stable model of P is consistent with at least one set $A \in \mathcal{A}$. The collection $\mathcal{A} = \{\{a\}, \{\mathbf{not}(a)\}\}$, where a is an atom of P, is a simple example of a complete collection for P.

In the description given below, we assume that *complete*(P) is a procedure that, for a program P, computes a collection of sets of literals that is complete for P.

stable(P, L)
(0) **if** L is consistent **then**
(1) **if** $[P]_L = \emptyset$ **then**
(2) check whether L^+ is a stable model of P and, if so, output it
(3) **else**
(4) $\mathcal{A} :=$ *complete*$([P]_L)$;
(5) **for every** $A \in \mathcal{A}$ **do**
(6) *stable*$(P, L \cup A)$
(7) **end of** *stable*.

Proposition 3
Let P be a normal finite propositional logic program. For every $L \subseteq Lit(P)$, *stable*(P, L) returns all stable models of P consistent with L.

Proof
We proceed by induction on $|At([P]_L)|$. To start, let us consider a call to $stable(P, L)$ in the case when $|At([P]_L)| = 0$ and let M be a set returned by $stable(P, L)$. It follows that L is consistent and that M is a stable model of P. Moreover, since $M = L^+$, M is consistent with L. Conversely, let M be a stable model of P that is consistent with L. By Proposition 2, $M \setminus L^+$ is a stable model of $[P]_L$. Since L is consistent (as M is consistent with L) and $[P]_L = \emptyset$, $M \setminus L^+ = \emptyset$. Since M is consistent with L, $M = L^+$. Thus, M is returned by $stable(P, L)$.

For the inductive step, let us consider a call to $stable(P, L)$, where $|At([P]_L)| > 0$. Let M be a set returned by this call. Then M is returned by a call to $stable(P, L \cup A)$, for some $A \in \mathcal{A}$, where $\mathcal{A}$ is a complete family for $[P]_L$. Since elements of a complete family are non-empty and consist of literals actually occurring in $[P]_L$, $|At([P]_{L \cup A})| < |At([P]_L)|$. By the induction hypothesis it follows that M is a stable model of P consistent with $L \cup A$ and, consequently, with L.

Conversely, let us assume that M is a stable model of P consistent with L. Then, by Proposition 2, $M \setminus L^+$ is a stable model of $[P]_L$. Since $\mathcal{A}$ (computed in line (4)) is a complete collection for $[P]_L$, there is $A \in \mathcal{A}$ such that $M \setminus L^+$ is consistent with A. Since $A \cap L = \emptyset$ (as $A \subseteq At([P]_L)$), M is a stable model of P consistent with $L \cup A$. Since $|At([P]_{L \cup A})| < |At([P]_L)|$, by the induction hypothesis it follows that M is output during the recursive call to $stable(P, L \cup A)$. □

Let us note that typically algorithms for computing stable models of logic programs and models of CNF theories search over a binary tree. That is, at every branch point a variable, say x, is chosen and the search splits into two paths: one where x is assumed to be true and the other one where x is assumed to be false (not necessarily in this order). The search tree traversed by our algorithm is not necessarily binary. At each branch point, the search splits into as many different paths as there are elements in the complete family returned by the call to procedure *complete*. While algorithms searching over binary trees can be derived from our template by specifying the procedure *complete* so that it always returns collections consisting of at most two sets, the class of algorithms that can be derived from our template is broader. We provide additional comments on this issue in Section 7.

We will now study the performance of the algorithm *stable*. In our discussion we follow the notation used to describe it. Let P be a normal logic program and let $L \subseteq Lit(P)$. Let us consider the following recurrence relation:

$$s(P, L) = \begin{cases} 1 & \text{if } [P]_L = \emptyset \text{ or } L \text{ is not consistent} \\ \sum_{A \in \mathcal{A}} s(P, L \cup A) & \text{otherwise.} \end{cases}$$

As a corollary to Proposition 3 we obtain the following result.

Corollary 1
Let P be a finite normal logic program and let $L \subseteq Lit(P)$. Then, P has at most $s(P, L)$ stable models consistent with L. In particular, P has at most $s(P, \emptyset)$ stable models.

Proof
It is easy to see that $s(P, L)$ bounds the number of sets output by the algorithm *stable*(P, L). By Proposition 3, all stable models consistent with L are output by a call to *stable*(P, L). Thus, the first part of the assertion follows. Since each stable model of P is consistent with the empty set of literals, the second part of the assertion follows, as well. □

We will use the function $s(P, L)$ to estimate not only the number of stable models in normal logic programs but also the running time of the algorithm *stable*. Indeed, let us observe that the total number of times we make a call to the algorithm *stable* when executing *stable*(P, L) (including the "top-level" call to *stable*(P, L)) is given by $s(P, L)$. We associate each execution of the instruction (i), where $0 \leq i \leq 5$, with the call in which the instruction is executed. Consequently, each of these instructions is executed no more than $s(P, L)$ times during the execution of *stable*(P, L).

There are linear-time algorithms to check whether a set of atoms is a stable model of a program P. Thus, we obtain the following result concerning the performance of the algorithm *stable*.

Theorem 1
If the procedure *complete* runs in time $O(t(m))$, where t is a function defined on the set of positive integers and m is the size of an input program P, then executing the call *stable*(P, L), where $L \subseteq Lit(P)$, requires $O(s(P, L)(t(m)+m))$ steps in the worst case.

The specific bound depends on the procedure *complete*, as it determines the recurrence for $s(P, L)$. It also depends on the implementation of the procedure *complete*, as the implementation determines the second factor in the running-time formula derived above.

Throughout the paper (except for Section 8, where a different approach is used), we specify algorithms to compute stable models by describing particular versions of the procedure *complete*. We obtain estimates on the running time of these algorithms by analyzing the recurrence for $s(P, L)$ implied by the procedure *complete*. As a byproduct to these considerations, we obtain bounds on the maximum number of stable models of a logic program with n atoms.

4 t-programs

In this section we will instantiate the general algorithm to compute stable models to the case of t-programs, where $t \geq 2$. To this end, we will describe a procedure that, given a *normal* t-program P, returns a complete collection for P. The assumption of normality imposed on a t-program P implies that every clause in P has a non-empty head and consists of at most k literals (including the head).

Let P be a normal t-program and let $x \leftarrow \beta_1, \ldots, \beta_k$, where β_i are literals and $k \leq t-1$, be a clause in P. Let us define $A_0 = \{x\}$. Further, for every $i = 1, \ldots, k$, let us define

$$A_i = \{\mathbf{not}(x), \beta_1, \ldots, \beta_{i-1}, \mathbf{not}(\beta_i)\}$$

(we recall that **not**(β) denotes the literal that is dual to the literal β). It is easy to see that the family $\mathcal{A} = \{A_0, A_1, \ldots, A_k\}$ is complete for P. The family $\mathcal{A}$ was first used by Monien and Speckenmeyer (Monien and Speckenmeyer 1985) in algorithms to compute models of t-CNF propositional theories. It also appeared in the work of Bonatti and Olivetti on sequent calculus for default logic (Bonatti and Olivetti 2002). We will assume that this complete collection $\mathcal{A}$ is computed and returned by the procedure *complete*. Clearly, computing this collection $\mathcal{A}$ can be implemented to run in time $O(m)$.

To analyze the resulting algorithm *stable*, we use our general results from the previous section. Let us define

$$c_n = \begin{cases} K_t & \text{if } 0 \leq n < t \\ c_{n-1} + \ldots + c_{n-t} & \text{otherwise,} \end{cases}$$

where K_t is the maximum possible value of $s(P, L)$ for a normal t-program P and a set of literals $L \subseteq Lit(P)$ such that $|At(P)| - |L| \leq t$. It is easy to see that K_t is a constant that depends neither on P nor on L. We will prove that if P is a normal t-program, $L \subseteq Lit(P)$, and $|At(P)| - |L| \leq n$, then $s(P, L) \leq c_n$. We proceed by induction on n. If $n < t$, then the assertion follows by the definition of K_t. So, let us assume that $n \geq t$. If L is not consistent or $[P]_L = \emptyset$, $s(P, L) = 1 \leq c_n$. Otherwise,

$$s(P, L) = \sum_{i=0}^{k} s(P, L \cup A_i) \leq \sum_{i=1}^{k+1} c_{n-i} \leq \sum_{i=1}^{t} c_{n-i} = c_n.$$

The first equality follows from the recursive definition of $s(P, L)$. The first inequality follows by the induction hypothesis. Indeed, for every $i = 0, 1, \ldots, k$,

$$|At(P)| - |L \cup A_i| \leq |At(P)| - |L| - i \leq n - i.$$

The second inequality is straightforward and the second equality follows from the definition of the sequence c_n. Thus, the induction step is complete.

The characteristic equation of the recurrence relation $c_n = c_{n-1} + \ldots + c_{n-t}$ is $x^t = x^{t-1} + \ldots + x + 1$. Let α_t be the largest real root of this equation. One can show that for $t \geq 2$, $1 < \alpha_t < 2 - 1/2^t$. In particular, $\alpha_2 \approx 1.61803$, $\alpha_3 \approx 1.83931$, $\alpha_4 \approx 1.92757$ and $\alpha_5 \approx 1.96595$. Based on the discussion in Section 3, we obtain the following two theorems.

Theorem 2
Let t be an integer, $t \geq 2$. There is an algorithm to compute stable models of t-programs that runs in time $O(m\alpha_t^n)$, where n is the number of atoms and m is the size of the input program.

Theorem 3
Let t be an integer, $t \geq 2$. There is a constant C_t such that every t-program P has at most $C_t\alpha_t^n$ stable models, where $n = |At(P)|$.

Since for every t, $\alpha_t < 2$, we indeed obtain an improvement over the straightforward approach. However, the scale of the improvement diminishes as t grows.

To establish lower bounds on the number of stable models and on the worst-case

performance of algorithms to compute them, we define $P(n,t)$ to be a logic program such that $|At(P(n,t))| = n$ and $P(n,t)$ consists of all clauses of the form

$$x \leftarrow \mathbf{not}(b_1), \ldots, \mathbf{not}(b_t),$$

where $x \in At(P(n,t))$ and $\{b_1, \ldots, b_t\} \subseteq At(P(n,t)) \setminus \{x\}$ are different atoms. It is easy to see that $P(n,t)$ is a $(t{+}1)$-program with n atoms and that stable models of $P(n,t)$ are precisely those subsets of $At(P(n,t))$ that have $n{-}t$ elements. Thus, $P(n,t)$ has exactly $\binom{n}{t}$ stable models.

Clearly, the program $P(2t{-}1, t{-}1)$ is a t-program over the set of $2t{-}1$ atoms. Moreover, it has $\binom{2t-1}{t-1} (= \binom{2t-1}{t})$ stable models and each of these stable models has t elements. Let $kP(2t{-}1, t{-}1)$ be the logic program formed by the disjoint union of k copies of $P(2t{-}1, t{-}1)$ (sets of atoms of different copies of $P(2t{-}1, t{-}1)$ are disjoint). Let us denote this program by $Q(k,t)$. It is easy to establish the following properties of the program $Q(k,t)$:

1. $|At(Q(k,t))| = k(2t{-}1)$
2. The total length of all clauses in $Q(k,t)$ (the size of $Q(k,t)$) is given by $k(t{+}1)(2t{-}1)\binom{2t-1}{t}$
3. $Q(k,t)$ has $\binom{2t-1}{t}^k$ stable models, each of them of cardinality kt.

Let us define $\mu_t = \binom{2t-1}{t}^{1/2t-1}$. These properties imply the following result.

Theorem 4
Let t be an integer, $t \geq 2$. There are positive constants d_t and D_t such that for every $n \geq 2t{-}1$ there is a t-program P with n atoms and such that

1. The size of P, m, satisfies $m \leq d_t n$
2. The sum of the cardinalities of all stable models of P is at least $D_t n \mu_t^n$.

As a corollary to this theorem, we obtain the following result.

Corollary 2
Let t be an integer, $t \geq 2$.

1. Every algorithm computing all stable models of t-programs requires in the worst case at least $\Omega(n\mu_t^n)$ steps
2. Let $0 < \alpha < \mu_t$. There is no algorithm for computing all stable models of t-programs with worst-case performance bounded by $O(f(m)\alpha^n)$, where f is a polynomial and m is the size of the input program.

Proof
The first part of the assertion follows by Theorem 4(2). Indeed, any algorithm computing all stable models of t-programs needs $n\mu_t^n$ to output the results of the computation when run on the programs discussed in Theorem 4.

For the second part of the assertion, let us assume that there is α, $0 < \alpha < \mu_t$, and an algorithm A computing all stable models of t-programs such that A runs in time $O(f(m)\alpha^n)$, for some polynomial f. For programs discussed in Theorem 4, $m = O(n)$. Thus, it follows from part (1) that $n\mu_t^n = O(f(n)\alpha^n)$, a contradiction. □

The lower bound given by Corollary 2 specializes to (approximately) $\Omega(n \times 1.44224^n)$, $\Omega(n \times 1.58489^n)$, $\Omega(n \times 1.6618^n)$ and $\Omega(n \times 1.71149^n)$, for $t = 2, 3, 4, 5$, respectively.

5 2-programs

Stronger results than those obtained in the previous section can be derived for more restricted classes of programs. In this section we study the case of 2-programs and prove the following two theorems.

Theorem 5
There is an algorithm to compute stable models of 2-programs that runs in time $O(m3^{n/3}) = O(m \times 1.44225^n)$, where n is the number of atoms in P and m is the size of P.

Theorem 6
There is a constant C such that every 2-program P with n atoms, has at most $C \times 3^{n/3}$ ($\approx C \times 1.44225^n$) stable models.

By Proposition 1, to prove these theorems it suffices to limit attention to the case of normal programs not containing tautologies and virtual constraints. We will adopt this assumption and derive both theorems from general results presented in Section 3.

Let P be a normal 2-program. We say that an atom $b \in At(P)$ is a *neighbor* of an atom $a \in At(P)$ if P contains a clause containing both a and b (one of them as the head, the other one appearing positively or negatively in the body). By $n(a)$ we will denote the number of neighbors of an atom a. Since we assume that our programs contain neither tautologies nor virtual constraints, no atom a is its own neighbor.

We will now describe the procedure *complete*. The complete family returned by the call to *complete*(P) depends on the program P. We list below several cases that cover all normal 2-programs without tautologies and virtual constraints. In each of these cases, we specify a complete collection to be returned by the procedure *complete*.

Case 1. There is an atom, say x, such that P contains a clause with the head x and with the empty body (in other words, x is a fact of P). We define $\mathcal{A} = \{\{x\}\}$. Clearly, every stable model of P contains x. Thus, $\mathcal{A}$ is complete.

Case 2. There is an atom, say x, that does not appear in the head of any clause in P. We define $\mathcal{A} = \{\{\mathbf{not}(x)\}\}$. It is well known that x does not belong to *any* stable model of P. Thus, $\mathcal{A}$ is complete for P.

Case 3. There are atoms x and y, $x \neq y$, such that $x \leftarrow y$ and at least one of $x \leftarrow \mathbf{not}(y)$ and $y \leftarrow \mathbf{not}(x)$ are in P. In this case, we set $\mathcal{A} = \{\{x\}\}$. Let M be a stable model of P. If $y \in M$, then $x \in M$ (due to the fact that the clause $x \leftarrow y$ is in P). Otherwise, $y \notin M$. Since M satisfies $x \leftarrow \mathbf{not}(y)$ or $y \leftarrow \mathbf{not}(x)$, it again follows that $x \in M$. Thus, $\mathcal{A}$ is complete.

Case 4. There are atoms x and y such that $x \leftarrow y$ and $y \leftarrow x$ are both in P. We define

$$\mathcal{A} = \{\{x, y\}, \{\mathbf{not}(x), \mathbf{not}(y)\}\}.$$

If M is a stable model of P then, clearly, $x \in M$ if and only if $y \in M$. It follows that either $\{x, y\} \subseteq M$ or $\{x, y\} \cap M = \emptyset$. Thus, $\mathcal{A}$ is complete for P. Moreover, since $x \neq y$ (P does not contain clauses of the form $w \leftarrow w$), each set in $\mathcal{A}$ has at least two elements.

Case 5. None of the Cases 1-4 holds and there is an atom, say x, with exactly one neighbor, y. Since P does not contain clauses of the form $w \leftarrow w$ and $w \leftarrow \mathbf{not}(w)$, we have $x \neq y$. Moreover, x must be the head of at least one clause (since we assume here that Case 2 does not hold).

Subcase 5a. P contains the clause $x \leftarrow y$. We define

$$\mathcal{A} = \{\{x, y\}, \{\mathbf{not}(x), \mathbf{not}(y)\}\}.$$

Let M be a stable model of P. If $y \in M$ then, clearly, $x \in M$. Since we assume that Case 3 does not hold, the clause $x \leftarrow y$ is the only clause in P with x as the head. Thus, if $y \notin M$, then we also have that $x \notin M$. Hence, $\mathcal{A}$ is complete.

Subcase 5b. P does not contain the clause $x \leftarrow y$. We define

$$\mathcal{A} = \{\{x, \mathbf{not}(y)\}, \{\mathbf{not}(x), y\}\}.$$

Let M be a stable model of P. Since x is the head of at least one clause in P, it follows that the clause $x \leftarrow \mathbf{not}(y)$ belongs to P. Thus, if $y \notin M$ then $x \in M$. If $y \in M$ then, since $x \leftarrow \mathbf{not}(y)$ is the only clause in P with x as the head, $x \notin M$. Hence, $\mathcal{A}$ is complete.

Case 6. None of the Cases 1-5 holds. Let $w \in At(P)$ be an atom. By $x_1, \ldots, x_p$ we denote all atoms x in P such that $w \leftarrow \mathbf{not}(x)$ or $x \leftarrow \mathbf{not}(w)$ is a clause in P. Similarly, by $y_1, \ldots, y_q$ we denote all atoms y in P such that $y \leftarrow w$ is a clause of P. Finally, by $z_1, \ldots, z_r$ we denote all atoms z of P such that $w \leftarrow z$ is a clause of P. By our earlier discussion it follows that the sets $\{x_1, \ldots, x_p\}$, $\{y_1, \ldots, y_q\}$ and $\{z_1, \ldots, z_r\}$, are pairwise disjoint and cover all neighbors of w. That is, the number of neighbors of w is given by $p+q+r$. Since we exclude Case 5 here, $p+q+r \geq 2$. Further, since w is the head of at least one edge (Case 2 does not hold), it follows that $p+r \geq 1$

Subcase 6a. For some atom w, $q \geq 1$ or $p+q+r \geq 3$. Then, we define

$$\mathcal{A} = \{\{w, y_1, \ldots, y_q\}, \{\mathbf{not}(w), x_1, \ldots, x_p, \mathbf{not}(z_1), \ldots, \mathbf{not}(z_r)\}\}.$$

It is easy to see that $\mathcal{A}$ is complete for P. Moreover, if $q \geq 1$ then, since $p+r \geq 1$, each of the two sets in $\mathcal{A}$ has at least two elements. If $p+q+r \geq 3$, then either each set in $\mathcal{A}$ has at least two elements, or one of them has one element and the other one at least four elements.

Subcase 6b. Every atom w has exactly two neighbors, and does not appear in the body of any Horn clause of P. It follows that all clauses in P are purely negative. Let w be an arbitrary atom in P. Let u and v be the two neighbors of w. The

atoms u and v also have two neighbors each, one of them being w. Let u' and v' be the neighbors of u and v, respectively, that are different from w. We define

$$\mathcal{A} = \{\{\mathbf{not}(w), u, v\}, \{\mathbf{not}(u), w, u'\}, \{\mathbf{not}(v), w, v'\}\}.$$

Let M be a stable model of P. Let us assume that $w \notin M$. Since w and u are neighbors, there is a clause in P built of w and u. This clause is purely negative and it is satisfied by M. It follows that $u \in M$. A similar argument shows that $v \in M$, as well. If $w \in M$ then, since M is a stable model of P, there is a 2-clause C in P with the head w and with the body satisfied by M. Since P consists of purely negative clauses, and since u and v are the only neighbors of w, $C = w \leftarrow \mathbf{not}(u)$ or $C = w \leftarrow \mathbf{not}(v)$. Let us assume the former. It is clear that $u \notin M$ (since M satisfies the body of C). Let us recall that u' is a neighbor of u. Consequently, u and u' form a purely negative clause of P. This clause is satisfied by M. Thus, $u' \in M$ and M is consistent with $\{\mathbf{not}(u), w, u'\}$. In the other case, when $C = w \leftarrow \mathbf{not}(v)$, a similar argument shows that M is consistent with $\{\mathbf{not}(v), w, v'\}$. Thus, every stable model of P is consistent with one of the three sets in $\mathcal{A}$. In other words, $\mathcal{A}$ is complete.

Clearly, given a normal 2-program P, deciding which of the cases described above holds for P can be implemented to run in linear time. Once that is done, the output collection can be constructed and returned in linear time, too.

This specification of the procedure *complete* yields a particular algorithm to compute stable models of normal 2-programs without tautologies and virtual constraints. To estimate its performance and obtain the bound on the number of stable models, we define

$$c_n = \begin{cases} K & \text{if } 0 \leq n < 4 \\ \max\{c_{n-1}, 2\,c_{n-2}, c_{n-1}+c_{n-4}, 3c_{n-3}\} & \text{otherwise,} \end{cases}$$

where K is the maximum possible value of $s(P, L)$, when P is a normal finite propositional logic program, $L \subseteq Lit(P)$ and $|At(P)|-|L| \leq 3$. It is easy to see that K is a constant that depends neither on P nor on L. We will prove that $s(P, L) \leq c_n$, where $n = |At(P)|-|L|$. If $n \leq 3$, then the assertion follows by the definition of K. So, let us assume that $n \geq 4$. If L is not consistent or $[P]_L = \emptyset$, $s(P, L) = 1 \leq c_n$. Otherwise,

$$s(P, L) = \sum_{A \in \mathcal{A}} s(P, L \cup A) \leq \max\{c_{n-1}, 2\,c_{n-2}, c_{n-1}+c_{n-4}, 3c_{n-3}\} = c_n.$$

The inequality follows by the induction hypothesis, the properties of the complete families returned by *complete* (the cardinalities of sets forming these complete families) and the monotonicity of c_n.

Using well-known properties of linear recurrence relations, it is easy to see that $c_n = O(3^{n/3}) = O(1.44225^n)$. Thus, Theorems 5 and 6 follow.

As concerns bounds on the number of stable models of a 2-program, a stronger (exact) result can be derived.

Let P be a 2-program that is normal, purely negative and contains no facts. Let $G(P)$ be a graph such that $At(P)$ is the vertex set of $G(P)$ and a and b are

connected with an edge if they appear in the same clause of P (if they are neighbors, in the terminology we used earlier). We recall that a subset X of the vertex set of a graph G is *independent* if no two vertices of X are connected in G with an edge. We have the following simple property.

Proposition 4
Let P be a normal purely-negative 2-program not containing facts. If M is a stable model of P then $At(P) \setminus M$ is a maximal independent set of $G(P)$.

The problem of finding the maximum number of independent sets in a graph was investigated in (Moon and Moser 1965). For $n \geq 1$, let us define

$$g_n = \begin{cases} 3^{n/3} & \text{if } n = 0 \pmod 3 \\ 4 \times 3^{(n-4)/3} & \text{if } n = 1 \pmod 3, \text{ and } n > 1 \\ 2 \times 3^{(n-2)/3} & \text{if } n = 2 \pmod 3 \\ 1 & \text{if } n = 1 \end{cases}$$

The following result is proved in (Moon and Moser 1965).

Theorem 7
Let G be a graph with n vertices. Then G has no more than g_n maximal independent sets.

Proposition 4 and Theorem 7 imply the following corollary concerning the number of stable models in 2-programs.

Corollary 3
Let P be a 2-program with n atoms. Then P has no more than g_n stable models.

Proof
By Proposition 1, we may assume that P is a normal program not containing rules of the form $x \leftarrow x$ and $x \leftarrow \mathbf{not}(x)$. By Proposition 2 we may assume that P does not contain facts. Lastly, we may assume that P contains no Horn clauses (Horn clauses can be eliminated in the process of *unfolding* without introducing new atoms and without changing the number of stable models (Bonatti and Eiter 1996)). Due to these assumptions, Proposition 4 applies. Thus, P has no more stable models as there are maximal independent sets in the graph $G(P)$. By Theorem 7, this number is at most g_n. □

The bound of Corollary 3 cannot be improved as there are logic programs that achieve it. Let $P(p_1, \ldots, p_k)$, where for every i, $p_i \geq 2$, be a disjoint union of programs $P(p_1, 1), \ldots, P(p_k, 1)$ (we discussed these programs in Section 2). Each program $P(p_i, 1)$ has p_i stable models. Thus, the number of stable models of $P(p_1, \ldots, p_k)$ is $p_1 p_2 \ldots p_k$. Let P be a logic program with $n \geq 2$ atoms and of the form $P(3, \ldots, 3)$, $P(2, 3, \ldots, 3)$ or $P(4, 3, \ldots, 3)$, depending on $n \pmod 3$. It is easy to see that P has g_n stable models.

It is also easy to see that our algorithm for computing all stable models of 2-programs is optimal. Indeed, Corollary 2 (for $t = 2$) implies the following result.

Corollary 4
Let $0 < \alpha < 3^{1/3} \approx 1.44225$. There is no algorithm for computing all stable models of 2-programs with worst-case performance bounded by $O(f(m)\alpha^n)$, where f is a polynomial and m is the size of the input program.

Narrowing the class of programs leads to still better bounds and faster algorithms. We will discuss one specific subclass of the class of 2-programs here. Namely, we will consider normal purely negative 2-programs with no *dual* clauses (two clauses are called *dual* if they are of the form $a \leftarrow \mathbf{not}(b)$ and $b \leftarrow \mathbf{not}(a)$). We denote the class of these programs by $\mathcal{P}_2^n$. We obtain the following two theorems for this class.

Theorem 8
There is an algorithm to compute stable models of 2-programs in the class $\mathcal{P}_2^n$ that runs in time $O(m \times 1.23651^n)$, where n is the number of atoms and m is the size of an input program.

Theorem 9
There is a constant C such that every 2-program $P \in \mathcal{P}_2^n$ has at most $C \times 1.23651^n$ stable models.

As before, when discussing algorithms to compute stable models of programs in this class, we restrict our attention even further by disallowing virtual constraints (in this case, clauses of the form $a \leftarrow \mathbf{not}(a)$). By Proposition 1, this additional assumption does not affect the generality of our results.

To discuss this specific case, we need more notation. We say that an atom $a \in At(P)$ is an *in-neighbor* of an atom $b \in At(P)$ if P contains the clause $b \leftarrow \mathbf{not}(a)$. In such case we also say that b is an *out-neighbor* of a. By $n^-(a)$ and $n^+(a)$ we denote the number of in-neighbors and out-neighbors of a, respectively. Thus, we have $n(a) = n^-(a)+n^+(a)$ (we recall that $n(a)$ denotes the number of neighbors of a in P). We also define

$$N(a) = \{\mathbf{not}(a), b_1, \ldots, b_k\},$$

where b_i, $1 \leq i \leq k$, are all the neighbors of a. Clearly, if M is a stable model of P and $a \notin M$ then M is consistent with $N(a)$. Indeed, since P contains either the clause $a \leftarrow \mathbf{not}(b_i)$ or $b_i \leftarrow \mathbf{not}(a)$, and since M is a model of P, $b_i \in M$.

Let $a \in At(P)$ and let c_i, $1 \leq i \leq p$, be the in-neighbors of a enumerated so that $n(c_1) \geq \ldots \geq n(c_p)$. It is easy to see that the following collections are complete:

$$\mathcal{A}^1(a) = \{N(a), \{a\}\}$$

and

$$\mathcal{A}^2(a) = \{N(a), N(c_1), N(c_2) \cup \{c_1\}, \ldots, N(c_p) \cup \{c_1, \ldots, c_{p-1}\}\}.$$

To specify the procedure *complete*, we proceed along the same lines as before. That is, we describe several cases that together cover all programs in the class of interest and in each of them we specify a complete collection of sets to be returned

by the procedure *complete*. For each of these collections we will also specify its *signature*. Let $\mathcal{A} = \{A_1, \ldots, A_k\}$ be a complete collection and let us assume that for every i, $1 \leq i \leq k-1$, $|A_i| \geq |A_{i+1}|$. We call the sequence of cardinalities of sets A_i, $(|A_1|, \ldots, |A_k|)$, the *signature* of $\mathcal{A}$. We denote it by $sig(\mathcal{A})$. We say that a signature $(a_1, \ldots, a_k)$ is *bounded* by a sequence $(b_1, \ldots, b_k)$ if $a_i \geq b_i$, for every $i = 1, \ldots k$.

Case 1 and **Case 2** are as in the general case considered above. In each case, the complete family $\mathcal{A}$ returned by the procedure *complete* consists of exactly one set that contains exactly one element. Thus, $sig(\mathcal{A}) = (1)$.

Case 3. There is an atom, say x, with at least seven neighbors, say $y_1, \ldots, y_k$, where $k \geq 7$. The procedure *complete* returns in this case the collection $\mathcal{A} = \mathcal{A}^1(x)$. Clearly, $sig(\mathcal{A}) = (8, 1)$.

From now on we will assume that Cases 1-3 do not hold. That is, we adopt the following assumption:

(A1) For every atom $x \in At(P)$, $n(x) \leq 6$ and $1 \leq n^-(x)$.

Case 4. There is an atom x such that $n(x) \geq 3$ and $n^-(x) = 1$. Let us assume that y is the only in-neighbor of x. By (A1), $n^-(y) \geq 1$ and, since x is an out-neighbor of y, $n^+(y) \geq 1$. Thus, $n(y) \geq 2$. The procedure *complete* returns

$$\mathcal{A} = \{N(x), N(y)\}.$$

Let M be a stable model of P. If $x \notin M$ then M is consistent with $N(x)$. If $x \in M$ then $y \notin M$ (due to the fact that $x \leftarrow \mathbf{not}(y)$ is the only rule in P with the head x). Thus, M is consistent with $N(y)$. It follows that $\mathcal{A}$ is complete and $sig(\mathcal{A})$ is bounded by $(4, 3)$.

Case 5. There is an atom x such that $n(x) = 2$ and $n^+(x) \geq 1$. By (A1), it follows that $n^-(x) = n^+(x) = 1$. Let us define $x_0 = x$. Let us assume that the atoms $x_0, \ldots, x_k$, $0 \leq k$, have been defined and that for every i, $0 \leq i \leq k-1$, $n(x_i) = 2$ and x_{i+1} is the (only) in-neighbor of x_i. If $n(x_k) = 2$ and $x_k \neq x_0$, we define x_{k+1} to be the unique in-neighbor of x_k. If $n(x_k) \geq 3$, or $n(x_k) = 2$ and $x_k = x_0$, the procedure terminates. Let us assume that the procedure terminated after x_k had been defined. We have the following possibilities.

Subcase 5a. $n(x_k) \geq 3$. We define

$$\mathcal{A} = \{N(x_k), N(x_{k-1})\}.$$

Obviously $\mathcal{A}$ is complete. Since $n(x_k) \geq 3$ and $n(x_{k-1}) \geq 2$, the signature of $\mathcal{A}$ is bounded by $(4, 3)$.

Subcase 5b. $x_k = x_0$. In this case, the atoms $x_0, \ldots, x_{k-1}$ form a cycle component of P. That is, $P = Q \cup \{x_i \leftarrow \mathbf{not}(x_{i+1}) \colon i = 0, 1, \ldots, k-1\}$, where no rule in Q contains any atom x_i.

If k is odd, we define

$$\mathcal{A} = \{At(P)\}.$$

In this case, P has no stable models (programs forming "odd cycles" do not have stable models and atoms $x_0, \ldots, x_{k-1}$ do not appear in any other clauses of P but

those that form the cycle). Thus, $\mathcal{A}$ is trivially complete. Clearly, $sig(\mathcal{A}) = (|At(P)|)$ and it is bounded by (1).

If k is even, we define

$$\mathcal{A} = \{\{x_0, x_2, \ldots, x_{k-2}, \mathbf{not}(x_1), \mathbf{not}(x_3), \ldots, \mathbf{not}(x_{k-1})\},$$
$$\{\mathbf{not}(x_0), \mathbf{not}(x_2), \ldots, \mathbf{not}(x_{k-2}), x_1, x_3, \ldots, x_{k-1}\}\}.$$

If M is a stable model of P then M is consistent with one of the sets in $\mathcal{A}$. Since P has no dual clauses, each set in $\mathcal{A}$ has at least 4 elements. That is, $sig(\mathcal{A})$ is bounded by $(4, 4)$.

Case 6. There is an atom, say x, with in-neighbors $y_1, \ldots, y_k$ and out-neighbors $z_1, \ldots, z_m$, and such that $k < m$. Moreover, we assume that Cases 1-5 do not hold.

If $k = 1$ then $m \geq 2$. This possibility is excluded as it is covered by Case 4. Since $k+m \leq 6$, $k+m = 5$ or 6 and $k = 2$. Let the two in-neighbors of x be y_1 and y_2. Without loss of generality we will assume that $n(y_1) \geq n(y_2)$. By (A1), $n(y_2) \geq 2$. Since $n^+(y_2) \geq 1$ (x is an out-neighbor of y_2) and since Case 5 is excluded, it follows that $n(y_2) \geq 3$.

Subcase 6a. $n(y_1) \geq 4$, $n(y_2) \geq 4$. In this case, the procedure *complete* returns the family $\mathcal{A}^2(x)$. It is easy to see that its signature is bounded by $(6, 5, 5)$.

Subcase 6b. $n(y_1) \geq 4$, $n(y_2) = 3$ and $y_1 \notin N(y_2)$. The procedure again returns $\mathcal{A}^2(x)$. Since $y_1 \notin N(y_2)$, the signature of $\mathcal{A}^2(x)$ is bounded by $(6, 5, 5)$.

Subcase 6c. $n(y_1) \geq 4$, $n(y_2) = 3$ and $y_1 \in N(y_2)$. Let z denote the third neighbor of y_2 (the other two are x and y_1). Since Case 4 is excluded and since x is an out-neighbor of y_2, y_1 and z are in-neighbors of y_2. We define

$$\mathcal{A} = \{N(x) \cup \{\mathbf{not}(z)\}, N(y_1), N(y_2)\}.$$

Let M be a stable model of P. If it is consistent with neither $N(y_1)$ nor $N(y_2)$, then it is consistent with $N(x)$. In this case, $y_1, y_2 \in M$. Since $y_2 \in M$, there is a rule in P with the head y_2 and with the body satisfied by M. Since y_2 has exactly two in-neighbors, y_1 and z and since $y_1 \in M$, it follows that $z \notin M$. Thus, $\mathcal{A}$ is complete. Moreover, it is easy to check that $z \notin N(x)$ so the signature of $\mathcal{A}$ is $(7, 5, 4)$.

Subcase 6d. $n(x) = 6$ and $n(y_1) = n(y_2) = 3$. The procedure *complete* returns $\mathcal{A} = \mathcal{A}^2(x)$. Since y_1 and y_2 are not neighbors (otherwise, there would be an atom in P with 3 neighbors, exactly one of which is an in-neighbor), its signature is, clearly, $(7, 5, 4)$.

Let us assume that none of the cases 1-6 applies. We denote by X the set of all those atoms x in P for which $n^-(x) < n^+(x)$. It follows from the considerations in Case 6 that every $x \in X$ has exactly 5 neighbors. Moreover, exactly two of these neighbors, say y_1^x and y_2^x, are in-neighbors and $n(y_1^x) = n(y_2^x) = 3$ (this is the only possibility not covered by Case 6). We denote by Y the set of all atoms y_i^x, $i = 1, 2$, where $x \in X$. Clearly, $X \cap Y = \emptyset$ and $2|X| = |Y|$ (indeed, let us note that since

Case 3 is excluded, $y_i^{x_1} \neq y_j^{x_2}$, for $1 \leq i, j \leq 2$ and $x_1 \neq x_2$). Finally, let us define $Z = At(P) \setminus (X \cup Y)$. We have the following identities:

$$\sum_{a \in At(P)} n^-(a) = \sum_{a \in At(P)} n^+(a),$$

$$\sum_{z \in Z} n^-(z) \geq \sum_{z \in Z} n^+(z),$$

$$\sum_{a \in At(P)} n^-(a) = 6|X| + \sum_{z \in Z} n^-(z),$$

$$\sum_{a \in At(P)} n^+(a) = 5|X| + \sum_{z \in Z} n^+(z).$$

These identities imply that $|X| = 0$. In other words, for every atom a, $n^+(a) \leq n^-(a)$. Since $\sum_{a \in At(P)} n^-(a) = \sum_{a \in At(P)} n^+(a)$, it follows that for every atom a, $n^+(a) = n^-(a)$.

Since the case $n^+(a) = n^-(a) = 1$ is excluded (Case 5), it follows that for every atom a in P, $n(a) = 4$ or 6. For $i = 4, 6$, we define $X_i = \{x \in At(P) \colon n(x) = i\}$. Let p be the number of clauses in P built only of atoms from X_4, let q be the number of clauses of the form $a \leftarrow \mathbf{not}(b)$, where $a \in X_6$ and $b \in X_4$, and finally, let r be the number of clauses in P of the form $a \leftarrow \mathbf{not}(b)$, where $a \in X_4$ and $b \in X_6$. Then,

$$p + r = \sum_{a \in X_4} n^-(a) = 2|X_4| = \sum_{a \in X_4} n^+(a) = p + q.$$

Thus, $q = r$.

Case 7. $r > 0$. There is an atom x with four neighbors and such that one of its in-neighbors has 6 neighbors. The procedure returns $\mathcal{A}^2(x)$. Since the other in-neighbor of x (we observe that $n^-(x) = 2$) has at least four neighbors, the signature of $\mathcal{A}^2(x)$ is bounded by $(7, 5, 5)$.

Case 8. $r = 0$ and there is an atom, say x, with 6 neighbors. Let the three in-neighbors of x be y_1, y_2 and y_3. Since $r = 0$, $q = 0$. Thus, for every rule $a \leftarrow \mathbf{not}(b)$ in P either $a, b \in X_4$ or $a, b \in X_6$. Consequently, each y_i, $i = 1, 2, 3$, has 6 neighbors. The procedure returns $\mathcal{A}^2(x)$. It is easy to see that $sig(\mathcal{A}^2(x)) = (7, 7, 7, 7)$.

Case 9. $r = 0$ and there is no atom in P with six neighbors. It follows that every atom in P has four neighbors. Moreover, exactly two of them are in-neighbors and two are out-neighbors. Let y_1 and y_2 be the two in-neighbors of x.

Subcase 9a. y_1 and y_2 are not neighbors (do not appear in the same clause). The procedure returns $\mathcal{A}^2(x)$. Clearly, its signature is $(6, 5, 5)$.

Subcase 9b. y_1 and y_2 are neighbors. Without loss of generality we may assume that $y_1 \leftarrow \mathbf{not}(y_2)$ is a clause of P. Let z be the other in-neighbor of y_1. If z is an out-neighbor of x, the procedure returns

$$\mathcal{A} = \{\{x\}\}.$$

Let M be a stable model of P. If $x \notin M$ then $y_2 \in M$, $y_1 \in M$ and $z \in M$. Since both in-neighbors of y_1 are in M, $y_1 \notin M$, a contradiction. Thus, $x \in M$. It follows that every stable model of P is consistent with $\{x\}$.

Otherwise, we define

$$\mathcal{A} = \{N(x) \cup \{\mathbf{not}(z)\}, N(y_1), N(y_2)\}.$$

Let M be a stable model of P. If y_1 or y_2 are not in M, M is consistent with $N(y_1)$ or $N(y_2)$, respectively. Otherwise, $y_1, y_2 \in M$. Consequently, $x \notin M$. Further $z \notin M$ (the other in-neighbor of y_1, y_2, is in M). Thus, M is consistent with $N(x) \cup \{\mathbf{not}(z)\}$. It follows that $\mathcal{A}$ is complete and that its signature is $(6, 5, 5)$.

It is clear that the procedure *complete* is correct. Indeed, the cases cover all possible normal purely negative 2-programs without virtual constraints. Taking into account other comments made in each case, we see that the following property of the procedures *complete* is true. Let P be a 2-program from $\mathcal{P}_2^n$ and let $\mathcal{A}$ be a complete collection produced by the call *complete*(P). Then $|\mathcal{A}| \leq 4$ and

1. if $\mathcal{A} = \{A\}$, then the signature of $\mathcal{A}$ is (1);
2. if $|\mathcal{A}| = 2$, then the signature of $\mathcal{A}$ is bounded by $(8, 1)$ or $(4, 3)$;
3. if $|\mathcal{A}| = 3$, then the signature of $\mathcal{A}$ is bounded by $(6, 5, 5)$ or $(7, 5, 4)$;
4. if $|\mathcal{A}| = 4$, then the signature of $\mathcal{A}$ is $(7, 7, 7, 7)$.

Let us define

$$c_n = \begin{cases} K & \text{if } 0 \leq n < 8 \\ \max\{c_n^i : i = 1, \ldots, 6\} & \text{otherwise,} \end{cases}$$

where K is the maximum possible value of $s(P, L)$ for a program P in $\mathcal{P}_2^n$ and a set of literals $L \subseteq Lit(P)$ such that $|At(P)|-|L| \leq 8$, and where c_n^i are defined as follows (for $n \geq 8$):

$$c_n^1 = c_{n-1}^1,$$
$$c_n^2 = c_{n-1}^2 + c_{n-8}^2,$$
$$c_n^3 = c_{n-3}^3 + c_{n-4}^3,$$
$$c_n^4 = 2c_{n-5}^4 + c_{n-6}^4,$$
$$c_n^5 = c_{n-4}^5 + c_{n-5}^5 + c_{n-7}^5,$$
$$c_n^6 = 4c_{n-7}^6.$$

It is easy to see that K is a constant that does not depend on P nor L. Reasoning as before, one can show that

$$s(P, L) \leq c_n,$$

where $n = |At(P)|-|L|$.

Using the properties of linear relations discussed in the appendix, it is easy to see that $c_n = O(\alpha^n)$, where α is the only root in the interval $[1, 2]$ of the equation $x^7 = x^3 + x^2 + 1$. Since $\alpha \leq 1.23651$, $c_n = O(1.23651^n)$.

Theorem 9 gives an upper bound on the number of stable models of a program in the class $\mathcal{P}_2^n$. To establish a lower bound, we reason as before but use a different collection of programs. namely, we define S_6 to be a program over the set of atoms $a_0, \ldots, a_5$ and containing the rules (the arithmetic of indices is performed modulo 6): $a_{i+1} \leftarrow \mathbf{not}(a_i)$ and $a_{i+2} \leftarrow \mathbf{not}(a_i)$, $i = 0, 1, 2, 3, 4, 5$. The program S_6 has three stable models: $\{a_0, a_1, a_3, a_4\}$, $\{a_1, a_2, a_4, a_5\}$ and $\{a_2, a_3, a_5, a_0\}$.

Let P be the program consisting of k copies of S_6, with mutually disjoint sets of atoms. Clearly, P has 3^k stable models, each of them with $4k$ elements. Thus, there is a constant D such that for every $n \geq 1$ there is a program P with n atoms and with the total length of all its stable models at least $Dn(3^{1/6})^n$. Reasoning as in the proof of Corollary 4, we obtain the following result.

Corollary 5
Let $0 < \alpha < 3^{1/6} \approx 1.20094$. There is no algorithm for computing all stable models of 2-programs with worst-case performance bounded by $O(f(m)\alpha^n)$, where f is a polynomial and m is the size of the input program.

6 3-programs

In this section we present our results for the class of 3-programs. Using similar techniques as those presented in the previous section, we prove the following two theorems.

Theorem 10
There is an algorithm to compute stable models of 3-programs that runs in time $O(m \times 1.70711^n)$, where m is the size of the input.

Theorem 11
There is a constant C such that every 3-program P has at most $C \times 1.70711^n$ stable models.

In order to construct a desired algorithm we will design a particular implementation of the procedure *complete*. Due to Proposition 1, we will restrict our attention to normal 3-programs without tautologies (that is, rules of the form $x \leftarrow x$, $x \leftarrow x, y$, $x \leftarrow x, \mathbf{not}(y)$, and $x \leftarrow y, \mathbf{not}(y)$) and virtual constraints. To describe the procedure *complete*, we will consider several cases covering the class of all such 3-programs. In each of these cases, we construct a complete collection $\mathcal{A}$ (one of eighteen collections $\{\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_{18}\}$).

With each collection $\mathcal{A}$ we associate a sequence $(c_n^{\mathcal{A}})_{n=1,2,\ldots}$. To define it, we set $p = \max\{|A|\colon A \in \mathcal{A}\}$ and $k(t) = |\{A \in \mathcal{A} : |A| = t\}|$, for $t = 1, 2, \ldots, p$. We denote by K the maximum possible value of $s(P, L)$, when P is a normal 3-program without tautologies and virtual constraints, $L \subseteq Lit(P)$ and $|At(P)| - |L| \leq 12$. It is easy to see that K is a constant that does not depend on P nor L. We define

$$c_n^{\mathcal{A}} = \begin{cases} K & \text{if } 0 \leq n \leq 12 \\ \sum_{t=1}^{p} k(t)\, c_{n-t}^{\mathcal{A}} & \text{otherwise.} \end{cases}$$

Finally, we define $c_n = \max\{c_n^{\mathcal{A}} : \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_{18}\}\}$. We will show that

$$s(P, L) \leq c_n,$$

where $n = |At(P)| - |L|$.

We will proceed by induction on n. If $n \leq 12$, then the assertion follows by the definition of K. So, let us assume that $n \geq 13$. Then, by the induction hypothesis,

$$s(P, L) = \sum_{A \in \mathcal{A}} s(P, L \cup A) \leq \max\{\sum_{t=1}^{p} k(t) c_{n-t}^{\mathcal{A}} : \mathcal{A} \in \{\mathcal{A}_1, \mathcal{A}_2 \ldots \mathcal{A}_{18}\}\} = c_n.$$

Using properties of linear recurrence relations discussed in the appendix, one can show that $c_n = O(1.70711^n)$.

Thus, to complete the proofs of Theorems 10 and 11 we need to describe *complete*. According to our earlier statements, we restrict our discussion to normal 3-programs without tautologies and virtual constraints. To describe complete families of sets we introduce the following notation. Let A and L be sets of literals based on two disjoint sets of atoms ($A^0 \cap L^0 = \emptyset$). We define

$$A_L = \{A \cup \overline{B} \colon B \subseteq L\},$$

where $\overline{B} = B \cup \{\mathbf{not}(\beta) \colon \beta \in L \setminus B\}$ (we recall again that $\mathbf{not}(\beta)$ denotes the literal that is dual to β). For example, if $A = \{\mathbf{not}(a)\}$ and $L = \{b, \mathbf{not}(c)\}$, then

$$\begin{aligned} A_L \quad = \quad & \{\{\mathbf{not}(a), \mathbf{not}(b), c\}, \{\mathbf{not}(a), \mathbf{not}(b), \mathbf{not}(c)\}, \\ & \{\mathbf{not}(a), b, c\}, \{\mathbf{not}(a), b, \mathbf{not}(c)\}\}. \end{aligned}$$

When $L = \{\beta_1, \ldots, \beta_k\}$, to simplify this notation we write $A_{\beta_1,\ldots,\beta_k}$ instead of $A_{\{\beta_1,\ldots,\beta_k\}}$.

When describing the procedure *complete* and arguing its correctness, we will take advantage of the following simple observation (its proof is evident and we omit it).

Proposition 5
Let $\mathcal{A}$ be a complete collection for P, let $A \in \mathcal{A}$ and let B be a set of literals. If for every stable model M of P that is consistent with A, M is also consistent with B then the collection $(\mathcal{A} \setminus \{A\}) \cup \{A \cup B\}$ is also complete for P.

Speaking informally, under the assumptions of this proposition, we can replace in $\mathcal{A}$ its element A with $A \cup B$ and the resulting family is also complete for P.

Throughout this argument, we will often construct a complete collection for a program P in two steps. We will start with some simple complete collection $\mathcal{A}$. Then, we will *close* each element A of $\mathcal{A}$ by expanding it with some additional literals that are consistent with all stable models consistent with A (the validity of this step is ensured by Proposition 5). We will refer to the resulting complete collection as a *closure* of $\mathcal{A}$ ($\mathcal{A}$ may have several closures as we do not require that its elements are "maximally" closed).

Case 1. There is an atom x in P which is not the head of any rule. Clearly, no stable model of P contains x. Thus, the family

$$\mathcal{A}_1 = \{\{\mathbf{not}(x)\}\}.$$

is complete for P. It is also easy to see that

$$c_n^{\mathcal{A}_1} = c_{n-1}^{\mathcal{A}_1},$$

for $n \geq 13$.

Case 2. There is a rule in P of the form $x \leftarrow$. We define

$$\mathcal{A}_2 = \{\{x\}\}.$$

Clearly every stable model of P contains x so $\mathcal{A}_2$ is complete for P. Moreover, we have

$$c_n^{\mathcal{A}_2} = c_{n-1}^{\mathcal{A}_2},$$

for $n \geq 13$.

Case 3. There is a rule in P of the form $x \leftarrow \beta$, where $\beta = y$ or $\beta = \mathbf{not}(y)$, for some atom $y \neq x$. Clearly, the family $\mathcal{A} = \{\{x\}, \{\mathbf{not}(x)\}\}$ is complete for P. We observe that every stable model of P consistent with $\mathbf{not}(x)$ is also consistent with $\mathbf{not}(\beta)$ (as it is a model of the clause $x \leftarrow \beta$). Hence, by Proposition 5, the family

$$\mathcal{A}_3 = \{\{x\}, \{\mathbf{not}(x), \mathbf{not}(\beta)\}\}$$

(the closure of $\mathcal{A}$ with respect to β) is complete for P. Since x and y are different (P contains neither tautologies nor virtual constraints), we have

$$c_n^{\mathcal{A}_3} = c_{n-1}^{\mathcal{A}_3} + c_{n-2}^{\mathcal{A}_3},$$

for $n \geq 13$.

We can assume from now on that every atom is the head of some rule in P, all rules in P have exactly 2 literals in the body, and the atoms occurring in each rule are pairwise different.

Case 4. There is a pair of rules of the form

$$x \leftarrow y, \gamma,$$

$$x \leftarrow \mathbf{not}(y), \delta,$$

where y is an atom, and γ and δ are literals. Let u and v be the atoms occurring in γ and δ, respectively. Since P contains neither tautologies nor virtual constraints, it follows that $\{u, v\} \cap \{x, y\} = \emptyset$ and $x \neq y$. Let us observe that every stable model M of P that is consistent with $\mathbf{not}(x)$ and y is also consistent with $\mathbf{not}(\gamma)$ (it follows from the fact that M is a model of the rule $x \leftarrow y, \gamma$). Similarly, every stable model M consistent with $\mathbf{not}(x)$ and $\mathbf{not}(y)$ is consistent with $\mathbf{not}(\delta)$. Clearly, the family

$$\mathcal{A} = \{\{x\}\} \cup \{\mathbf{not}(x)\}_y = \{\{x\}, \{\mathbf{not}(x), y\}, \{\mathbf{not}(x), \mathbf{not}(y)\}\}$$

is complete for P. Closing $\mathcal{A}$ with respect to literals γ and δ yields the family

$$\mathcal{A}_4 = \{\{x\}, \{\mathbf{not}(x), y, \mathbf{not}(\gamma)\}, \{\mathbf{not}(x), \mathbf{not}(y), \mathbf{not}(\delta)\}\}.$$

By Proposition 5, $\mathcal{A}_4$ is complete for P. Since atoms appearing in literals forming each set in $\mathcal{A}$ are pairwise distinct, we also have

$$c_n^{\mathcal{A}_4} = c_{n-1}^{\mathcal{A}_4} + 2\, c_{n-3}^{\mathcal{A}_4},$$

for $n \geq 13$.

For an atom $x \in At(P)$, let us denote by $\Gamma(x)$ an undirected graph whose vertices are literals occurring in the bodies of rules with the head x. A pair of literals $\{\beta, \gamma\}$ is an edge in $\Gamma(x)$ if there is a rule with the head x and the body containing literals β and γ. We will assume no graph $\Gamma(x)$ contains as vertices a pair of dual literals, as that possibility has been already covered by Case 4.

Next, we consider a number of cases that reflect different possible structures of the graphs $\Gamma(x)$. We use the following graph-theoretic notation. By P_k and C_k we denote a path and a cycle with k vertices, respectively. By the disjoint union of two graphs G and H, denoted $G \dot{\cup} H$, we mean the graph with two disjoint components that are isomorphic to G and H, respectively. We denote by kG the disjoint union of k copies of a graph G.

Case 5. There is an atom x such that $\Gamma(x)$ contains a vertex of degree at least 3. In this case there are three rules with the head x which contain a common literal, say β, in the body. Let γ, δ, and ε be the remaining three literals in the bodies of these three rules. Let M be a stable model of P. If M is consistent with $\mathbf{not}(x)$ and β then, since M satisfies all clauses in P, M is consistent with $\mathbf{not}(\gamma)$, $\mathbf{not}(\delta)$, and $\mathbf{not}(\varepsilon)$.

The family $\mathcal{A} = \{\{x\}, \{\mathbf{not}(x)\}_\beta\}$ is complete[3]. Hence, the family

$$\mathcal{A}_5 = \{\{x\}, \{\mathbf{not}(x), \mathbf{not}(\beta)\}, \{\mathbf{not}(x), \beta, \mathbf{not}(\gamma), \mathbf{not}(\delta), \mathbf{not}(\varepsilon)\}\}$$

(which is the closure of $\mathcal{A}$ with respect to γ, δ and ϵ) is complete for P. Moreover, all literals in each set are pairwise distinct and are not duals of each other. Thus,

$$c_n^{\mathcal{A}_5} = c_{n-1}^{\mathcal{A}_5} + c_{n-2}^{\mathcal{A}_5} + c_{n-5}^{\mathcal{A}_5},$$

for $n \geq 13$.

Case 6. There is an atom x such that $\Gamma(x)$ contains one of the graphs: $P_3 \dot{\cup} 2P_2$, $P_4 \dot{\cup} P_2$, $C_3 \dot{\cup} P_2$. Let F be a subgraph of $\Gamma(x)$ isomorphic to one of the graphs listed above. Let us denote by β_1 any vertex of degree 2 in F and by β_2 any vertex in P_2. The graph $\Gamma(x)$ contains a vertex different from β_1 and β_2, and adjacent to a vertex different from β_1 and β_2. Let us denote one such vertex by β_3.

The literals β_2 and β_3 have two different neighbors in F, say γ_2, γ_3, such that $\gamma_2, \gamma_3 \notin \{\beta_1, \beta_2, \beta_3\}$. By the definition of β_2 and β_3, $\gamma_2 \neq \gamma_3$. It follows that P contains rules $x \leftarrow \beta_i, \gamma_i$, $i = 2, 3$.

We observe that the family

$$\mathcal{A} = \{\{x\}, \{\mathbf{not}(x), \beta_1\}_{\beta_2}, \{\mathbf{not}(x), \mathbf{not}(\beta_1)\}_{\beta_2, \beta_3}\}$$

is complete for P.

We also observe that every stable model M of P consistent with literals $\mathbf{not}(x)$ and β_i (where $i = 2, 3$) is also consistent with $\mathbf{not}(\gamma_i)$ (as M is a model of $x \leftarrow \beta_i, \gamma_i$).

[3] Throughout the paper we will be listing basic parts of $\mathcal{A}$ rather than formally define it as the union of these parts. Here, for instance, we write $\{\{x\}, \{\mathbf{not}(x)\}_\beta\}$ rather than $\{\{x\}\} \cup \{\{\mathbf{not}(x)\}_\beta\}$.

Let us denote by γ_1 and δ_1 the neighbors of β_1 in F. By the definition of β_2, we have $\gamma_1, \delta_1 \notin \{\beta_1, \beta_2\}$ and $\gamma_2 \notin \{\gamma_1, \delta_1\}$.

Let M be a stable model consistent with $\mathbf{not}(x)$ and β_1. Since P contains the rules $x \leftarrow \beta_1, \gamma_1$ and $x \leftarrow \beta_1, \delta_1$, M is consistent with $\mathbf{not}(\gamma_1)$ and $\mathbf{not}(\delta_1)$.

Let $\mathcal{A}_6$ be the closure of $\mathcal{A}$ with respect to literals δ_1 and γ_i, $1, 2, 3$. By Proposition 5, $\mathcal{A}_6$ is complete for P. Moreover, one can easily verify that

$$c_n^{\mathcal{A}_6} = c_{n-1}^{\mathcal{A}_6} + c_{n-4}^{\mathcal{A}_6} + 3c_{n-5}^{\mathcal{A}_6} + 2c_{n-6}^{\mathcal{A}_6},$$

for $n \geq 13$.

Case 7. There is an atom x such that $\Gamma(x)$ contains one of the graphs: $2P_3$, P_5, C_4, C_5.

Let F be a subgraph of $\Gamma(x)$ isomorphic to one of the graphs listed above. We will denote by β_1 and β_2 some two nonadjacent vertices of degree 2 in F.

Clearly, the family

$$\mathcal{A} = \{\{x\}, \{\mathbf{not}(x), \beta_1\}, \{\mathbf{not}(x), \mathbf{not}(\beta_1)\}_{\beta_2}\}$$

is complete for P. Let γ_i and δ_i be the neighbors of β_i, $i = 1, 2$ in F. That is, P contains the rules $x \leftarrow \beta_i, \gamma_i$ and $x \leftarrow \beta_i, \delta_i$, $i = 1, 2$. Thus, if a stable model M of P is consistent with $\mathbf{not}(x)$ and β_i, $i = 1, 2$, it is also consistent with $\mathbf{not}(\gamma_i)$ and $\mathbf{not}(\delta_i)$. Let $\mathcal{A}_7$ be the closure of $\mathcal{A}$ with respect to literals γ_i and δ_i, $i = 1, 2$. By Proposition 5, $\mathcal{A}_7$ is complete for P. Moreover, since $\beta_1, \beta_2 \notin \{\gamma_1, \gamma_2, \delta_1, \delta_2\}$ and $\gamma_i \neq \delta_i$, $i = 1, 2$, it follows that

$$c_n^{\mathcal{A}_7} = c_{n-1}^{\mathcal{A}_7} + c_{n-3}^{\mathcal{A}_7} + c_{n-4}^{\mathcal{A}_7} + c_{n-5}^{\mathcal{A}_7},$$

for $n \geq 13$.

Case 8. There is an atom x such that $\Gamma(x)$ is isomorphic to one of the graphs: $P_3 \dot{\cup} P_2$, P_4, C_3. Let β_1 be a vertex of degree 2 in $\Gamma(x)$ and let β_2 be any vertex which has a neighbor in $\Gamma(x)$ different from β_1. Further, let γ_2 be a neighbor of β_2 in $\Gamma(x)$ different from β_1.

Let M be a stable model consistent with x and $\mathbf{not}(\beta_1)$. Since $x \in M$ and the rule $x \leftarrow \beta_2, \gamma_2$ is the only rule with head x in P that does not contain β_1 in the body, it follows that M is consistent with β_2, and γ_2.

Next, we consider the case of a stable model M consistent with $\mathbf{not}(x)$ and β_1. In this case, since M is a model of the two rules with the head x and β_1 in the body, say $x \leftarrow \beta_1, \gamma_1$ and $x \leftarrow \beta_1, \delta_1$, M is also consistent with $\mathbf{not}(\gamma_1)$, $\mathbf{not}(\delta_1)$.

Finally, if M is a stable model consistent with $\mathbf{not}(x)$ and β_2 then, since M satisfies the rule $x \leftarrow \beta_2, \gamma_2$, it follows that M is also consistent with $\mathbf{not}(\gamma_2)$.

Now, we observe that the family

$$\mathcal{A} = \{\{x\}_{\beta_1}, \{\mathbf{not}(x), \beta_1\}, \{\mathbf{not}(x), \mathbf{not}(\beta_1)\}_{\beta_2}\}$$

is complete for P. By closing its elements by means of observations listed earlier, we get a family $\mathcal{A}_8$, which is also complete for P. Moreover, one can show that

$$c_n^{\mathcal{A}_8} = c_{n-2}^{\mathcal{A}_8} + c_{n-3}^{\mathcal{A}_8} + 3c_{n-4}^{\mathcal{A}_8},$$

for $n \geq 13$.

Case 9. There is an atom x such that $\Gamma(x)$ is isomorphic to the graph $2P_2$. We denote by β_1 and β_2 any two nonadjacent vertices in $\Gamma(x)$. Let γ_i, $i = 1, 2$, be the (unique) neighbor of β_i in $\Gamma(x)$.

Let M be a stable model of P consistent with x and $\mathbf{not}(\beta_1)$. Since $x \leftarrow \beta_2, \gamma_2$ is the only rule in P with head x that does not contain β_1, and since $x \in M$, it follows that M is consistent with β_2 and γ_2.

Similarly, if M is a stable model of P consistent with x and $\mathbf{not}(\beta_2)$, then it is consistent with β_1 and γ_1.

Finally, every stable model consistent with $\mathbf{not}(x)$ and β_i, $i = 1, 2$, is consistent with $\mathbf{not}(\gamma_i)$.

Clearly, the family $\mathcal{A} = \{\{x, \mathbf{not}(\beta_1)\}, \{x, \beta_1\}_{\beta_2}, \{\mathbf{not}(x)\}_{\beta_1,\beta_2}\}$ is complete for P. Let $\mathcal{A}_9$ be the closure of $\mathcal{A}$ by means of the three observations discussed in this case. By Proposition 5, $\mathcal{A}_9$ is complete for P. Moreover, one can verify that

$$c_n^{\mathcal{A}_9} = 2c_{n-3}^{\mathcal{A}_9} + 4c_{n-4}^{\mathcal{A}_9} + c_{n-5}^{\mathcal{A}_9},$$

for $n \geq 13$.

Case 10. There is an atom x such that $\Gamma(x)$ is isomorphic to the graph P_3. Let β be the vertex of degree 2 in $\Gamma(x)$. Every stable model consistent with x is consistent with β too because otherwise every rule with the head x is blocked. Thus, the family $\mathcal{A}_{10} = \{\{\mathbf{not}(x)\}, \{x, \beta\}\}$ is complete for P. Moreover,

$$c_n^{\mathcal{A}_{10}} = c_{n-1}^{\mathcal{A}_{10}} + c_{n-2}^{\mathcal{A}_{10}},$$

for $n \geq 13$.

Case 11. There is an atom x such that $\Gamma(x)$ is isomorphic to the graph P_2. Let $x \leftarrow \beta, \gamma$ be the only rule with the head x. Every stable model consistent with x must be consistent with β and γ, too (because $x \leftarrow \beta, \gamma$ is the only rule that can justify the membership of x in M). Hence, the family $\mathcal{A}_{12} = \{\{\mathbf{not}(x)\}, \{x, \beta, \gamma\}\}$ is complete for P. Moreover,

$$c_n^{\mathcal{A}_{11}} = c_{n-1}^{\mathcal{A}_{11}} + c_{n-3}^{\mathcal{A}_{11}},$$

for $n \geq 13$.

It is a routine task to check that if a graph has no vertices of degree larger than 2, does not contain any of the graphs $P_3 \dot{\cup} 2P_2, P_4 \dot{\cup} P_2, C_3 \dot{\cup} P_2, 2P_3, P_5, C_4, C_5$ and is not isomorphic to any of the graphs $P_3 \dot{\cup} P_2, P_4, C_3, 2P_2, P_3, P_2$ then it is a matching of size at least 3.

Therefore, we will assume from now on that for all atoms x in P, the graphs $\Gamma(x)$ are matchings of size at least 3.

Case 12. For all atoms x in P, the graphs $\Gamma(x)$ are matchings of size at least 3 and P contains a rule with a positive occurrence of an atom in the body. Let us assume that P contains a rule $b \leftarrow x, \delta$, where $x \neq b$ is an atom. Let $b \leftarrow \gamma, \varepsilon$ be another rule in P with the head b. Clearly the atoms appearing in the literals $x, \gamma, \delta, \varepsilon$ are pairwise different (all these literals are vertices of the graph $\Gamma(b)$ and, by Case 4, no two vertices of $\Gamma(b)$ are dual to each other). Let us denote by $\beta_1, \beta_2, \beta_3$ any three literals in $\Gamma(x)$ which are pairwise nonadjacent in $\Gamma(x)$.

If a stable model M is consistent with $\mathbf{not}(b)$ and x then M is consistent with $\mathbf{not}(\delta)$ (since M is a model of the rule $b \leftarrow x, \delta$). For a similar reason every stable model consistent with $\mathbf{not}(b)$, and γ is consistent with $\mathbf{not}(\varepsilon)$.

Let γ_i, $i = 1, 2, 3$, be the neighbor of β_i in $\Gamma(x)$. It is easy to see that every stable model consistent with $\mathbf{not}(x)$ and β_i $(i = 1, 2, 3)$ is consistent with $\mathbf{not}(\gamma_i)$ as M is a model of the rule $a \leftarrow \beta_i, \gamma_i$.

Clearly, the family

$$\mathcal{A} = \{\{x, b\}, \{x, \mathbf{not}(b)\}_{\gamma}, \{\mathbf{not}(x)\}_{\beta_1,\beta_2,\beta_3}\}$$

is complete for P. Let $\mathcal{A}_{12}$ be the result of closing elements of $\mathcal{A}$ by means of the observations listed above. Then $\mathcal{A}_{12}$ is complete for P and one can show that

$$c_n^{\mathcal{A}_{12}} = c_{n-2}^{\mathcal{A}_{12}} + 2c_{n-4}^{\mathcal{A}_{12}} + 4c_{n-5}^{\mathcal{A}_{12}} + 3c_{n-6}^{\mathcal{A}_{12}} + c_{n-7}^{\mathcal{A}_{12}},$$

for $n \geq 13$.

Case 13. For every atom x in P, the graph $\Gamma(x)$ is a matching of size at least 3, P is a purely negative program, and there is a pair of rules in P with exactly 2 common atoms. Let us denote by $\{x, y, u\}$ and $\{x, y, v\}$ the sets of atoms of some two such rules. Since $\Gamma(x)$ is a matching of size at least 3, P contains rules of the form $x \leftarrow \mathbf{not}(b_1), \mathbf{not}(c_1)$, $x \leftarrow \mathbf{not}(b_2), \mathbf{not}(c_2)$, and $x \leftarrow \mathbf{not}(b_3), \mathbf{not}(c_3)$, such that $b_1, b_2, b_3, c_1, c_2, c_3$ are pairwise different atoms.

Subcase 13a. For some i, $1 \leq i \leq 3$, $y, u, v \notin \{b_i, c_i\}$. Without loss of generality, we may assume that $y, u, v \notin \{b_1, c_1\}$. Clearly for some $i = 2, 3$, $y \notin \{b_i, c_i\}$. Again without loss of generality, we assume that $y \notin \{b_2, c_2\}$.

It follows that every stable model consistent with $\mathbf{not}(x)$ and $\mathbf{not}(b_i)$, $i = 1, 2$, is also consistent with c_i. Moreover, every stable model consistent with $\mathbf{not}(x)$ and $\mathbf{not}(y)$ is consistent with u and v. Indeed, $\{x, y, u\}$ and $\{x, y, v\}$ are sets of atoms of two rules in a purely negative program P. Thus, every stable model of P must contain at least one atom from each of the rules. Using these observations, we now close the family

$$\mathcal{A} = \{\{x\}, \{\mathbf{not}(x), y\}_{b_1,b_2}, \{\mathbf{not}(x), \mathbf{not}(y)\}_{b_1}\}$$

(which, clearly, is complete for P). In this way we obtain another family complete for P, say $\mathcal{A}_{13}$. Since $y, u, v \notin \{b_1, c_1\}$ and $y \notin \{b_2, c_2\}$, the literals in each of the sets of $\mathcal{A}_{13}$ are pairwise different and so are their atoms. Thus,

$$c_n^{\mathcal{A}_{13}} = c_{n-1}^{\mathcal{A}_{13}} + c_{n-4}^{\mathcal{A}_{13}} + 3c_{n-5}^{\mathcal{A}_{13}} + 2c_{n-6}^{\mathcal{A}_{13}},$$

for $n \geq 13$.

Subcase 13b. For each rule $x \leftarrow \mathbf{not}(b_i), \mathbf{not}(c_i)$, $i = 1, 2, 3$, we have $\{y, u, v\} \cap \{b_i, c_i\} \neq \emptyset$. Since the sets $\{b_1, c_1\}$, $\{b_2, c_2\}$, and $\{b_3, c_3\}$ are pairwise disjoint, for some $i = 1, 2, 3$ (say, for $i = 1$), $y \in \{b_i, c_i\}$. Let us assume, without loss of generality, that $y = c_1$. Then $b_1 \notin \{u, v\}$ (otherwise, for $i = 2$ or $i = 3$ we would have $\{y, u, v\} \cap \{b_i, c_i\} = \emptyset$, contrary to our assumption for this subcase).

Our program P contains two rules with the sets of atoms $\{x, y, u\}$ and $\{x, y, v\}$, respectively. It also contains the rule $x \leftarrow \mathbf{not}(y), \mathbf{not}(b_1)$ (as $y = c_1$). Since all

these rules are purely negative, every stable model of P contains at least one (unnegated) atom from each of them. Thus, each stable model of P that is consistent with $\mathbf{not}(x)$ and $\mathbf{not}(y)$ is also consistent with u, v, and b_1. We apply this observation to the family $\mathcal{A} = \{\{x\}, \{\mathbf{not}(x)\}_y\}$, which is complete for P. Its closure, $\mathcal{A}_{14}$ is also complete for P. Moreover, since $y \notin \{u, v\}$, $b_1 \notin \{u, v\}$, and $y = c_1 \neq b_1$, literals appearing in all sets of $\mathcal{A}_{14}$ are pairwise distinct and so are their atoms. Thus,

$$c_n^{\mathcal{A}_{14}} = c_{n-1}^{\mathcal{A}_{14}} + c_{n-2}^{\mathcal{A}_{14}} + c_{n-5}^{\mathcal{A}_{14}},$$

for $n \geq 13$.

From now on we assume that there is no pair of rules in P with exactly 2 common atoms. We also reiterate that at this point we are already assuming that P is purely negative and that for every atom x, $\Gamma(x)$ is a matching of size at least 3.

Case 14. There is an atom x in P such that $\Gamma(x)$ is a matching of size at least 4. Let $\{\beta_1, \gamma_1\}$, $\{\beta_2, \gamma_2\}$, $\{\beta_3, \gamma_3\}$, and $\{\beta_4, \gamma_4\}$ be any pairwise different edges in $\Gamma(x)$. Let M be a stable model of P consistent with $\mathbf{not}(x)$ and β_j (where $j = 1, 2, 3, 4$). Then, since M is a model of the rule $x \leftarrow \beta_j, \gamma_j$, M is consistent with $\mathbf{not}(\gamma_j)$. We will make use of this observation several times in the course of the proof to close collections complete for P.

Subcase 14a. There is a rule $b_1 \leftarrow \mathbf{not}(b_2), \mathbf{not}(b_3)$ in P such that the literals $\mathbf{not}(b_1)$, $\mathbf{not}(b_2)$, and $\mathbf{not}(b_3)$ are all vertices of $\Gamma(x)$.

Since no two clauses in P have exactly two common atoms, the literals $\mathbf{not}(b_1)$, $\mathbf{not}(b_2)$, $\mathbf{not}(b_3)$ are nonadjacent in $\Gamma(x)$. Therefore we can assume that $\beta_j = \mathbf{not}(b_j)$, for $j = 1, 2, 3$.

Let is note that the family

$$\mathcal{A}' = \{\{x\}, \{\mathbf{not}(x), \mathbf{not}(\beta_1)\}_{\beta_2,\beta_3,\beta_4}, \{\mathbf{not}(x), \beta_1, \mathbf{not}(\beta_2)\}_{\beta_3,\beta_4}, \\ \{\mathbf{not}(x), \beta_1, \beta_2, \mathbf{not}(\beta_3)\}_{\beta_4}, \{\mathbf{not}(x), \beta_1, \beta_2, \beta_3\}\}$$

is complete for P. Let M be a stable model of P. Since M satisfies the rule $b_1 \leftarrow \mathbf{not}(b_2), \mathbf{not}(b_3)$ and since $\beta_j = \mathbf{not}(b_j)$, for $j = 1, 2, 3$, it follows that M is not consistent with all three literals $\beta_1, \beta_2, \beta_3$. Thus, the family $\mathcal{A} = \mathcal{A}' \setminus \{\{\mathbf{not}(x), \beta_1, \beta_2, \beta_3\}\}$ is also complete for P.

Let $\mathcal{A}_{15}$ be the closure of $\mathcal{A}$ by means of the observation that we made earlier that any stable model consistent with $\mathbf{not}(x)$ and the literal β_i, $i = 1, 2, 3, 4$, is also consistent with the corresponding literal $\mathbf{not}(\gamma_i)$. It is easy to verify that

$$c_n^{\mathcal{A}_{15}} = c_{n-1}^{\mathcal{A}_{15}} + c_{n-5}^{\mathcal{A}_{15}} + 4c_{n-6}^{\mathcal{A}_{15}} + 6c_{n-7}^{\mathcal{A}_{15}} + 3c_{n-8}^{\mathcal{A}_{15}}.$$

Subcase 14b. There is a rule $b_1 \leftarrow \mathbf{not}(b_2), \mathbf{not}(b_3)$ in P such that exactly two of the literals $\mathbf{not}(b_1)$, $\mathbf{not}(b_2)$, and $\mathbf{not}(b_3)$ are vertices of $\Gamma(x)$ and $x \notin \{b_1, b_2, b_3\}$.

Since $x \notin \{b_1, b_2, b_3\}$ and no two clauses in P have exactly two common atoms, the two literals of $\Gamma(x) \cap \{\mathbf{not}(b_1), \mathbf{not}(b_2), \mathbf{not}(b_3)\}$ are nonadjacent in $\Gamma(x)$. Therefore, we can assume without loss of generality that the two literals in $\Gamma(x) \cap \{\mathbf{not}(b_1), \mathbf{not}(b_2), \mathbf{not}(b_3)\}$ are β_1 and β_2. Let us denote the third literal of $\{\mathbf{not}(b_1), \mathbf{not}(b_2), \mathbf{not}(b_3)\}$ by δ.

Since b_1, b_2, b_3 are atoms of a purely negative rule, every stable model M of P must contain at least one b_i, $i = 1, 2, 3$. In other words, M must be consistent with at least one of the literals $\mathbf{not}(\beta_1), \mathbf{not}(\beta_2), \mathbf{not}(\delta)$. Moreover, as in the subcase 14a, stable models consistent with $\mathbf{not}(x)$ and β_i, $i = 1, 2, 3, 4$, must be consistent with the corresponding literal $\mathbf{not}(\gamma_i)$.

The family

$$\mathcal{A} = \{\{x\}, \{\mathbf{not}(x)\}_{\beta_1,\beta_2,\beta_3,\beta_4}\}$$

is complete for P. Let $\mathcal{A}_{16}$ be the closure of $\mathcal{A}$ with respect to these two observations. Clearly, $\mathcal{A}_{16}$ is complete for P and, moreover,

$$c_n^{\mathcal{A}_{16}} = c_{n-1}^{\mathcal{A}_{16}} + c_{n-5}^{\mathcal{A}_{16}} + 4c_{n-6}^{\mathcal{A}_{16}} + 5c_{n-7}^{\mathcal{A}_{16}} + 3c_{n-8}^{\mathcal{A}_{16}} + 2c_{n-9}^{\mathcal{A}_{16}} + c_{n-10}^{\mathcal{A}_{16}},$$

for $n \geq 13$.

Subcase 14c. None of the conditions defining the subcases 14a and 14b holds. Let $\beta_j = \mathbf{not}(b_j)$, for $j = 1, 2, 3, 4$. Since we assume that conditions defining subcases 14a and 14b do not hold, it follows that if $b_1 \leftarrow \beta, \gamma$ is a rule in P then either $\{\beta, \gamma\} = \{\mathbf{not}(x), \gamma\}$ or $\beta, \gamma \notin \Gamma(x) \cup \{x\}$, as there are no rules in P with exactly two common atoms. The graph $\Gamma(b_1)$ is a matching of size at least three. Thus, there are two rules with the head b_1, say $b_1 \leftarrow \beta_5, \gamma_5$ and $b_1 \leftarrow \beta_6, \gamma_6$, whose bodies are disjoint from $\Gamma(x)$ and do not contain the atom x. It follows from our discussion that the literals $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6$ are pairwise different.

Clearly, the family

$$\mathcal{A} = \{\{x\}, \{\mathbf{not}(x), \mathbf{not}(\beta_1)\}_{\beta_2,\beta_3,\beta_4}, \{\mathbf{not}(x), \beta_1\}_{\beta_2,\beta_3,\beta_4,\beta_5,\beta_6}\}$$

is complete for P. We recall that every stable model consistent with $\mathbf{not}(x)$ and β_i, $i = 1, 2, 3, 4$, is consistent with $\mathbf{not}(\gamma_i)$. Similarly, every stable model consistent with $\beta_1 = \mathbf{not}(b_1)$ and β_i, $i = 5, 6$, is consistent with $\mathbf{not}(\gamma_i)$ (because it has to satisfy the rule $b_1 \leftarrow \beta_i, \gamma_i$). Closing $\mathcal{A}$ by means of these two observations yields another complete collection for P, say $\mathcal{A}_{17}$. One can verify that

$$\begin{aligned} c_n^{\mathcal{A}_{17}} \quad = \quad & c_{n-1}^{\mathcal{A}_{17}} + c_{n-5}^{\mathcal{A}_{17}} + 3c_{n-6}^{\mathcal{A}_{17}} + 3c_{n-7}^{\mathcal{A}_{17}} + 2c_{n-8}^{\mathcal{A}_{17}} + 5c_{n-9}^{\mathcal{A}_{17}} + 10c_{n-10}^{\mathcal{A}_{17}} \\ & + 10c_{n-11}^{\mathcal{A}_{17}} + 5c_{n-12}^{\mathcal{A}_{17}} + c_{n-13}^{\mathcal{A}_{17}}, \end{aligned}$$

for $n \geq 13$.

Case 15. For every atom x in P (which is, we recall, purely negative), the graph $\Gamma(x)$ is a matching of size exactly three. Let $x \leftarrow \beta_1, \gamma_1$, $x \leftarrow \beta_2, \gamma_2$, and $x \leftarrow \beta_3, \gamma_3$ be the rules with the head x. Let $\beta_i = \mathbf{not}(b_i)$, $i = 1, 2, 3$, for some atoms $b_1, b_2, b_3 \in At(P)$. Since Case 13 is excluded and $x \leftarrow \beta_i, \gamma_i$ is in P, a rule in P with the head b_i contains either both $\mathbf{not}(x)$ and γ_i in its body or none of them. Thus, for each $i = 1, 2, 3$, at least two rules with the head b_i have bodies containing neither $\mathbf{not}(x)$ nor γ_i. Let, for $i = 1, 2, 3$, $b_i \leftarrow \delta_i, \varepsilon_i$ and $b_i \leftarrow \varphi_i, \psi_i$ be these two rules.

Every stable model M of P that is consistent with x is consistent with at least one of the literals $\beta_1, \beta_2, \beta_3$ because at least one of the rules $x \leftarrow \beta_1, \gamma_1$, $x \leftarrow \beta_2, \gamma_2$, and $x \leftarrow \beta_3, \gamma_3$ has its body satisfied in M (provides a justification for x). Therefore, the

family $\mathcal{A} = \{\{\mathbf{not}(x)\}_{\beta_1,\beta_2,\beta_3}, \{x,\beta_1\}_{\delta_1,\varphi_1}, \{x,\beta_2\}_{\delta_2,\varphi_2}, \{x,\beta_3\}_{\delta_3,\varphi_3}\}$ is complete for P.

Every stable model consistent with $\mathbf{not}(x)$ and β_i, $i = 1,2,3$, is consistent with $\mathbf{not}(\gamma_i)$ (as it satisfies the rule $x \leftarrow \beta_i, \gamma_i$). Similarly, every stable model consistent with $\beta_i = \mathbf{not}(b_i)$ and δ_i (respectively, φ_i), $i = 1,2,3$, is consistent with $\mathbf{not}(\varepsilon_i)$ (respectively, $\mathbf{not}(\psi_i)$). These observations allow us to close the collection $\mathcal{A}$ and obtain another collection complete for P, say $\mathcal{A}_{18}$. Moreover,

$$c_n^{\mathcal{A}_{18}} = c_{n-4}^{\mathcal{A}_{18}} + 6c_{n-5}^{\mathcal{A}_{18}} + 9c_{n-6}^{\mathcal{A}_{18}} + 4c_{n-7}^{\mathcal{A}_{18}},$$

for $n \geq 13$.

Cases 1-15 exhaust all possibilities for normal 3-programs without tautologies nor virtual constraints. Moreover, one can verify that for each i, $1 = 1,\ldots,18$, the maximum root of the characteristic equation of the recurrence defining $c_n^{\mathcal{A}_i}$ is less than or equal to 1.70711. Thus, Theorems 10 and 11 follow.

The lower bound in this case follows from general observations made in Section 4. Namely, Corollary 2 implies the following result.

Corollary 6
Let $0 < \alpha < \mu_3 \approx 1.58489$. There is no algorithm for computing all stable models of 3-programs with worst-case performance bounded by $O(f(m)\alpha^n)$, where f is a polynomial and m is the size of the input program.

7 Performance of *smodels* on 2-programs

In this section, we briefly discuss applicability of the approach proposed in Section 3 to the analysis of the performance of the algorithm of *smodels* (Niemelä and Simons 2000). To this end, we first describe a version of the procedure *complete* that yields an algorithm for computing stable models whose performance estimate provides an upper bound on the running time of *smodels*. In the description, we will refer to the procedure *expand*(P, A) that is used by *smodels*. Given a logic program P and a set of literals $A \subseteq Lit(P)$, *expand*(P, A) returns a set of literals B such that $A \subseteq B$ and every stable model of P consistent with A is also consistent with B. Informally, *expand*(P, A) serves as a unit propagation engine in *smodels* (it propagates the literals in A to establish additional literals implied by P and A).

The specific implementation of the procedure *expand* that is used by *smodels* generalizes ideas underlying the notion of the well-founded semantics. We will not discuss this procedure in detail. We will only mention some of its propagation rules that are of importance in our arguments here. Namely, given a normal logic program P and a set of literals A, the set of literals $B = expand(P, A)$ is closed under the following rules:

1. If p is a fact in P or $p \in A$, $p \in B$
2. If p is not in the head of any clause in P, $\mathbf{not}(p) \in B$
3. If $p \leftarrow \alpha_1,\ldots,\alpha_k$ is a rule in P ($p \in At(P)$, $\alpha_i \in Lit(P)$) and $\alpha_i \in B$, $1 \leq i \leq k$, then $p \in B$

4. If $p \leftarrow \alpha_1, \ldots, \alpha_k$ is a rule in P ($p \in At(P)$, $\alpha_i \in Lit(P)$), $\mathbf{not}(p) \in B$ and $\alpha_j \in B$, for $j = 1, \ldots, i-1, i+1, \ldots, k$, then $\mathbf{not}(\alpha_i) \in B$
5. If $p \leftarrow \alpha_1, \ldots, \alpha_k$ is a rule in P ($p \in At(P)$, $\alpha_i \in Lit(P)$), no other rule in P has head p, and if $\mathbf{not}(\alpha_i) \in B$, for some i, $1 \leq i \leq k$, then $\mathbf{not}(p) \in B$.

Let P be a non-empty logic program and let $\alpha \in Lit(P)$. We define $E(\alpha) = expand(P, \{\alpha\})$. Using the properties listed above, it is easy to show that every stable model of P that is consistent with α is also consistent with $E(\alpha)$. In particular, it follows that for every atom $x \in At(P)$, $\{E(x), E(\mathbf{not}(x))\}$ is a complete collection for P.

We will now define a procedure that, for a given non-empty normal program P returns a collection of sets that is complete for P. The procedure looks for the first case whose assumption holds, returns the corresponding collection of sets and terminates. The four cases that are considered are:

1. If there is $x \in At(P)$ such that both $x \in E(\mathbf{not}(x))$ and $\mathbf{not}(x) \in E(x)$, then return $\mathcal{A} = \{At(P)\}$
2. If there is $x \in At(P)$ such that $x \in E(\mathbf{not}(x))$, then return $\mathcal{A} = \{E(x)\}$
3. If there is $x \in At(P)$ such that $\mathbf{not}(x) \in E(x)$, then return $\mathcal{A} = \{E(\mathbf{not}(x))\}$
4. Otherwise, return this collection $\mathcal{A} = \{E(x), E(\mathbf{not}(x))\}$ for which

   $$\min\{|E(x)|, |E(\mathbf{not}(x))|\}$$

   is maximized. If there are more than one with the same maximum value, choose this one among them that maximizes $|E(x)|+|E(\mathbf{not}(x))|$.

It follows from our earlier discussion that, given a non-empty logic program P, this procedure returns a complete collection $\mathcal{A}$ for P. We denote this procedure by $complete^{sm}$. The procedure $complete^{sm}$ determines a specific instantiation of the algorithm *stable* for computing all stable models of P. We will denote this algorithm by $stable^{sm}$. In the case when we restrict to 2-programs, we can analyze the worst-case performance of this algorithm using the techniques developed and used earlier in the paper. Namely, we have the following two results. The proofs closely resemble those presented earlier. Therefore, we provide only their outlines.

Theorem 12
The algorithm $stable^{sm}$ when used on 2-programs runs in time $O(m \times 1.46558^n)$, where n is the number of atoms in P and m is the size of P.

Proof
(Sketch) We follow the same case analysis as in the proof of Theorem 5. It is easy to see that in Cases 1 and 3, we have $x \in E(\mathbf{not}(x))$, while in Case 2 we have $\mathbf{not}(x) \in E(x)$. Thus, in each of these cases, the procedure $complete^{sm}$ returns a complete collection with the signature bounded by (1).

So, let us assume that Case 4 of the proof of Theorem 5 holds. It is easy to see using the properties of the procedure *expand*, given above, that $\{x, y\} \subseteq E(x)$ and $\{\mathbf{not}(x), \mathbf{not}(y)\} \subseteq E(\mathbf{not}(x))$. Thus, the signature of the collection $complete^{sm}$ is

bounded by (1) or (2,2). The same claim holds for the complete collection returned in Case 5.

In Case 6 we assume that none of the Cases 1-5 holds. Under the notation introduced in the analysis of Case 6 in the proof of Theorem 5,

$$\{w, y_1, \ldots, y_q\} \subseteq E(w)$$

and

$$\{\mathbf{not}(w), x_1, \ldots, x_p, \mathbf{not}(z_1), \ldots, \mathbf{not}(z_r)\} \subseteq E(\mathbf{not}(w)).$$

Since $p+q+r \geq 2$ and $p+r \geq 1$, it follows that the collection $\{E(w), E(\mathbf{not}(w))\}$ has the signature bounded by (1,3) or (2,2).

Thus, in each of the cases, the complete collection returned by the procedure $complete^{sm}$ has the signature bounded by (1), (2,2) or (1,3). The corresponding recurrence relations are

$$c_n^1 = c_{n-1}^1, \quad c_n^2 = 2c_{n-2}^2, \quad \textit{and} \quad c_n^3 = c_{n-1}^3 + c_{n-3}^3.$$

The characteristic root of the third recurrence relation is (approximately) 1.46558 and is larger then the characteristic roots of the two remaining relations. Thus, the assertion of Theorem 12 follows by the same argument as in other cases. □

Theorem 13
The algorithm $stable^{sm}$ when used on purely-negative 2-programs without dual clauses runs in time $O(m \times 1.32472^n)$, where n is the number of atoms and m is the size of an input program.

Proof
(Sketch) We consider several cases that cover the class of all purely negative 2-programs without dual clauses. We assume in each case that all earlier ones are excluded.

Case 1. Program P contains an atom that is not the head of any clause in P. In this case, the procedure $complete^{sm}$ returns a complete collection consisting of one non-empty set. Thus, its signature is bounded by (1).

Case 2. There is an atom x in P that is the head of exactly one clause, say $x \leftarrow \mathbf{not}(y)$. Since P has no dual clauses and since y is the head of at least one clause, there is an atom z such that $z \neq x$ and $y \leftarrow \mathbf{not}(z)$ is a clause of P. Let us observe that $|E(y)| \geq 2$. Indeed, $y \in E(y)$ (by rule 1) and, by rule 5, $\mathbf{not}(x) \in E(y)$, as well. Furthermore, y has at least two neighbors, x, and another one, say z. By rules 1, 3 and 4, $\mathbf{not}(y), x, z \in E(\mathbf{not}(y))$. Thus, the procedure $complete^{sm}$ returns either a complete collection with the signature bounded by (1) or, if it gets to case 4, a two-element complete collection with the signature bounded by $(2, 3)$.

From now on, we will assume that every atom of P is the head of at least two clauses. In particular, for every atom x of P, $|E(\mathbf{not}(x))| \geq 3$. Moreover, it follows also that there is an atom with at least four neighbors (otherwise, for every atom in P the number of its occurrences in the bodies of rules would be smaller than the number of its occurrences as the head, a contradiction).

Case 3. There is an atom x in P such that $|E(x)| \geq 2$. Since x has at least two neighbors, the signature of $\{E(x), E(\mathbf{not}(x))\}$ is bounded by $(2,3)$. It follows that the procedure *complete*^{sm} returns a complete collection with the signature bounded by (1) or $(2,3)$.
Case 4. For every atom x, $|E(x)| = 1$. Let y be an atom in P such that y has at least four neighbors (we noted that such an atom exists). It follows by rules 1, 3 and 4 that $|E(\mathbf{not}(y))| \geq 5$. Thus, the procedure *complete*^{sm} returns a complete collection with the signature bounded by (1) or $(1,5)$.

Thus, in each of the cases, the complete collection returned by the procedure *complete*^{sm} has the signature bounded by (1), (2,3) or (1,5). The corresponding recurrence relations are

$$c_n^1 = c_{n-1}^1, \quad c_n^2 = c_{n-2}^2 + c_{n-3}, \quad \textit{and} \quad c_n^3 = c_{n-1}^3 + c_{n-5}^3.$$

The characteristic roots of the second and third recurrence relations are the same and are (approximately) equal 1.46558. This quantity is greater than the characteristic root of the first relation (which equals 1). Thus, we obtain the assertion. □

The algorithm *stable*^{sm} follows closely the way in which *smodels* works. There are however, some differences. If, during the search, the case (1) of the procedure *complete*^{sm} applies, there are no stable models on this search path. *Smodels* recognizes that and backtracks immediately. Our algorithm backtracks from the next recursive call to *stable*^{sm}. This difference is minor and has no effect on the asymptotic performance analysis.

Second, *smodels*, when looking for the first case that applies during the execution of the procedure *complete*^{sm}, checks whether $E(\alpha)$ is consistent not only with α but also with all literals whose values where set earlier in the search. In this way, *smodels* increases a possibility that the procedure *complete*^{sm} will not enter case (4), the only case when the search splits into two branches. It follows that the worst-case performance estimate of *smodels* is bounded from above by the worst-case performance of the algorithm *stable*^{sm} described here. Consequently, the performance bounds of Theorems 12 and 13 apply to *smodels*, as well.

The techniques we developed in this paper seem to have only limited applicability in the analysis of the performance of *smodels*. First, it is not clear whether it can be applied to obtain non-trivial performance bounds for *smodels* on programs with longer clauses, for instance, for 3-programs. Second, the bounds they imply in the case of 2-programs are weaker than those we derived in Section 5. We stressed in several places that *smodels* uses stronger propagation techniques than those needed by our analysis. It is possible that the performance of bounds we obtained can be improved. However, there is most likely no simple way to do so. The algorithms and performance bounds we derived in Sections 5 and 6 strongly depend on the ability of the search to split in each branch point into more than two search paths, a property that *smodels* does not have.

8 The general case

In this section we present an algorithm that computes all stable models of arbitrary propositional logic programs. It runs in time $O(m2^n/\sqrt{n})$ and so, provides an improvement over the trivial bound $O(m2^n)$. However, our approach is quite different from that used in the preceding sections. The key component of the algorithm is an auxiliary procedure *stable_aux*(P, π). Let P be a logic program and let $At(P) = \{x_1, x_2, \ldots, x_n\}$. Given P and a permutation π of $\{1, 2, \ldots, n\}$, the procedure *stable_aux*(P, π) looks for an index j, $1 \leq j \leq n$, such that the set $\{x_{\pi(j)}, \ldots, x_{\pi(n)}\}$ is a stable model of P. Since no stable model of P is a proper subset of another stable model of P, for any permutation π there is at most one such index j. If such j exists, the procedure outputs the set $\{x_{\pi(j)}, \ldots, x_{\pi(n)}\}$.

In the description of the algorithm *stable_aux*, we use the following notation. For every atom a, by $pos(a)$ we denote the list of all clauses which contain a (as a non-negated atom) in their bodies, and by $neg(a)$ a list of all clauses that contain **not**(a) in their bodies. Given a standard linked-list representation of logic programs, all these lists can be computed in time linear in m.

Further, for each clause C, we introduce counters $p(C)$ and $n(C)$. We initialize $p(C)$ to be the number of positive literals (atoms) in the body of C. Similarly, we initialize $n(C)$ to be the number of negative literals in the body of C. These counters are used to decide whether a clause belongs to the reduct of the program and whether it "fires" when computing the least model of the reduct.

stable_aux(P, π)
(1) $M = At(P)$;
(2) $Q :=$ set of clauses C such that $p(C) = n(C) = 0$;
(3) $lm := \emptyset$;
(4) **for** $j = 1$ **to** n **do**
(5) **while** $Q \neq \emptyset$ **do**
(6) $C_0 :=$ any clause in Q;
(7) mark C_0 as used and remove it from Q;
(8) **if** $h(C_0) \notin lm$ **then**
(9) $lm := lm \cup \{h(C_0)\}$;
(10) **for** $C \in pos(h(C_0))$ **do**
(11) $p(C) := p(C)-1$;
(12) **if** $p(C) = 0$ & $n(C) = 0$ & C not used **then** add C to Q;
(13) **if** $lm = M$ **then** output M and stop;
(14) $M := M \setminus \{x_{\pi(j)}\}$;
(15) **for** $C \in neg(x_{\pi(j)})$ **do**
(16) $n(C) := n(C)-1$;
(17) **if** $n(C) = 0$ & $p(C) = 0$ & C not used **then** add C to Q.

Let us define $M_j = \{x_{\pi(j)}, \ldots, x_{\pi(n)}\}$. Intuitively, the algorithm *stable_aux* works as follows. In the iteration j of the **for** loop (4) it computes the least model of the reduct P^{M_j} (lines (5)-(12)). Then it tests whether $M_j = lm(P^{M_j})$ (line (13)). If so, it outputs M_j (it is a stable model of P) and terminates. Otherwise, it computes the reduct $P^{M_{j+1}}$. In fact, the reduct is not explicitly computed. Rather, the number of

negated literals in the body of each rule is updated to reflect the fact that we shift attention from the set M_j to the set M_{j+1} and one more negated literal is satisfied with respect to M_{j+1} (lines (14)-(17)). The key to the algorithm is the fact that it computes reducts P^{M_j} and least models $lm(P^{M_j})$ in an incremental way and, so, tests n candidates M_j for stability in time $O(m)$ (where m is the size of the program). We make these comments and claims more precise in the statement and proof of the next result.

Proposition 6
Let P be a logic program and let $At(P) = \{x_1, \ldots, x_n\}$. For every permutation π of $\{1, \ldots, n\}$, if $M = \{x_{\pi(j)}, \ldots, x_{\pi(n)}\}$ then the procedure *stable_aux*(P, π) outputs M if and only if M is a stable model of P. Moreover, the procedure *stable_aux* runs in $O(m)$ steps, where m is the size of P.

Proof
We first observe that during the execution of the algorithm, each time when the while loop in the lines (5)-(12) terminates we have the following property:

(I1) For every clause C, $p(C)$ is the number of atoms in $b^+(C)$ that are not in lm.

Indeed, $p(C)$ is initialized so that (I1) holds at the start of the algorithm *stable*. Then, each time a new atom, say a, is added to lm in line (9), the counter $p(C)$ is decreased by 1, in line (11), for every clause C such that $a \in b^+(C)$.

It is also easy to see that for every iteration j, $1 \leq j \leq n$, at any time during the execution of the **while** loop the following property holds:

(I2) For every clause C, $n(C) = 0$ if and only if $b^-(C) \cap M_j = \emptyset$.

Let us denote by lm_j the value of lm when the j^{th} iteration of the **for** loop terminates. To complete the proof of the correctness of the algorithm *stable*, we will now argue that for every iteration j, $1 \leq j \leq n$, we have the following two properties:

(I3) After every iteration of the **while** loop, lm is a subset of $lm(P^{M_j})$, and
(I4) $lm_j = lm(P^{M_j})$.

We prove (I3) and (I4) simultaneously by double induction with respect to j and, for each j, with respect to the number of iterations of the **while** loop within the j^{th} iteration of the **for** loop. We will provide the reasoning for the inductive step for the outermost induction (the one with respect to j). The argument to establish the basis of this induction is similar and we omit it.

Thus, let us consider the iteration j+1, where $j \geq 1$, and let us assume that (I3) and (I4) hold just before it starts. We will prove that (I3) and (I4) hold after the iteration j+1 is completed. To this end, we proceed by induction on the number of iterations of the **while** loop. We start by establishing the basis of this induction. By the induction hypothesis (for the outermost induction), $lm_j = lm(P^{M_j})$. Since $M_{j+1} \subseteq M_j$, $lm(P^{M_j}) \subseteq lm(P^{M_{j+1}})$. Hence, at the start of the **while** loop during the $(j+1)^{st}$ iteration of the **for** loop, the set lm (which is the same at that time as lm_j) is contained in $lm(P^{M_{j+1}})$. Thus, the basis of of the inner induction holds.

For the inductive step, let us consider a particular iteration of the **while** loop. In this iteration, we consider a clause C_0 such that $p(C_0) = 0$ and $n(C_0) = 0$. By (I1) it follows that $b^+(C_0) \subseteq lm$ (as computed up to that point, that is, at the end of the previous iteration of the **while** loop). By the induction hypothesis, $lm \subseteq lm(P^{M_{j+1}})$. Further, by (I2), the *reduced* rule C_0 (that is, the rule obtained from C_0 by removing all its negated literals) belongs to $P^{M_{j+1}}$. Thus, it follows that $h(C_0)$ belongs to $lm(P^{M_{j+1}})$. When line (9) of this iteration of the **while** loop is completed, lm is either as it was during the previous iteration or it is expanded by the addition of $h(C_0)$. In either case, the set lm at the end of the present iteration satisfies $lm \subseteq lm(P^{M_{j+1}})$.

In particular, it follows that $lm_{j+1} \subseteq lm(P^{M_{j+1}})$ (as lm_{j+1} is the set lm after the last iteration of the **while** loop in the $(j+1)^{st}$ iteration of the **for** loop). We will show that lm_{j+1} is a model of $P^{M_{j+1}}$. Let $C \in P^{M_{j+1}}$ and let C' be a clause in P such that C is the result of removing all negated literals from C'. By (I2) it follows that $n(C') = 0$. If $p(C') = 0$ when the **while** loop (within the $(j+1)^{st}$ iteration of the **for** loop) ends, C' was placed on the Q during the execution of this **while** loop and then removed at some point later during the execution of this loop. Thus, $h(C') = h(C) \in lm_{j+1}$. On the other hand, if $p(C') > 0$ then, by (I1), there is an atom a in the body of C such that $a \notin lm_{j+1}$. In either case, lm_{j+1} satisfies C. Since lm_{j+1} is a model of $P^{M_{j+1}}$, $lm(P^{M_{j+1}}) \subseteq lm_{j+1}$ and, consequently, $lm_{j+1} = lm(P^{M_{j+1}})$.

It is clear that the first part of the assertion follows directly from (I4). To complete the proof of the proposition, we note that each clause of the program P is processed by the **while** loop (5) at most once. Thus, the time needed for steps (6)-(9), over all iterations of the **for** loop (4), is $O(|P|) = O(m)$. Moreover, the total time needed for all iterations of **for** loop (10), over all iterations of the **for** loop (4), is bounded by the total number of positive occurrences of atoms in the program (there is at most one iteration for each such occurrence), that is, it is $O(m)$. Similarly, the total time needed for steps (13)-(17) over all iterations of the **for** loop (4) is $O(m)$. □

We will now describe how to use the procedure *stable_aux* in an algorithm to compute stable models of a logic program. A collection $\mathcal{S}$ of permutations of $\{1, 2, \ldots, n\}$ is *full* if every subset S of $\{1, 2, \ldots, n\}$ is a final segment (suffix) of a permutation in $\mathcal{S}$ or, more precisely, if for every subset S of $\{1, 2, \ldots, n\}$ there is a permutation $\pi \in \mathcal{S}$ such that $S = \{\pi(n-|S|+1), \ldots, \pi(n)\}$.

If S_1 and S_2 are of the same cardinality then they cannot occur as suffixes of the same permutation. Since there are $\binom{n}{\lfloor n/2 \rfloor}$ subsets of $\{1, 2, \ldots, n\}$ of cardinality $\lfloor n/2 \rfloor$, every full family of permutations must contain at least $\binom{n}{\lfloor n/2 \rfloor}$ elements. An important property is that for every $n \geq 0$ there is a full family of permutations of cardinality $\binom{n}{\lfloor n/2 \rfloor}$. An algorithm to compute such a minimal full set of permutations, say $\mathcal{S}_{min}$, is described in (Knuth 1998) (Vol. 3, pages 579 and 743-744). We refer to this algorithm as *perm*(n). The algorithm *perm*(n) enumerates all permutations in $\mathcal{S}_{min}$ by generating each next permutation entirely on the basis of the previous one. The algorithm *perm*(n) takes $O(n)$ steps to generate a permutation and each permutation is generated only once.

We modify the algorithm $perm(n)$ to obtain an algorithm to compute all stable models of a logic program P. Namely, each time a new permutation, say π, is generated, we make a call to $stable\_aux(P, \pi)$. We call this algorithm $stable^p$. Since $\binom{n}{\lfloor n/2 \rfloor} = \Theta(2^n/\sqrt{n})$ we have the following result.

Proposition 7
The algorithm $stable^p$ is correct and runs in time $O(m2^n/\sqrt{n})$.

Proof
Let $M = \{x_{i_1}, \ldots, x_{i_k}\}$, where $k = |M|$, be a stable model of P. Since $\mathcal{S}_{min}$ is complete, there is a permutation π in $\mathcal{S}_{min}$ such that $\{\pi(n-k+1), \ldots, \pi(n)\} = \{i_1, \ldots, i_k\}$ or, equivalently, $\{x_{\pi(n-k+1)}, \ldots, x_{\pi(n)}\} = \{x_{i_1}, \ldots, x_{i_k}\} = M$. Since $\pi \in \mathcal{S}_{min}$, π will be generated during the execution of $perm\_stable(P)$. Thus, by Proposition 6, the call to $stable(P, \pi)$, made right after π is generated, outputs M as a stable model of P. Conversely, every set M output by *perm_stable* is, clearly, a stable model of P. Thus, sets output by *perm_stable* are precisely the stable models of P (we note, however, that some sets M may be suffixes of several permutations and, consequently, will be output more than once).

As for the running time, the procedure $stable(P, \pi)$ is called $\binom{n}{\lfloor n/2 \rfloor}$ times during the execution of $perm\_stable(P)$. Since $\binom{n}{\lfloor n/2 \rfloor} = O(2^n/\sqrt{n})$, the total time needed for all these calls is $O(m2^n/\sqrt{n})$. The running time of the algorithm $perm(n)$ is $O(n2^n/\sqrt{n})$. Thus, the algorithm $perm\_stable(P)$ runs in time $O(m2^n/\sqrt{n})$. □

Since the program $P(n, \lfloor n/2 \rfloor)$ has exactly $\binom{n}{\lfloor n/2 \rfloor}$ stable models and each of these models has $\Theta(n)$ elements, every algorithm to compute all stable models of a logic program must take at least $\Omega(\sqrt{n}2^n)$ steps. Whether in this case, there is $\alpha < 2$, a polynomial f and an algorithm computing all stable models of an arbitrary logic program P in time $O(f(m)\alpha^n)$ is an open problem. Since the size of the program $P(n, \lfloor n/2 \rfloor)$ is not $O(n)$, the methods used in the proof of Corollary 2 do not work here.

9 Discussion and conclusions

We presented algorithms for computing stable models of logic programs with worst-case performance bounds asymptotically better than the trivial bound of $O(m2^n)$. These are the first results of that type in the literature. In the general case, we proposed an algorithm that runs in time $O(m2^n/\sqrt{n})$ improving the performance over the brute-force approach by the factor of $\sqrt{n}$. Most of our work, however, was concerned with algorithms for computing stable models of t-programs. We proposed an algorithm that computes stable models of t-programs in time $O(m\alpha_t^n)$, where $\alpha_t < 2 - 1/2^t$. We strengthened these results in the case of 2- and 3-programs. In the first case, we presented an algorithm that runs in time $O(m3^{n/3})$ ($\approx O(m \times 1.44225^n)$). For the case of 3-programs, we presented an algorithm running in the worst case in time $O(m \times 1.70711^n)$.

In addition to these contributions, our work leads to several interesting questions. A foremost among them is whether our results can be further improved. First, we

observe that in the case when the task is to compute *all* stable models, we already have proved optimality (up to a polynomial factor) of the algorithms developed for the class of all programs and the class of all 2-programs. However, in all other cases there is still room for improvement — our lower and upper bounds do not coincide.

The situation gets even more interesting when we want to compute one stable model (if stable models exist) rather than all of them. The algorithms we presented here can, of course, be adapted to this case (by terminating them as soon as the first model is found). Thus, the upper bounds derived in this paper remain valid. But the lower bounds, which we derive on the basis of the number of stable models input programs may have, do not. In particular, it is no longer clear whether the algorithm we developed for the case of 2-programs remains optimal. One cannot exclude existence of pruning techniques that, in the case when the input program has stable models, would on occasion eliminate from considerations parts of the search space possibly containing some stable models, recognizing that the remaining portion of the search space still contains some.

Such search space pruning techniques are possible in the case of satisfiability testing. For instance, the pure literal rule, sometimes used by implementations of the Davis-Putnam procedure, eliminates from considerations parts of search space that may contain stable models (Monien and Speckenmeyer 1985; Kullmann 1999). However, the part that remains is guaranteed to contain a model as long as the input theory has one. No examples of analogous search space pruning methods are known in the case of stable model computation. We feel that nonmonotonicity of the stable model semantics is the reason for that but a formal account of this issue remains an open problem.

Finally, we note that we obtained, to the best of our knowledge, first non-trivial worst-case performance bounds for *smodels*. While our bounds apply only to the case when input programs are restricted to be 2-programs and our techniques do not seem to be best suited for the analysis of *smodels*, the results we presented demonstrate that the worst-case analysis of algorithms such as *smodels* may be possible.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 9874764 and 0097278. The authors also wish to thank anonymous referees for their comments and suggestions.

References

Apt, K. 1990. Logic programming. In *Handbook of theoretical computer science*, J. van Leeuven, Ed. Elsevier, Amsterdam, 493–574.

Bonatti, P. A. 2001. Reasoning with infinite stable models. In *Proceedings of IJCAI-01*. Morgan Kaufmann, 603–608.

Bonatti, P. A. 2002. Reasoning with infinite stable models ii: disjunctive programs. In *Logic programming, Proceedings of the 2002 International Conference on Logic Programming*. Lecture Notes in Computer Science, vol. 2401. Springer-Verlag, 333–346.

Bonatti, P. A. and Eiter, T. 1996. Querying Disjunctive Databases Through Nonmonotonic Logics. *Theoretical Computer Science 160*, 321–363.

Bonatti, P. A. and Olivetti, N. 2002. Sequent Calculi for Propositional Nonmonotonic Logics. *ACM Transactions on Computational Logic 2*, 352–304.

Cholewiński, P. and Truszczyński, M. 1999. Extremal problems in logic programming and stable model computation. *Journal of Logic Programming 38*, 219–242.

Dix, J. 1995. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae 22,* 3, 257 – 288.

Eiter, T., Faber, W., Leone, N., and Pfeifer, G. 2000. Declarative problem-solving in DLV. In *Logic-Based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publishers, Dordrecht, 79–103.

Gelfond, M. and Lifschitz, V. 1988. The stable semantics for logic programs. In *Proceedings of the 5th International Conference on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

Knuth, D. E. 1998. *The Art of Computer Programming*. Vol. 3. Addison Wesley. Second edition.

Kullmann, O. 1999. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science 223*, 1–72.

Lonc, Z. and Truszczyński, M. 2001. On the problem of computing the well-founded semantics. *Theory and Practice of Logic Programming 1*, 591–609.

Lonc, Z. and Truszczyński, M. 2003. Fixed-parameter complexity of semantics for logic programs. *ACM Transactions on Computational Logic*, to appear.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, K. Apt, W. Marek, M. Truszczyński, and D. Warren, Eds. Springer Verlag, 375–398.

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

Marek, W. and Truszczyński, M. 1993. *Nonmonotonic Logic; Context-Dependent Reasoning*. Springer-Verlag, Berlin.

Monien, B. and Speckenmeyer, E. 1985. Solving satisfiability in less than 2^n steps. *Discrete Applied Mathematics 10*, 287–295.

Moon, J. and Moser, L. 1965. On cliques in graphs. *Israel J. Math 3*, 23–28.

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

Niemelä, I. and Simons, P. 2000. Extending the smodels system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*, J. Minker, Ed. Kluwer Academic Publishers, 491–521.

Subrahmanian, V., Nau, D., and Vago, C. 1995. WFS + branch bound = stable models. *IEEE Transactions on Knowledge and Data Engineering 7*, 362–377.

Syrjänen, T. 1999. *lparse*, a procedure for grounding domain restricted logic programs. `http://www.tcs.hut.fi/Software/smodels/lparse/`.

Truszczyński, M. 2002. Computing large and small stable models. *Theory and Practice of Logic Programming 2*, 1–23.

Appendix: linear recurrence relations

In the paper we use some basic properties of linear recurrence relations of the form:

$$a_n = \begin{cases} k_a & \text{if } 0 \leq n < p \\ \alpha_1 a_{n-1} + \ldots + \alpha_p a_{n-p} & \text{otherwise,} \end{cases} \quad (1)$$

where for every i, $1 \leq i \leq p$, $\alpha_i \geq 0$.

We denote by r_a the maximum root of the *characteristic* equation of the recurrence defining

$$x^p - \alpha_1 x^{p-1} - \ldots - \alpha_{p-1} x - \alpha_p = 0.$$

If $r_a \geq 1$ then it is easy to prove by induction that for every $n \geq 0$,

$$a_n \leq k_a r_a^n.$$

In addition, for every $r > r_a$,

$$\alpha_1 r^{p-1} + \ldots + \alpha_{p-1} r + \alpha_p < r^p. \tag{2}$$

Indeed, the inequality holds for every sufficiently large r. If it fails for some $r > r_a$, then there is a root of the characteristic equation in the interval $[r, \infty)$, a contradiction.

Let us consider another relation of the form (1):

$$b_n = \begin{cases} k_b & \text{if } 0 \leq n < q \\ \beta_1 b_{n-1} + \ldots + \beta_q b_{n-q} & \text{otherwise.} \end{cases}$$

Let us assume that $r_b \geq r_a \geq 1$, where r_a and r_b are the maximum roots of the characteristic equations of the respective recurrence relations. Let us define

$$c_n = \begin{cases} k_c & \text{if } 0 \leq n < \max\{p, q\} \\ \max\{\alpha_1 c_{n-1} + \ldots + \alpha_p c_{n-p}, & \\ \qquad \beta_1 c_{n-1} + \ldots + \beta_q c_{n-q}\} & \text{otherwise.} \end{cases}$$

It is easy to show by induction and using property (2) that for every $n \geq 0$,

$$c_n \leq \max\{k_a, k_b, k_c\} r_b^n.$$

Moreover, the property can be easily generalized to the case when c_n is defined in terms of more than two recurrence relations of type (1).