Implementing Preprocessing Transformations for Smodels Extended Rules

Axel Eirola

6th October 2008

Abstract

We examine the utilisation of known syntactic transformations of Answer Set Programming (ASP) for preprocessing programs written with extended rules of the *smodels* format. As an byproduct we also introduce a candidate for a canonical form for representing said rules. In addition we examine the performance benefit given by the transformation on solvers supporting the *smodels* format, in this paper focusing on *smodels* and *clasp*.

1 Introduction

With the application of *smodels* [7] based solving in many fields combined with the availability and usage of various logic program translators we often encounter logical programs in a unoptimised representation, be it either because of machine translations, or lack of resources for thorough optimisations by hand. To tackle the issue of decreased performance of poor encodings usage of preprocessing is advised, but still nearly all available Answer Set Programming (ASP) [6] solvers lack the ability.

Building on previous studies [2] of simplifying transformations for ASP and inspired by the recent work by Gebser [4] we seek to find an efficient and widely applicable way to speed up solving of *smodels* formatted ASP problems with any compatible solver by reducing the given program to a simplified more optimised, yet equivalent, representation before handing it over to the solver. The function is largely similar of that which has been done inside clasp [3], but the method rather different. The basic idea is to implement known ASP rule transformations for the three extended *smodels* rule types and search the given program for set of rules with specific attributes, and with the help of the predetermined transformations simplify these to more efficient representations. With the help of these simplifications we hope that the implementation of this technique in our tool *simplify* [1] will enable us to in deterministic polynomial time reduce the exponential runtime of the actual problem solving.

Additionally we introduce a very simple canonical form for easily representing and sorting rules in *smodels* format. The original usage for this normalisation was for increasing internal performance of the implementation in *simplify* , but it may also be used for wider applications.

2 Background

Striving to use a similar notation to both *lparse* and *smodels* [8] internal formats we define a logic program P as a triple (R, O, C) containing rules, output atoms and compute statements respectively. Rules consist of a head and optionally empty body, heads consisting of atoms, and bodies of literals (an atom with an optionally preceding negation). Atoms have both a unique textual name (a) , and identifying integer $(\#a)$. Rules can be one of four types, basic rules, constraint rules, choice rules and weight rules, denoted as following:

Basic rule: $a \leftarrow b_1, \ldots, b_p, \mathit{not}\ b_{p+1}, \ldots, \mathit{not}\ b_n$
Choice rule: $\{a_1, \ldots, a_h\} \leftarrow b_1, \ldots, b_p, \mathit{not}\ b_{p+1}, \ldots, \mathit{not}\ b_n$
Constraint rule: $a \leftarrow l\{b_1, \ldots, b_p, \mathit{not}\ b_{p+1}, \ldots, \mathit{not}\ b_n\}$
Weight rule: $a \leftarrow l[b_1 = w_1, \ldots, b_p = w_p, \mathit{not}\ b_{p+1} = w_{p+1}, \ldots, \mathit{not}\ b_n = w_n]$

The output atoms O is a set of atoms that are shown to the user and specify the solution to the problem at hand, in this paper they only appear in situations where it is essential to keep a redundant rule to retain definitions for output atoms. A compute statement provides a set of literals ($C = \{c_1, \ldots, c_p, \mathit{not}\ c_{p+1}, \ldots, \mathit{not}\ c_n\}$) that are required to be true in all stable models.

For the sake of saving space we will use the following shorthands for commonly used sets and elements:

$H = \{a_1, \ldots, a_h\}$
$B^+ = \{b_1, \ldots, b_p\}$
$B^- = \{b_{p+1}, \ldots, b_n\}$
$\mathit{not}\ B^- = \{\mathit{not}\ b_{p+1}, \ldots, \mathit{not}\ b_n\}$
$B = B^+ \cup \mathit{not}\ B^-$
$W = w_1, \ldots, w_n$
$C^+ = \{c_1, \ldots, c_p\}$
$C^- = \{c_{p+1}, \ldots, c_n\}$
$h(r)$: The head of a rule r
$B(r)$: The body of a rule r
$w(a)$: Weight of atom a
$W(B)$: List of weights for body B

3 Normalisation

Simplify uses a canonical form for representing rules to speed up comparisons as well as sorting and search algorithms. The canonical form is similar to that of the *smodels* file format with the special requirement that all integer representations of atoms in the head, negative body and positive body must be in ascending order. More formally:

$1␣\#a␣(p+n)␣n␣\#b_1␣\cdots␣\#b_p␣\#b_{p+1}␣\cdots␣\#b_n$
$2␣\#a␣(p+n)␣n␣l␣\#b_1␣\cdots␣\#b_p␣\#b_{p+1}␣\cdots␣\#b_n$
$3␣h␣\#a␣\cdots␣\#b_h(p+n)n\#b_1␣\cdots␣\#b_p␣\#b_{p+1}␣\cdots␣\#b_n$
$5␣\#a␣l␣(p+n)␣n␣\#b_1␣\cdots␣\#b_p␣\#b_{p+1}␣\cdots␣\#b_n␣w_{p+1}␣\cdots␣w_{p+n}␣w_1␣\cdots␣w_p$

where

$\#a_1 < \ldots < \#a_h$
$\#b_1 < \ldots < \#b_p$
$\#b_{p+1} < \ldots < \#b_n$

Notice how the canonical form requires that there are no duplicate literals in the positive nor negative bodies, therefore duplicates are removed at program input and in the case of weight rules the sum of weights of the duplicate pair is set as the weight of the remaining literal.

4 Transformations

In the following we present the simplifications as they are used in the tool simplify. They are mostly based on the special case of single-atom heads for transformations addressed in [2] and implemented for the three additional rules of the *smodels* format. Note that for clarity the programs are here represented as sets of rules, and bodies as sets of literals, implying that no duplicates exist, therefore we do not address them here although they are properly taken care of in the tool implementation.

4.1 Trivialities

Trivial instances of non-basic rules can be reduced to simpler notation without loss of information, this includes constraint and weight rules with trivial or unsatisfiable limits.

Transformation	Condition
$R' = R \setminus \{H \leftarrow B\}$	$\lvert H\rvert = 0$
$R' = (R \setminus \{h \leftarrow l\{B\}\}) \cup \{h\}$	$l = 0$
$R' = (R \setminus \{h \leftarrow l\{B\}\}) \cup \{h \leftarrow B\}$	$l = \lvert B\rvert$
$R' = R \setminus \{h \leftarrow l\{B\}\}$	$l > \lvert B\rvert$
$R' = (R \setminus \{h \leftarrow l[B]\}) \cup \{h\}$	$l = 0$
$R' = (R \setminus \{h \leftarrow l[B]\}) \cup \{h \leftarrow B\}$	$\sum W(B) - \min\{w(b) \mid b \in B\} < l$
$R' = (R \setminus \{h \leftarrow l[B]\}) \cup \{h \leftarrow 1\{B\}\}$	$l < \min\{w(b) \mid b \in B\}$
$R' = R \setminus \{h \leftarrow l[B]\}$	$l > \sum W(B)$
$R' = (R \setminus \{h \leftarrow l[B]\}) \cup \{h \leftarrow \lceil \frac{l}{w} \rceil\{B\}\}$	$\forall w' \in W(B) : w' = w$

Table 1: Trivial transformations

4.2 Contradictions

Contradiction transformations build upon the logical fact that if the positive and the negative body contains a common atom one of the literals has to be true and the other one false, allowing us to remove the whole rule, as the body can never be satisfied.

Transformation	Condition
$R' = R \setminus \{h \leftarrow B\}$	$B^+ \cap B^- \neq \emptyset$
$R' = R \setminus \{H \leftarrow B\}$	$B^+ \cap B^- \neq \emptyset$

Table 2: Transformations for rules with a contradictory body

4.3 Tautologies

Positive literals occurring in both the head and the body of a rule means that for the head to be true it must already be true in the body, resulting in the head atom being redundant.

Transformation	Condition
$R' = R \setminus \{h \leftarrow B\}$	$h \in B^+$
$R' = (R \setminus \{H \leftarrow B\}) \cup \{H \setminus B \leftarrow B\}$	$H \cap B^+ \neq \emptyset$
$R' = (R \setminus \{h \leftarrow l\{h, B\}\}) \cup \{h \leftarrow l\{B\}\}$	
$R' = (R \setminus \{h \leftarrow l[h, B]\}) \cup \{h \leftarrow l[B]\}$	

Table 3: Tautological transformations

4.4 Compute statements

Literals in compute statements existing in bodies of rules can in many cases be preemptively evaluated. The limitation being that positive literals cannot safely be removed from bodies while retaining equivalence, shown by the simple example of

$$a \leftarrow b.$$
$$b \leftarrow a.$$
$$compute\{a\}.$$

Transformation	Condition
$R' = R \setminus \{h \leftarrow B\}$	$B^+ \cap C^- \neq \emptyset$
$R' = R \setminus \{h \leftarrow B\}$	$B^- \cap C^+ \neq \emptyset$
$R' = R \setminus \{h \leftarrow \mathit{not}\, a, B\} \cup \{h \leftarrow B\}$	$a \in C^-$
$R' = R \setminus \{H \leftarrow B\}$	$B^+ \cap C^- \neq \emptyset$
$R' = R \setminus \{H \leftarrow B\}$	$B^- \cap C^+ \neq \emptyset$
$R' = (R \setminus \{H \leftarrow \mathit{not}\, a, B\}) \cup \{h \leftarrow B\}$	$a \in C^-$
$R' = (R \setminus \{h \leftarrow l\{\mathit{not}\, a, B\}\}) \cup \{h \leftarrow l\{B\}\}$	$a \in C^+$
$R' = (R \setminus \{h \leftarrow l\{\mathit{not}\, a, B\}\}) \cup \{h \leftarrow \min(0, l-1)\{B\}\}$	$a \in C^-$
$R' = (R \setminus \{h \leftarrow l[\mathit{not}\, a = w, B]\}) \cup \{h \leftarrow l[B]\}$	$a \in C^+$
$R' = (R \setminus \{h \leftarrow l[\mathit{not}\, a = w, B]\}) \cup \{h \leftarrow \min(0, l-w)[B]\}$	$a \in C^-$

Table 4: Compute transformations

4.5 Non-minimal

A rule that defines the same head as an existing rule, but requiring more literals to be satisfied, can in most cases be removed, as the smaller rule defines the head in all cases for which the larger rule defines it.

Transformation	Condition
$R' = R \setminus \{h \leftarrow B_2\}$	$\exists (h \leftarrow B_1) \in R : B_1 \subset B_2$
$R' = (R \setminus \{h_1, H_2 \leftarrow B_2\}) \cup \{H_2 \leftarrow B_2\}$	$\exists (h_1 \leftarrow B_1) \in R : B_1 \subseteq B_2$
$R' = (R \setminus \{H_2 \leftarrow B_2\}) \cup \{H_2 \setminus H_1 \leftarrow B_2\}$	$\exists (H_1 \leftarrow B_1) \in R : B_1 \subseteq B_2,$ $H_1 \neq H_2 \bigvee B_1 \neq B_2$
$R' = R \setminus \{h \leftarrow B_2\}$	$\exists (h \leftarrow l\{B_1\}) \in R : l \leq \lvert B_1 \cap B_2 \rvert$
$R' = R \setminus \{h \leftarrow l_2\{B_2\}\}$	$\exists (h \leftarrow l_1\{B_1\}) \in R : B_2 \subseteq B_1, l_1 \leq l_2,$ $B_1 \neq B_2 \bigvee l_1 \neq l_2$
$R' = R \setminus \{h \leftarrow B_2\}$	$\exists (h \leftarrow l[B_1]) \in R : l \leq \sum W_1(B_1 \cap B_2)$
$R' = R \setminus \{h \leftarrow l_2[B_2]\}$	$\exists (h \leftarrow l_1[B_1]) \in R, \forall b \in B_2 : w_2(b) \leq w_1(b),$ $B_2 \subseteq B_1, l_1 \leq l_2, B_1 \neq B_2 \bigvee l_1 \neq l_2$

Table 5: Non-minimal transformations

4.6 Literal non-minimal

Head atoms occurring in bodies alongside their defining bodies can be evaluated to true or false due to the fact that the head atom must always be true if the rest of the body, including its defining body, is true.

Transformation	Condition
$R' = (R \setminus \{h_2 \leftarrow h_1, B_1, B_2\} \cup \{h_2 \leftarrow B_1, B_2\}$	$\exists (h_1 \leftarrow B_1) \in R$
$R' = (R \setminus \{H_2 \leftarrow h_1, B_1, B_2\} \cup \{H_2 \leftarrow B_1, B_2\}$	$\exists (h_1 \leftarrow B_1) \in R$
$R' = (R \setminus \{h_2 \leftarrow \mathit{not}\, h_1, B_1, B_2\}$	$\exists (h_1 \leftarrow B_1) \in R$
$R' = (R \setminus \{H_2 \leftarrow \mathit{not}\, h_1, B_1, B_2\}$	$\exists (h_1 \leftarrow B_1) \in R$

Table 6: Literal non-minimal transformations

4.7 Partial evaluation

Uniquely defined heads appearing in bodies of other rules can be evaluated by replacing their occurrence with the body defining the head. This would also be possible for non-unique heads, but it results in an increased number of required rules, therefore not contributing to simplifying the program as a whole.

Transformation	Condition
$R' = (R \setminus \{h_2 \leftarrow h_1, B_2\}) \cup \{h_2 \leftarrow B_1 \cup B_2\}$	$\mathit{unq}(h_1)^1 = h_1 \leftarrow B_1$
$R' = (R \setminus \{H_2 \leftarrow h_1, B_2\}) \cup \{H_2 \leftarrow B_1 \cup B_2\}$	$\mathit{unq}(h_1) = h_1 \leftarrow B_1$
$R' = (R \setminus \{h_2 \leftarrow \mathit{not}\, h_1, B_2\}) \cup \{h_2 \leftarrow \mathit{not}\, b_1, B_2\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{H_2 \leftarrow \mathit{not}\, h_1, B_2\}) \cup \{H_2 \leftarrow \mathit{not}\, b_1, B_2\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{h_2 \leftarrow l\{h_1, B_2\}\}) \cup \{h_2 \leftarrow l\{b_1, B_2\}\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{h_2 \leftarrow l\{\mathit{not}\, h_1, B_2\}\}) \cup \{h_2 \leftarrow l\{\mathit{not}\, b_1, B_2\}\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{h_2 \leftarrow l[h_1 = w, B_2]\}) \cup \{h_2 \leftarrow l[b_1 = w, B_2]\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{h_2 \leftarrow l[\mathit{not}\, h_1 = w, B_2]\}) \cup \{h_2 \leftarrow l[\mathit{not}\, b_1 = w, B_2]\}$	$\mathit{unq}(h_1) = h_1 \leftarrow b_1$
$R' = (R \setminus \{h_2 \leftarrow l\{h_1, B_2\}\}) \cup \{h_2 \leftarrow l\{\mathit{not}\, b_1, B_2\}\}$	$\mathit{unq}(h_1) = h_1 \leftarrow \mathit{not}\, b_1$
$R' = (R \setminus \{h_2 \leftarrow l[h_1 = w, B_2]\}) \cup \{h_2 \leftarrow l[\mathit{not}\, b_1 = w, B_2]\}$	$\mathit{unq}(h_1) = h_1 \leftarrow \mathit{not}\, b_1$
$R' = (R \setminus \{h_2 \leftarrow l\{h_1, B_2\}\}) \cup \{h_2 \leftarrow \min(0, l-1)\{B_2\}\}$	$\mathit{unq}(h_1) = h_1 \leftarrow$
$R' = (R \setminus \{h_2 \leftarrow l\{\mathit{not}\, h_1, B_2\}\}) \cup \{h_2 \leftarrow l\{B_2\}\}$	$\mathit{unq}(h_1) = h_1 \leftarrow$
$R' = (R \setminus \{h_2 \leftarrow l[h_1 = w, B_2]\}) \cup \{h_2 \leftarrow \min(0, l-w)[B_2]\}$	$\mathit{unq}(h_1) = h_1 \leftarrow$
$R' = (R \setminus \{h_2 \leftarrow l[\mathit{not}\, h_1, B_2]\}) \cup \{h_2 \leftarrow l[B_2]\}$	$\mathit{unq}(h_1) = h_1 \leftarrow$
$R' = R \setminus \{h \leftarrow B\}$	$\neg\exists\, r \in R : h \notin O$, $h \in B^+(r) \bigvee \mathit{not}\, h \in B^-(r)$

Table 7: Partial evaluation transformations

[1] $\mathit{unq}(h) = r$, if $\exists!\, r \in R : h(r) = h$

5 Experiments

Conducted performance experiments are divided into two categories. One observing the individual effects of transformations on solving time in search for a safe set of transformations that only positively effects the performance. The second category of tests focuses on the combined effort of transformations considered safe to benchmark the actual benefit of the method. All tests were run with *smodels* version 2.32 (restarts enabled) and *clasp* version 1.05 on a 2.0 GHz Linux system with 3GB RAM, test instances were picked from the *SLparse* category in the ASP-competition [5].

Test	original	tcTC [2]	Non-min	Partev	Litnonmin
Blocked N-Queens	36.24	35.79	33.19	36.88	36.43
Factoring	27.66	27.41	27.53	34.2	27.28
Hamiltonian Path	28.74	28.92	25.68	17.68	28.8
Hashiwokakero	3.57	3.54	3.42	3.55	3.53
Knights Tour	2.28	2.25	2.26	2.31	2.24
Random Non Tight	32.26	23.63	33.15	33.42	32.82
RLP-150	5.51	4.75	5.48	3.52	6.24
Schur	10.61	10.52	4.43	10.68	10.6
searchTest plain	13.68	12.47	12.75	13.03	13.06
searchTest verbose	51.68	49.77	47.69	215.31	51.52
Social Golfer	1.02	1.35	0.96	1.15	0.82
Solitaire Backward	1.16	1.17	1.16	1.17	1.16
Solitaire Forward	9.22	17.93	10.93	29.59	10.27
Su-Doku	29.07	29.16	14.88	29.74	29.6
Towers Of Hanoi	76.85	72.27	78.16	79.24	73.51
verifyTest	0.53	0.52	0.5	0.48	0.52
Weighted Spanning Tree	243.35	240.48	228	234.92	242.2

Table 8: Transformation tests

In table 8 is displayed the average runtimes for smodels with original problem encoding and after different transformations, average was obtained by running one instance of each test nine times shuffled. The used tests were chosen because of their stable runtime with *smodels*. As can be seen all transformations except for partial evaluation either decreases or does not significantly impact performance, whereas partial evaluation behaves more erratically and can therefore not be considered safe.

The actual reason for the reduced performance of partial evaluations is hidden in the inner workings of the model solvers and outside the scope of this document. Speculating on the resons we encounter a possible explanation that the removed atoms in some significant way divide the search space giving the solver a good choice point for making its decisions. In this case an analysis of the nature of the transformation to be done seems much more expensive than the actual benefit available from the simplification.

[2]Trivialities, Contradictions, Tautologies and Compute statements

Alltough partial evaluation is considered unsafe it is not useless, as it in certain cases greatly decreases solving time. Exploiting problem specific program structure we can determine effectiveness of certain transformations on small instances of said problem and witness similar behaviour in larger instances.

Problem	**s**	**rules**		**smodels**		**clasp**	
		orig.	simp.	orig.	simp.	orig.	simp.
15-Puzzle	0.2	38250	38250	300.0_{55}	300.0_{55}	191.3_{16}	167.5_{11}
Blocked N-Queens	0.1	5024	4877	57.2	51.4	12.7	29.1_{2}
Bounded Spanning Tree	6.1	206557	197376	241.0_{31}	240.6_{32}	5.9	5.8
Car Sequencing	0.0	1582	1497	300.0_{55}	300.0_{55}	300.0_{55}	296.9_{54}
Factoring	0.0	7685	7685	8.9	8.7	9.6	9.3
Hamiltonian Cycle	0.0	10502	10502	181.4_{30}	163.7_{28}	0.2	0.3
Hamiltonian Path	0.0	4924	3499	179.6_{31}	159.5_{27}	0.1	0.1
Hashiwokakero	7.9	1077971	1062342	181.0_{29}	180.8_{31}	112.2_{15}	101.5_{12}
Knights Tour	0.4	58062	58062	48.8	50.0	0.9	0.9
Random Non Tight	0.0	848	796	72.1_{4}	56.6_{2}	29.6_{4}	8.7
RLP-150	0.0	735	728	3.9	4.0	0.4	0.4
RLP-200	0.0	793	786	31.1	24.0	1.1	1.1
Schur	1.4	85319	43187	242.1_{44}	240.9_{44}	110.4_{15}	97.0_{12}
searchTest plain	29.6	690808	633670	80.2_{11}	78.3_{11}	87.5_{14}	108.7_{18}
searchTest verbose	34.2	802803	736548	133.8_{11}	121.2_{11}	78.9_{8}	42.5_{2}
Social Golfer	0.2	31506	31219	180.4_{33}	185.9_{34}	60.8_{11}	60.8_{11}
Solitaire Backward	0.1	20508	20508	209.9_{38}	186.3_{33}	3.4	4.8
Solitaire Backward 2	0.2	27435	21580	292.0_{53}	296.6_{53}	189.4_{27}	164.3_{24}
Solitaire Forward	0.1	19606	19606	71.6_{11}	91.2_{14}	55.3_{10}	60.0_{10}
Su-Doku	33.4	1003593	503593	30.5	15.6	50.4_{4}	19.6_{1}
Towers Of Hanoi	0.1	18340	14728	206.4_{29}	197.6_{25}	36.3_{2}	33.8
Traveling Salesperson	0.0	3825	3825	137.7_{24}	150.8_{26}	0.5	0.3
verifyTest	0.1	12914	12190	0.5	0.5	0.2	0.2
Weight Bounded Dom. Set	0.0	3163	3163	298.3_{53}	300.0_{55}	134.1_{24}	129.1_{20}
Weighted Latin Square	0.0	997	781	256.0_{41}	262.5_{44}	0.0	0.0
Weighted Spanning Tree	2.0	112034	106186	255.6_{34}	211.1_{22}	3.6	3.4

Table 9: Benchmark results

Actual performance results are included in table 9 where the full test suite of the ASP-competition is utilised to give an overview of the impact on performance the safe transformations have on solvingtime with both *smodels* and *clasp*. For each problem five instances (except for Factoring 4, and Su-Doku 3) were run eleven times shuffled, with a timeout of 300 seconds. All available transformations except for partial evaluation were used. In the table s shows the average time in seconds taken by *simplify*, rules the average number of rules in the problem, smodels the average runtime for *smodels* and clasp the average runtime for *clasp*. The lefthand side value is for the original instance, and righthand side value for the simplified instance. Number of timeouts are shown as subscript after the value.

As we can notice not all problems can be efficiently simplified as there is always a limit on how efficiently a problem can be represented. But we still see a noticeable improvement in some of the cases. Comparing runtimes of *clasp* we can see that it is not as prone to improvement by simplification as *smodels*, mainly due to its already implemented internal preprocessing and SAT-based solving techniques. Despite this we can still see improvements in a few problems implying that there are transformations done in *simplify* not detected by the internal preprocessing of *clasp*.

6 Conclusions

We presented an alternative approach to simplifying logic programs with a *smodels* specific rulebase, in comparison to the work done on *clasp*. On the basis of conducted experiments we have shown the effectiveness and usefullness of listed transformations on real world problems. Our tool *simplify*, implementing the presented simplifications, is the first look at external independent ASP-specific optimising preprocessors. With further development and experimentation in combination with other tools it can contribute to positive results outside the scope of this document.

References

[1] A. Eirola. *simplify*. `http://www.tcs.hut.fi/~aeirola/simplify.tar.gz`.

[2] T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying logic programs under uniform and strong equivalence. In *LPNMR*, pages 87–99, 2004.

[3] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. *lasp* : A conflict-driven answer set solver. In *LPNMR*, pages 260–265, 2007.

[4] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Advanced preprocessing for answer set solving. In *ECAI*, pages 15–19, 2008.

[5] M. Gebser, L. Liu, G. Namasivayam, A. Neumann, T. Schaub, and M. Truszczynski. The first answer set programming system competition. In *LPNMR*, pages 3–17, 2007.

[6] V. W. Marek and M. Truszczynski. Stable models and an alternative logic programming paradigm. *CoRR*, 1998.

[7] P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002.

[8] T. Syrjänen. *Lparse 1.0 User's Manual*. Available form the *smodels* website `http://www.tcs.hut.fi/Software/smodels/lparse.ps`.