Computing Answer Sets Using Program Completion

Yuliya Babovich and Vladimir Lifschitz

Department of Computer Sciences
University of Texas, Austin, USA
{yuliya, vl}@cs.utexas.edu

Abstract. The answer sets for a tight logic program can be characterized as the models of its completion, and consequently they can be generated using a satisfiability solver. This is the main idea of Version 1 of the answer set solver CMODELS—a system that computes answer sets for the logic programs that are tight or can be transformed into tight programs. Except for this restriction, CMODELS has the same input language as the answer set solver SMODELS. We discuss the computational methods used in Version 1 of CMODELS and the mathematical principles they are based on, and compare the performance of the two systems.

1 Introduction

Computing answer sets for logic programs is an important computational problem. The creation of efficient systems for computing answer sets has led to the development of the programming methodology known as answer set programming. Such systems are called answer set solvers, and the two best known systems of this kind are SMODELS with its front-end LPARSE[1] and DLV[2]. These systems are somewhat similar to SAT solvers—systems for solving the propositional satisfiability problem. But in some ways the problem of computing answer sets is computationally more difficult.

This paper is about Version 1 of the answer set solver CMODELS,[3] which computes answer sets using SAT solvers for search. It is similar to SMODELS in that its input can be generated by the preprocessor LPARSE. Besides negation as failure, the input may contain cardinality expressions ("constraint literals" in the terminology of [1, Section 5.3]) and choice rules [1, Section 5.4]—two constructs widely used in answer set programming.[4] Handling these constructs is based on the method for eliminating them in favor of nested expressions in the sense of [2] that is described in [3].

The use of SAT solvers for generating answer sets is based on the fact that for the logic programs with a certain syntactic property, called tightness, the

[1] `http://www.tcs.hut.fi/Software/smodels/` .
[2] `http://www.dbai.tuwien.ac.at/proj/dlv/` .
[3] `http://www.cs.utexas.edu/users/tag/cmodels` . About Version 2 see Section 5.
[4] The input can also contain general weight expressions ("weight literals"). However, optimize statements [1, Section 5.6] are not allowed.

answer set semantics is equivalent to the completion semantics [4], [5], [6]. Programs used in answer set programming, including those related to planning and commonsense reasoning, are often tight. CMODELS simplifies the given program, and, if the simplified program is tight, forms its completion. Then it converts the completion to clausal form and calls a SAT solver to find its models. This method is applicable only when the simplified program is tight; if it is not, Version 1 of CMODELS will tell the user that it is unable to find answer sets for the given program. On the other hand, in some cases CMODELS computes answer sets faster than "general-purpose" answer set solvers.

The system was written by the first author with the help of other members of Texas Action Group at Austin.[5] The program includes some parts of the code of SMODELS written by Patrik Simons.

In the next section we review the syntax of logic program rules of two kinds—rules generated by LPARSE and rules with nested expressions. Then we discuss the computational methods used in Version 1 of CMODELS and the mathematical principles they are based on (Section 3), and report on experimental results (Section 4). We conclude by brief comments on related work and future plans (Section 5).

2 Syntax of Rules

The output of LPARSE is a fully grounded program—a program without variables. Accordingly, in this paper we understand the word "atom" as in propositional logic.

2.1 Lparse Programs

The output of LPARSE may include rules of three types. We define a *basic rule* to be an expression of the form

$$A_0 \leftarrow A_1, \dots, A_m, not\ A_{m+1}, \dots, not\ A_n \qquad (1)$$

where A_0 is an atom or the symbol $\bot$, and $A_1, \dots, A_n$ are atoms ($n \geq m \geq 0$). A *choice rule* has the form

$$\{A_0, \dots, A_k\} \leftarrow A_{k+1}, \dots, A_m, not\ A_{m+1}, \dots, not\ A_n \qquad (2)$$

where A_i are atoms ($n \geq m \geq k \geq 0$). A *weight constraint rule* has the form

$$A_0 \leftarrow L\,\{A_1 = w_1, \dots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \dots, not\ A_n = w_n\} \qquad (3)$$

where A_0 is an atom or the symbol $\bot$, $A_1, \dots, A_n$ are atoms ($n \geq 0$), and L (lower bound) and $w_1, \dots, w_n$ (weights) are integers.

In particular, a *cardinality constraint rule* is a weight constraint rule with all weights equal to 1. In such a rule, all $= w_i$ can be dropped.

An lparse program is a finite set of basic, choice and weight constraint rules. The concept of an answer set (stable model) for programs of this kind, and for more general logic programs, is defined in [7] and reviewed in [3, Section 3].

[5] `http://www.cs.utexas.edu/users/tag/` .

2.2 Basic Nested Programs

A *basic nested rule* is an expression of the form

$$A_0 \leftarrow A_1, \ldots, A_k, not\ A_{k+1}, \ldots, not\ A_m, not\ not\ A_{m+1}, \ldots, not\ not\ A_n \quad (4)$$

where A_0 is an atom or the symbol $\bot$, and $A_1, \ldots, A_n$ are atoms ($n \geq m \geq k \geq 0$). The list ("conjunction") $A_1, \ldots, A_k$ in the body of the rule will be called the *positive part* of the body.

A basic nested program is a finite set of basic nested rules. This is a special case of the concept of a program with nested expressions introduced in [2]. In the general case, three connectives—negation as failure, conjunction and disjunction (;)—can be nested arbitrarily. The concept of an answer set for such programs is defined in [2] and reviewed in [3, Section 2].

If the body of a rule of any of the types described above is empty then we will drop $\leftarrow$. Rules with the head $\bot$ are called *constraints*; in a constraint, we will drop the head.

2.3 Turning an Lparse Program into a Basic Nested Program

The results of [8] and [3, Section 4.3] show that choice rules and weight constraint rules can be eliminated from an lparse program at the price of introducing additional atoms.[6] In other words, any lparse program can be turned into an essentially equivalent set of basic rules. This fact is important for the design of CMODELS, because the usual definition of program completion [9] covers neither choice rules nor weight constraint rules.

The computational procedure employed in CMODELS uses a translation that is similar to the one defined in [3, Section 4.3] but produces programs of a slightly more general form—basic nested programs in the sense of Section 2.2. This modification allows us to reduce the number of additional atoms introduced in the translation. The concept of completion is extended to programs with nested expressions in the bodies of rules in [10], and the tightness condition is generalized to such programs in [6].

For instance, the lparse program

$$\begin{array}{l} \{p, q\} \\ r \leftarrow p \\ r \leftarrow q \end{array} \quad (5)$$

is translated by CMODELS into the following program with nested expressions:

$$\begin{array}{l} p \leftarrow not\ not\ p \\ q \leftarrow not\ not\ q \\ r \leftarrow p \\ r \leftarrow q. \end{array} \quad (6)$$

[6] In the first paper, this theorem is stated only for the case of cardinality constraint rules (all weights are equal to 1). Another advantage of the translation defined in the second paper is that it is more economical.

The completion of this program is

$$\begin{array}{l} p \equiv \neg\neg p \\ q \equiv \neg\neg q \\ r \equiv p \vee q. \end{array} \tag{7}$$

To find the answer sets for program (5), CMODELS calls a SAT solver to find the models of (7).

3 Operation of CMODELS

In the process of its operation, CMODELS

(1) simplifies the given lparse program,
(2) turns it into a basic nested program,
(3) verifies that the resulting program is tight,
(4) forms the program's completion and calls a SAT solver to find its models.

We will discuss each of these steps one by one.

3.1 Simplifying the Program

Some of the simplifications described below are similar to those involved in the operation of SMODELS [11].

If Π is an lparse program or a basic nested program, by $At^{+}(\Pi)$ we denote the intersection of all answer sets of Π, and $At^{-}(\Pi)$ stands for the set of atoms that do not belong to any of the answer sets of Π.

Theorem 1. *Any lparse program Π of the form*

$$\begin{array}{l} Head \leftarrow Body, F \\ \Pi' \end{array}$$

where $F \in At^{-}(\Pi)$ has the same answer sets as

$$\begin{array}{l} \leftarrow F \\ \Pi'. \end{array}$$

Theorem 2. *Let Π be an lparse program of the form*

$$\begin{array}{l} Head \leftarrow Body, not\ F \\ \Pi'. \end{array}$$

(a) If $F \in At^{+}(\Pi)$ then Π has the same answer sets as

$$\begin{array}{l} \leftarrow not\ F \\ \Pi'. \end{array}$$

(b) If $F \in At^{-}(\Pi)$ then Π has the same answer sets as

$$\begin{array}{l} \leftarrow F \\ Head \leftarrow Body \\ \Pi'. \end{array}$$

Example 1. Let Π be the program

$$\begin{array}{l} a \leftarrow b, not\ c \\ d \leftarrow not\ a \\ c \leftarrow not\ d. \end{array}$$

The only answer set for this program is $\{d\}$. Consequently, $At^+(\Pi) = \{d\}$ and $At^-(\Pi) = \{a, b, c\}$. According to Theorem 1, Π has the same answer sets as program

$$\begin{array}{l} \leftarrow b \\ d \leftarrow not\ a \\ c \leftarrow not\ d. \end{array}$$

For this program Π',

$$At^+(\Pi') = At^+(\Pi) = \{d\}$$

and

$$At^-(\Pi') = At^-(\Pi) = \{a, b, c\}.$$

By Theorem 2(a), Π' has the same answer sets as

$$\begin{array}{l} \leftarrow b \\ d \leftarrow not\ a \\ \leftarrow not\ d. \end{array}$$

This program can be further simplified using Theorem 2(b), and the result is

$$\begin{array}{l} \leftarrow b \\ \leftarrow a \\ d \leftarrow \\ \leftarrow not\ d. \end{array}$$

Theorem 3. *Any lparse program Π of the form*

$$\begin{array}{l} Head \leftarrow L\,\{F = w,\ Tail\} \\ \Pi' \end{array}$$

where $F \in At^-(\Pi)$ has the same answer sets as

$$\begin{array}{l} \leftarrow F \\ Head \leftarrow L\,\{\,Tail\} \\ \Pi'. \end{array}$$

Theorem 4. *Let Π be an lparse program of the form*

$$\begin{array}{l} Head \leftarrow L\,\{not\ F = w,\ Tail\} \\ \Pi'. \end{array}$$

(a) If $F \in At^+(\Pi)$ then Π has the same answer sets as

$$\begin{array}{l} \leftarrow not\ F \\ Head \leftarrow L\,\{\,Tail\} \\ \Pi'. \end{array}$$

(b) If $F \in At^-(\Pi)$ then Π has the same answer sets as

$$\begin{array}{l} \leftarrow F \\ Head \leftarrow L - w\,\{Tail\} \\ \Pi'. \end{array}$$

Example 2. Let Π be the program

$$\begin{array}{l} \{a\} \\ c \leftarrow 4\{a = 1, b = 1, not\ d = 2\}. \end{array}$$

The answer sets for this program are $\{a\}$ and $\emptyset$. Consequently, $At^+(\Pi) = \emptyset$ and $At^-(\Pi) = \{b, c, d\}$. According to Theorem 3, Π has the same answer sets as the program

$$\begin{array}{l} \{a\} \\ \leftarrow b \\ c \leftarrow 4\{a = 1, not\ d = 2\}. \end{array}$$

For this program Π',

$$At^+(\Pi') = At^+(\Pi) = \emptyset$$

and

$$At^-(\Pi') = At^-(\Pi) = \{b, c, d\}.$$

By Theorem 4(b), Π' has the same answer sets as

$$\begin{array}{l} \{a\} \\ \leftarrow b \\ \leftarrow d \\ c \leftarrow 2\{a = 1\}. \end{array}$$

The SMODELS procedures *Atmost* and *Atleast* [11, Sections 4.1–4.3] [12, Section 7], applied to a program Π and the empty set, find a subset of $At^+(\Pi)$ and a subset of $At^-(\Pi)$. This part of the code of SMODELS is incorporated in CMODELS, and the sets computed by these procedures are used to simplify the program in accordance with Theorems 1–4.

3.2 Translating to a Basic Nested Program

As discussed above, before performing the completion procedure, CMODELS converts the simplified lparse program into a basic nested program using the ideas of [3]. In Section 2.3 we saw an example of translating a choice rule into a set of basic nested rules. Now we will consider an example of eliminating a weight constraint rule—an operation that may involve the introduction of auxiliary atoms.

Example 3. The rule

$$p \leftarrow 3\,\{q = 3, r = 2, s = 2\}$$

is first replaced by four simpler rules:

$$\begin{array}{l} p \leftarrow aux_1 \\ aux_1 \leftarrow 3\,\{r = 2, s = 2\} \\ p \leftarrow q, aux_2 \\ aux_2 \leftarrow 0\,\{r = 2, s = 2\}. \end{array}$$

Intuitively, the first pair of rules corresponds to the answer sets that don't contain q, and the second pair to the answer sets that do. The auxiliary variable aux_1 is an "abbreviation" for the expression $3\,\{r = 2, s = 2\}$, and aux_2 stands for $0\,\{r = 2, s = 2\}$. Since the last expression is identically true, aux_2 can be dropped from the program; we get:

$$\begin{array}{l} p \leftarrow aux_1 \\ aux_1 \leftarrow 3\,\{r = 2, s = 2\} \\ p \leftarrow q. \end{array}$$

The next step is similar:

$$\begin{array}{l} p \leftarrow aux_1 \\ aux_1 \leftarrow aux_3 \\ aux_3 \leftarrow 3\,\{s = 2\} \\ aux_1 \leftarrow r, aux_4 \\ aux_4 \leftarrow 1\,\{s = 2\} \\ p \leftarrow q. \end{array}$$

Here aux_3 stands for $3\,\{s = 2\}$, and aux_4 stands for $1\,\{s = 2\}$. Since $3\,\{s = 2\}$ is identically false, the program can be simplified as follows:

$$\begin{array}{l} p \leftarrow aux_1 \\ aux_1 \leftarrow r, aux_4 \\ aux_4 \leftarrow 1\,\{s = 2\} \\ p \leftarrow q. \end{array}$$

Finally, we use the fact that $1\{s = 2\}$ can be equivalently replaced by s, and arrive at a set of basic rules:

$$\begin{array}{l} p \leftarrow aux_1 \\ aux_1 \leftarrow r, aux_4 \\ aux_4 \leftarrow s \\ p \leftarrow q. \end{array}$$

Every auxiliary atom introduced in the process of translating a weight constraint rule (3) "abbreviates" an expression that differs from the body

$$L\,\{A_1 = w_1, \ldots, A_m = w_m, not\ A_{m+1} = w_{m+1}, \ldots, not\ A_n = w_n\}$$

of that rule in two ways: the lower bound L is replaced by an integer between 1 and L, and the list of equalities in the brackets is replaced by its proper nonempty suffix. It follows that the number of auxiliary atoms in the translation of rule (3) does not exceed $L(n-1)$. If (3) is a cardinality constraint rule (that is, $w_1 = \cdots = w_n = 1$) then we can also say that the lower bound is replaced by an integer between $L - n + 1$ and L, so that the number of auxiliary atoms in the translation of a cardinality constraint rule can be estimated from above by $n(n-1)$.

3.3 Verifying Tightness

The definition of an absolutely tight program below is a special case of the definition from [6, Section 5], which is based on [4]. Answer sets for such a program are identical to the models of its completion ([4], Theorem 3.2; [6], Corollary 6).

The *positive dependency graph* of a basic nested program Π is the directed graph G such that

- the vertexes of G are the atoms occurring in Π,
- G has an edge from A to A' if Π has a rule with the head A' that contains A in the positive part of the body.

A program is *(absolutely) tight* if its positive dependency graph has no cycles.

Example 4. The positive dependency graph of the program

$$\begin{array}{l} p \leftarrow not\ not\ p \\ p \leftarrow q \\ q \leftarrow p \end{array} \tag{8}$$

consists of two vertexes p and q and two edges, from p to q and from q to p. The graph has a cycle and hence the program is not tight.

Before verifying whether the basic nested program generated from the input is tight, CMODELS applies to it two transformations that sometimes turn a non-tight program into a tight program. First, it drops all rules of the form

$$A \leftarrow \ldots, A, \ldots.$$

Obviously a program containing such rules cannot be tight, and removing these rules from a program does not change its answer sets.

Second, if the program contains a rule of the form

$$A \leftarrow not\ not\ A \tag{9}$$

then CMODELS replaces all occurrences of A in the positive parts of the bodies of the rules of the program by *not not* A. For instance, in application to program (8), this transformation inserts *not not* in front of p in its last rule, and the program becomes tight. The proposition below shows that this is an equivalent transformation. Recall that a program Π_1 is *strongly equivalent* to a program Π_2 if, for any program Π, $\Pi_1 \cup \Pi$ has the same answer sets as $\Pi_2 \cup \Pi$ [13].

Theorem 5. *Any basic nested program of the form*

$$\begin{array}{l} Head \leftarrow A, Body \\ A \leftarrow not\ not\ A \end{array}$$

is strongly equivalent to the program

$$\begin{array}{l} Head \leftarrow Body, not\ not\ A \\ A \leftarrow not\ not\ A. \end{array}$$

During the tightness verification step, CMODELS builds the positive dependency graph and uses a standard depth first search algorithm [14, Section 23.3] to detect a cycle in it. To be more precise, the algorithm is applied to the subgraph of the positive dependency graph that does not contain some "inessential" vertexes:

- the atoms that do not occur in the heads of rules, and
- the atoms that occur in rules of the form (9).

3.4 Simplifying and Clausifying Completion

The completion of a basic nested program Π can be formed as follows. First, in the body of every rule of Π replace each occurrence of *not* with $\neg$ and each comma with $\wedge$. Second, for every atom A make the list of all rules in Π with the head A:

$$A \leftarrow Body_i$$

and form the equivalence

$$A \equiv \bigvee_i Body_i. \tag{10}$$

Third, for every constraint $\bot \leftarrow Body$ in Π form the negation of its body:

$$\neg Body. \tag{11}$$

The completion $Comp(\Pi)$ of Π consists of all formulas (10) and (11). For instance, in application to program (6) this process produces formulas (7).

Before converting the completion of a tight program Π to clausal form, CMODELS simplifies it using the subsets of $At^+(\Pi)$ and $At^-(\Pi)$ computed by the procedures *Atmost* and *Atleast* (Section 3.1). Since every atom A from $At^+(\Pi)$ belongs to all answer sets of Π, and the answer sets of Π are identical to the models of $Comp(\Pi)$, A is entailed by $Comp(\Pi)$. The completion of Π is simplified by replacing all occurrences of A in it by $\top$ and adding the atom A as a new formula. Since the atoms A from $At^-(\Pi)$ do not belong to the answer sets of Π, $\neg A$ is entailed by $Comp(\Pi)$ for such atoms. The completion of Π is simplified by replacing all occurrences of A in it by $\bot$ and adding $\neg A$ as a new formula.

The process of converting an equivalence (10) whose left-hand side is not replaced by $\bot$ to clausal form usually begins with distributing the disjunction in the right-hand side over the conjunctions $Body_i$. But if at least two of these disjunctive terms contain more than one literal then auxiliary atoms are introduced as abbreviations for these disjunctive terms, to avoid a significant increase in the size of the formula. For instance, the equivalence

$$p \equiv (q_1 \wedge r_1) \vee (q_2 \wedge r_2) \vee s$$

is turned into

$$\begin{aligned} &p \equiv aux_1 \vee aux_2 \vee s \\ &aux_1 \equiv q_1 \wedge r_1 \\ &aux_2 \equiv q_2 \wedge r_2 \end{aligned}$$

as a preliminary step.

4 Experimental Data

Tables 1 and 2 show the running times of CMODELS and SMODELS, in seconds, for several benchmark problems. The data for most problems, and many of the programs in the input language of LPARSE used in the tests, can be found at

`http://www.cs.engr.uky.edu/ai/benchmarks.html` ,
`http://www.tcs.hut.fi/Software/smodels/tests/lp-csp-tests.tar.gz` ,
`http://www.tcs.hut.fi/~kepa/experiments/boundsmodels/` .

The SAT solver invoked by CMODELS in these experiments is MCHAFF[7], except of the experiments reported in one column of Table 2 that describes the results of running CMODELS with RELSAT[8]. In Table 1 we demonstrate the performance of systems with respect to the task of finding one solution, while in Table 2 systems search for many solutions. We restrict our experimental analysis only to comparison with SMODELS since both systems have identical input. The experiments of Tables 1 and 2 were run on a Pentium IV 1.8 Ghz processor with 512MB of RAM, DDR 266Mhz, running Linux.

CMODELS with MCHAFF can perform much better than SMODELS (sometimes having more than 3 orders of magnitude speed-up) when the task is to compute one solution. However, when computing all solutions, CMODELS with MCHAFF is outperformed by CMODELS with RELSAT and also by SMODELS. Overall, the results show that there is a configuration of CMODELS which can be competitive with other solvers at the state-of-the-art.

Table 3 illustrates the role of the process of program simplification described in Section 3.1. The "preassigned queens" problem in that table is the N queens problem in which the positions of several queens are preassigned. In Problem 6.70.2, for instance, $N = 70$ and the positions of 6 queens are preassigned. Simplification can have a very positive impact on the overall performances of the system. On all the instances but one, the performances of both CMODELS and MCHAFF are better with simplification enabled. Performance of tightness

[7] `http://www.ee.princeton.edu/~chaff/software.php` .
[8] `http://www.satlib.org/Solvers/SAT/REL_SAT.2/relsat_2.00.tar.gz` .

Table 1. Finding one solution to the N queens, 4-coloring planar graphs, blocks world planning and bounded model checking problems.

Problem		CMODELS	SMODELS
N queens,	$N = 19$	0.48	2.64
	21	0.47	73.31
	24	0.46	972.01
	50	1.68	> 1 hour
Coloring planar graphs	p600	0.44	7.58
	p1000	0.89	23.20
	p6000	71.40	838.25
Blocks world planning	bw-large.c.7	0.36	0.14
	bw-large.c.8	0.40	1.24
	bw-large.d.8	0.56	0.25
	bw-large.d.9	0.65	2.42
	bw-large.e.9	0.81	0.38
	bw-large.d.10	0.97	2.75
Bounded model checking	dp_10.fsa-D-i-O2-b10	0.28	165.01
	dp_12.fsa-D-i-O2-b9	340.51	646.96
	elevator_2-D-i-O2-b12	2.80	2.60
	elevator_3-D-i-O2-b15	85.83	113.94
	elevator_4-D-i-O2-b13	35.14	719.81
	elevator_4-D-s-O2-b10	8.24	60.99
	key_2-D-i-O2-b29	200.53	2088.77
	key_2-D-s-O2-b29	44.11	1893.34
	mmgt_3.fsa-D-i-O2-b10	4.51	23.98
	mmgt_4.fsa-D-i-O2-b12	252.17	2855.44
	mmgt_4.fsa-D-s-O2-b8	3.55	172.23
	q_1.fsa-D-i-O2-b17	916.50	1092.21

verification step is also influenced greatly by simplification since its performance is related to the number of atoms in the program. Finally, the positive impact of the simplification is most noticeable in the results on 6.70.5 problem instance: the problem gets solved by the simplification procedure.

In each of the CMODELS running times reported in Tables 1–3, the search time (that is, the running time of the SAT solver) is between 30% and 95% of the total. The percentage is usually higher for more difficult computational problems.

5 Conclusion

The use of program completion is a promising approach to the design of answer set solvers, even when choice rules and cardinality expressions are allowed in the input. System ASSAT[9] uses program completion for generating answer sets

[9] `http://assat.cs.ust.hk/` .

Table 2. Finding all solutions to the N queens problem.

N	CMODELS with MCHAFF	CMODELS with RELSAT	SMODELS
9	0.37	0.17	0.38
10	6.42	0.56	1.62
11	112.04	3.40	8.38
12	1657.72	16.25	52.51

Table 3. The effect of program simplification (Section 3.1) on the running time of CMODELS, on its two components—the time spent on the running time of MCHAFF and the verification of tightness, on the number of atoms and the number of clauses generated by CMODELS. In each entry, the number before the slash refers to the case when the simplification step is omitted. The experiments were run on SunBlade 100 Workstation, 500 MHz Ultra Sparc, 128 MB RAM.

Problem		CMODELS	MCHAFF	Tightness verification	Number of atoms	Number of clauses
Blocks	bw-large.c.7	2.60/1.51	0.65/0.39	0.69/0.12	6281/5298	60512/32939
world	bw-large.c.8	3.23/1.84	1.74/0.55	0.8/0.15	7096/6098	68256/40052
planning	bw-large.d.8	3.97/2.25	1/0.6	1.12/0.19	8920/7553	92448/51312
	bw-large.d.9	4.60/2.73	1.28/0.76	1.23/0.22	9945/8561	102924/61005
	bw-large.e.9	5.78/3.22	1.35/0.84	1.67/0.25	12211/10341	135320/74616
	bw-large.e.10	6.89/3.73	2.04/0.99	1.86/0.31	13470/11581	149080/87425
Preassigned	6.70.2	27.15/15.15	6/3.39	7.37/0.01	5251/5251	577087/352630
queens	6.70.5	26.14/4.79	5.19/0	7.46/0	5251/0	577087/0
Schur	schur4.42	1.43/1.38	0.85/0.96	0.17/0.02	588/588	11634/5325
numbers	schur4.43	1.73/1.41	1.14/0.97	0.18/0.03	602/602	12169/5580
15-puzzle	puzzle12.12	1.84/1.47	0.73/0.78	0.26/0.1	12884/6533	42766/18019
	puzzle13.13	4.34/5.93	3.1/4.85	0.28/0.18	13975/11300	46426/35747
	puzzle14.14	3.54/1.79	2.19/0.96	0.3/0.13	15066/7871	50086/22211

also. It differs from CMODELS in two ways. First, it adds loop formulas [15] to the completion, and consequently is capable of handling nontight programs. Second, neither choice rules nor cardinality expressions are allowed in the input of the current version of ASSAT. For the benchmarks that can be processed by both systems—tight programs consisting of basic rules—the running times of CMODELS and ASSAT are comparable; differences in performance can be explained to a large degree by differences in the choice of parameters the two systems select when SAT solver is invoked. Version 2 of CMODELS, currently under development, will combine the advantages of both systems.

Lee and Lifschitz [16] extended the concepts of completion, tight program and loop formula to disjunctive logic programs. We plan to investigate the possibility of using this idea for the design of an answer set solver for disjunctive programs comparable to the system DLV.

Acknowledgments

We are grateful to Enrico Giunchiglia, Marco Maratea, Hudson Turner for useful suggestions related to the paper, to Patrik Simons for permitting to use SMODELS code, to Miroslaw Truszczyński for communicating to us preassigned queens benchmarks. This work was partially supported by the Texas Higher Education Coordinating Board under Grant 003658-0322-2001.

References

1. Tomi Syrjanen. Lparse manual[10] In *Lparse Manual*, pages 1–56, 2003.
2. Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.
3. Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. In progress, 2003.
4. François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
5. Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' theorem and answer set programming.[11] In *Proc. Eighth Int'l Workshop on Non-Monotonic Reasoning*, 2000.
6. Esra Erdem and Vladimir Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 2003. To appear.
7. Ilkka Niemelä and Patrik Simons. Extending the Smodels system with cardinality and weight constraints. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 491–521. Kluwer, 2000.
8. Victor Marek and Jeffrey Remmel. On logic programs with cardinality constraints. In *Proc. NMR-02*, 2002.
9. Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
10. John Lloyd and Rodney Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.
11. Patrik Simons. Extending and implementing the stable model semantics. In *Doctoral dissertation*, pages 305–316, 2000.
12. Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138:181–234, 2002.
13. Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
14. Thomas Cormen, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. MIT Press, 1994.
15. Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. In *Proc. AAAI-02*, 2002.
16. Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs[12] Unpublished draft, 2003.

[10] `http://www.tcs.hut.fi/software/smodels/lparse.ps.gz` .
[11] `http://arxiv.org/abs/cs.ai/0003042` .
[12] `http://www.cs.utexas.edu/users/appsmurf/papers/disjunctive.ps` .