Extending the Stable Model Semantics with More Expressive Rules

Patrik Simons*
Department of Computer Science and Engineering
Helsinki University of Technology, FIN-02015 HUT, Finland
Patrik.Simons@hut.fi, Tel/Fax +358-9-4513258/4513369

Abstract

The rules associated with propositional logic programs and the stable model semantics are not expressive enough to let one write concise programs. This problem is alleviated by introducing some new types of propositional rules. Together with a decision procedure that has been used as a base for an efficient implementation, the new rules supplant the standard ones in practical applications of the stable model semantics.

Keywords: stable model semantics, constraint satisfaction, implementation.

1 Introduction

Logic programming with the stable model semantics has emerged as a viable method for solving constraint satisfaction problems [4, 5]. The state-of-the-art system *smodels* [6] can often handle non-stratified programs with tens of thousands of rules. However, propositional logic programs can not compactly encode several types of constraints. For example, expressing the subsets of size k of an n-sized set as stable models requires on the order of nk rules. In order to remedy this problem, we improve upon the techniques of *smodels*, by extending the semantics with some new types of propositional rules:

- choice rules for encoding subsets of a set,
- constraint rules for enforcing cardinality limits on the subsets, and
- weight rules for writing inequalities over weighted linear sums.

*The financial support of the Academy of Finland and the Helsinki Graduate School on Computer Science and Engineering is gratefully acknowledged.

The extended semantics is not based on subset-minimal models as is the case for disjunctive logic programs. For instance, the choice rule is more of a generalization of the disjunctive rule of the possible model semantics [7].

A system that computes the stable models of programs containing the new rules has been implemented [9], and it has successfully been applied to deadlock and reachability problems in a class of Petri nets [3]. Other problem domains, such as planning and configuration, will benefit by the improved rules as well. The system is based on *smodels* 1.10 from which it evolved.

The new rules and the stable model semantics are introduced in Section 2. A decision procedure for the extended syntax is presented in Section 3, and some important implementation details are described in Section 4. Experimental results are found in Section 5.

2 The Stable Model Semantics

Let *Atoms* be a set of primitive propositions, or atoms, and consider logic programs consisting of rules of the form

$$h \leftarrow a_1, \ldots, a_n, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_m,$$

where the head h and the atoms $a_1, \ldots, a_n, b_1, \ldots, b_m$ in the body are members of *Atoms*. Call the expression *not* b a not-atom — atoms and not-atoms are referred to as literals.

The stable model semantics for a logic program P is defined as follows [2]. The reduct P^A of P with respect to the set of atoms A is obtained by

1. deleting each rule in P that has a not-atom *not* x in its body such that $x \in A$, and by
2. deleting all not-atoms in the remaining rules.

Definition 2.1. A set of atoms S is a stable model of P if and only if S is the deductive closure of P^S when the rules in P^S are seen as inference rules.

In order to facilitate the definition of more general forms of rules, we introduce an equivalent characterization of the stable model semantics.

Proposition 2.2. *We say that* $f_P : 2^{Atoms} \rightarrow 2^{Atoms}$ *is a closure if*

$$f_P(S) = \{h \mid h \leftarrow a_1, \ldots, a_n, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_m \in P, \\ a_1, \ldots, a_n \in f_P(S),\ b_1, \ldots, b_m \notin S\}.$$

Let $g_P(S) = \bigcap\{f_P(S) \mid f_P : 2^{Atoms} \to 2^{Atoms}$ *is a closure*$\}$. *Then,* S *is a stable model of the program* P *if and only if* $S = g_P(S)$.

Proof. Note that the deductive closure of the reduct P^S is a closure, and note that for every f_P that is a closure, the deductive closure of P^S is a subset of $f_P(S)$. □

A stable model is therefore a model that follows from itself by means of the smallest possible closure. In other words, a stable model is a ground model, and this is the essence of the semantics. The alternative characterization lets us define new rules as consistency demands on the models while guaranteeing that the resulting models are ground.

Definition 2.3. A basic rule r is of the form

$$h \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

and is interpreted by the function $f_r : 2^{Atoms} \times 2^{Atoms} \to 2^{Atoms}$ as follows.

$$f_r(S, C) = \{h \mid a_1, \ldots, a_n \in C,\ b_1, \ldots, b_m \notin S\}.$$

The function f_r produces the result of a deductive step when applied to a candidate stable model S and its consequences C.

Definition 2.4. A constraint rule r is of the form

$$h \leftarrow k\,\{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m\}$$

and is interpreted by

$$f_r(S, C) = \left\{h \,\middle|\, |\{a_1, \ldots, a_n\} \cap C| + |\{b_1, \ldots, b_m\} - S| \geq k\right\}.$$

A choice rule r is of the form

$$\{h_1, \ldots, h_k\} \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

and is interpreted by

$$f_r(S, C) = \left\{h \,\middle|\, h \in \{h_1, \ldots, h_k\} \cap S, a_1, \ldots, a_n \in C,\ b_1, \ldots, b_m \notin S\right\}.$$

Finally, a weight rule r is of the form

$$h \leftarrow \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, not\ b_1 = w_{b_1}, \ldots, not\ b_m = w_{b_m}\} \geq w,$$

for $w_{a_i}, w_{b_i} \geq 0$, and is interpreted by

$$f_r(S, C) = \{h \mid \sum_{a_i \in C} w_{a_i} + \sum_{b_i \notin S} w_{b_i} \geq w\}.$$

Definition 2.5. Let P be a set of rules. As before we say that $f_P : 2^{Atoms} \to 2^{Atoms}$ is a closure if $f_P(S) = \bigcup_{r \in P} f_r\big(S, f_P(S)\big)$, and we define

$$g_P(S) = \bigcap\{f_P(S) \mid f_P : 2^{Atoms} \to 2^{Atoms} \text{ is a closure}\}.$$

Then, S is a stable model of the program P if and only if $S = g_P(S)$.

The motivation for defining constraint, choice, and weight rules is that they can be easily and efficiently implemented and that they are quite expressive. For example, the constraint rule $h \leftarrow k\,\{a_1, \ldots, a_n, \mathit{not}\ b_1, \ldots, \mathit{not}\ b_m\}$ replaces the program

$$\{h \leftarrow a_{i_1}, \ldots, a_{i_{k_1}}, \mathit{not}\ b_{j_1}, \ldots, \mathit{not}\ b_{j_{k_2}} \mid k_1 + k_2 = k, \\ 1 \leq i_1 < \cdots < i_{k_1} \leq n,\ 1 \leq j_1 < \cdots < j_{k_2} \leq m\},$$

which contains $\binom{n+m}{k}$ rules.

Thus, a constraint rule guarantees that if the sum of the number of atoms in its body that are in a stable model and the number of not-atoms in its body that are not is at least k, then the head is in the model. Similarly, if the body of a choice rule agrees with a stable model, then the rule motivates the inclusion of any number of atoms from its head. A weight rule

$$h \leftarrow \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, \mathit{not}\ b_1 = w_{b_1}, \ldots, \mathit{not}\ b_m = w_{b_m}\} \geq w,$$

in turn, will force the head to be a member of a stable model S if $\sum_{a_i \in S} w_{a_i} + \sum_{b_i \notin S} w_{b_i} \geq w$.

Example 2.6. The stable models of the program

$$\{a_1, \ldots, a_n\} \leftarrow \\ \mathit{false} \leftarrow \{a_1 = w_1, \ldots, a_n = w_n\} \geq w \\ \mathit{true} \leftarrow \{a_1 = v_1, \ldots, a_n = v_n\} \geq v$$

containing the atom *true* but not the atom *false* correspond to the ways one can pack a subset of $a_1, \ldots, a_n$ in a bin such that the total weight is less than w and the total value is at least v. The individual weights and values of the items are given by respectively $w_1, \ldots, w_n$ and $v_1, \ldots, v_n$.

Example 2.7. Each of the 16 stable models of the program

$$\{a_1, a_2, a_3, a_4, a_5\} \leftarrow \\ \mathit{false} \leftarrow 3\,\{a_1, a_2, a_3, a_4, a_5\}$$

that does not contain *false* consists of at most two atoms.

3 The Decision Procedure

For an atom a, let $not\,(a) = not\ a$, and for a not-atom $not\ a$, let $not\,(not\ a) = a$. For a set of literals A, define

$$not\,(A) = \{not\,(a) \mid a \in A\}.$$

Let $A^+ = \{a \in Atoms \mid a \in A\}$ and let $A^- = \{a \in Atoms \mid not\ a \in A\}$. Define $Atoms(A) = A^+ \cup A^-$, and for a program P, define $Atoms(P) = Atoms(L)$, where L is the set of literals that appear in the program.

A set of literals A is said to cover a set of atoms B if $B \subseteq Atoms(A)$, and B is said to agree with A if $A^+ \subseteq B$ and $A^- \subseteq Atoms - B$.

Algorithm 1 displays a decision procedure for the stable model semantics. The function $smodels(P, A)$ returns true whenever there is a stable model of P agreeing with A, and it relies on the three functions $expand(P, A)$, $conflict(P, A)$, and $lookahead(P, A)$.

Let $A' = expand(P, A)$. We assume that

E1 $A \subseteq A'$ and that

E2 every stable model of P that agrees with A also agrees with A'.

Moreover, we assume that the function $conflict(P, A)$ satisfies the two conditions

C1 if A covers $Atoms(P)$ and there is no stable model that agrees with A, then $conflict(P, A)$ returns true, and

C2 if $conflict(P, A)$ returns true, then there is no stable model of P that agrees with A.

In addition, $lookahead(P, A)$ is expected to return literals not covered by A.

Theorem 3.1. *Let P be a set of rules and let A be a set of literals. Then, there is a stable model of P agreeing with A if and only if $smodels(P, A)$ returns true.*

Let S be a stable model of P agreeing with the set of literals A. Then, $f_r(S, S) \subseteq S$ for $r \in P$, and we make the following observations. Let

$$min_r(A) = \bigcap_{\substack{A^+ \subseteq C \\ A^- \cap C = \emptyset}} f_r(C, C)$$

Algorithm 1 A decision procedure for the stable model semantics

function $smodels(P, A)$
  $A' := expand(P, A)$
  if $conflict(P, A')$ **then**
    return false
  else if A' covers $Atoms(P)$ **then**
    return true $\{A'^{+}$ is a stable model$\}$
  else
    $x := lookahead(P, A')$
    if $smodels(P, A' \cup \{x\})$ **then**
      return true
    else
      return $smodels\big(P, A' \cup \{not\,(x)\}\big)$.

function $expand(P, A)$
  repeat
    $A' := A$
    $A := Atleast(P, A)$
    $A := A \cup \{not\ x \mid x \in Atoms(P)$ and $x \notin Atmost(P, A)\}$
  until $A = A'$
  return A.

function $conflict(P, A)$
  {Precondition: $A = expand(P, A)$}
  if $A^+ \cap A^- \neq \emptyset$ **then**
    return true
  else
    return false.

function $lookahead(P, A)$
  $B := Atoms(P) - Atoms(A)$; $B := B \cup not\,(B)$
  while $B \neq \emptyset$ **do**
    Take any literal $x \in B$
    $A' := expand(P, A \cup \{x\})$
    if $conflict(P, A')$ **then**
      return x
    else
      $B := B - A'$
  return $heuristic(P, A)$.

be the inevitable consequences of A, and let

$$max_r(A) = \bigcup_{\substack{A^+ \subseteq C \\ A^- \cap C = \emptyset}} f_r(C, C)$$

be the possible consequences of A. Then,

1. for all $r \in P$, S agrees with $min_r(A)$,
2. if there is an atom a such that for all $r \in P$, $a \notin max_r(A)$, then S agrees with $\{not\ a\}$,
3. if the atom $a \in A$, if there is only one $r \in P$ for which $a \in max_r(A)$, and if there exists a literal x such that $a \notin max_r(A \cup \{x\})$, then S agrees with $\{not\,(x)\}$, and
4. if $not\ a \in A$ and if there exists a literal x such that for some $r \in P$, $a \in min_r(A \cup \{x\})$, then S agrees with $\{not\,(x)\}$.

The four statements help us deduce additional literals that are in agreement with S. Define $Atleast(P, A)$ as the smallest set of literals containing A that can not be enlarged using 1–4 above, i.e., let $Atleast(P, A)$ be the least fixed point of the operator

$$\begin{aligned} f(B) = A \cup B \cup \{a \in min_r(B) \mid r \in P\} \\ \cup \{not\ a \mid a \in Atoms(P) \text{ and for all } r \in P,\ a \notin max_r(B)\} \\ \cup \big\{not\,(x) \,\big|\, \text{there exists } a \in B \text{ such that } a \in max_r(B) \\ \text{for only one } r \in P \text{ and } a \notin max_r(B \cup \{x\})\big\} \\ \cup \big\{not\,(x) \,\big|\, \text{there exists } not\ a \in B \text{ and } r \in P \text{ such that} \\ a \in min_r(B \cup \{x\})\big\}. \end{aligned}$$

We conclude,

Proposition 3.2. *If the stable model S of P agrees with A, then S agrees with $Atleast(P, A)$.*

Furthermore, we can bound the stable models from above.

Proposition 3.3. *For a choice rule r of the form*

$$\{h_1, \ldots, h_k\} \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m,$$

let $f'_r(S,C) = \big\{h \in \{h_1,\ldots,h_k\} \,\big|\, a_1,\ldots,a_n \in C,\ b_1,\ldots,b_m \notin S\big\}$*, and for any other type of rule, let* $f'_r(S,C) = f_r(S,C)$*. Let* S *be a stable model of* P *that agrees with* A*. Define* $Atmost(P,A)$ *as the least fixed point of* $f'(B) = \bigcup_{r\in P} f'_r(A^+, B - A^-) - A^-$*. Then,* $S \subseteq Atmost(P,A)$*.*

It follows that $expand(P,A)$ satisfies the conditions E1 and E2. The function $conflict(P,A)$ obviously fulfills C2, and the next proposition shows that also C1 holds.

Proposition 3.4. *If* $A = expand(P,A)$ *covers the set* $Atoms(P)$ *and* $A^+ \cap A^- = \emptyset$*, then* A^+ *is a stable model of* P*.*

3.1 Looking Ahead and the Heuristic

Besides $Atleast(P,A)$ and $Atmost(P,A)$, there is a third way to prune the search space. If the stable model S agrees with A but not with $A \cup \{x\}$ for some literal x, then S agrees with $A \cup \{not\,(x)\}$. One can therefore avoid futile choices if one looks ahead and tests whether $A \cup \{x\}$ gives rise to a conflict for some literal x. Since $x' \in expand(P, A \cup \{x\})$ implies

$$expand(P, A \cup \{x'\}) \subseteq expand(P, A \cup \{x\})$$

due to the monotonicity of $Atleast(P,A)$ and $Atmost(P,A)$, it is not even necessary to examine all literals not covered by A, see $lookahead(P,A)$.

When looking ahead fails to find a literal that causes a conflict, one falls back on a heuristic. For a literal x, let $A_p = expand(P, A \cup \{x\})$ and $A_n = expand\big(P, A \cup \{not\,(x)\}\big)$. Assume that the search space is a full binary tree of height H, and let $p = |A_p - A|$ and $n = |A_n - A|$. Then,

$$2^{H-p} + 2^{H-n} = 2^H \frac{2^n + 2^p}{2^{p+n}}$$

is an upper bound on the size of the remaining search space. Minimizing this number is equal to minimizing

$$\log \frac{2^n + 2^p}{2^{p+n}} = \log(2^n + 2^p) - (p+n).$$

Since $2^{\max(n,p)} < 2^n + 2^p \leq 2^{\max(n,p)+1}$ is equivalent to $\max(n,p) < \log(2^n + 2^p) \leq \max(n,p) + 1$ and

$$-\min(n,p) < \log(2^n + 2^p) - (p+n) \leq 1 - \min(n,p),$$

it suffices to maximize $\min(n,p)$. If two different literals have equal minimums, then one chooses the one with the greater maximum, $\max(n,p)$.

4 Implementation Details

The deductive closures $Atleast(P, A)$ and $Atmost(P, A)$ can both be implemented using two versions of a linear time algorithm of Dowling and Gallier [1]. The basic algorithm associates with each rule a counter that keeps track of how many literals in the body of a rule are not included in a partially computed closure. If a counter reaches zero, then the head of the corresponding rule is included in the closure. From the inclusion follows changes in other counters, and in this manner is membership in the closure propagated.

We begin with basic rules of the form $h \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$. For every rule r we create a literal counter $r.literal$, which is used as above, and an inactivity counter $r.inactive$. If the set A is a partial closure, then the inactivity counter records the number of literals in the body of r that are in $not\,(A)$. The counter $r.inactive$ is therefore positive, and the rule r is inactive, if one can not now nor later use r to deduce its head. For every atom a we create a head counter $a.head$ that holds the number of active rules with head a.

Recall that a literal can be brought into $Atleast(P, A)$ in four different ways. We handle the four cases with the help of the three counters.

1. If $r.literal$ reaches zero, then the head of r is added to the closure.

2. If $a.head$ reaches zero, then $not\ a$ is added to the closure.

3. If $a.head$ is equal to one and a is in the closure, then every literal in the body of the only active rule with head a is added to the closure.

4. Finally, if a is the head of r, if $not\ a$ is in the closure, and if $r.literal = 1$ and $r.inactive = 0$, then there is precisely one literal x in the body of r that is not in the closure, and $not\,(x)$ is added to the closure.

Constraint rules and choice rules are easily incorporated into the same framework. Specifically, one does not use the first case together with choice rules, and one does not compare the literal and inactivity counters of a constraint rule $h \leftarrow k\,\{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m\}$ with zero but with $m+n-k$. A weight rule $h \leftarrow \{a_1 = w_{a_1}, \ldots, a_n = w_{a_n}, not\ b_1 = w_{b_1}, \ldots, not\ b_m = w_{b_m}\} \geq w$, is managed using the upper and lower bound of the sum of the weights in its body. Given a set of literals A, the lower bound is $\sum_{a_i \in A^+} w_{a_i} + \sum_{b_i \in A^-} w_{b_i}$ and the upper bound is $\sum_{a_i \notin A^-} w_{a_i} + \sum_{b_i \notin A^+} w_{b_i}$.

If the upper bound is less than w, then the rule is inactive, and if the lower bound is at least w, then the head is in the closure.

Notice that the implementation provides for incremental updates to the closure $Atleast(P, A)$ as A changes. This is crucial for achieving a high performance.

Since the function $Atmost(P, A)$ is anti-monotonic, it will shrink as A grows. It is no good computing $Atmost(P, A)$ anew each time A is modified. Instead all atoms that might not be in the newer and smaller closure are found using a variant of the basic algorithm. By inspecting these atoms it is possible to decide which ones must be in the closure, and then the basic algorithm can again be used to compute the final closure. A small example will make the method clear.

Example 4.1. Suppose P is the program $\{a \leftarrow b,\ b \leftarrow a,\ a \leftarrow \mathit{not}\ c,\ a \leftarrow \mathit{not}\ d\}$, and suppose A has changed from the empty set to $\{d\}$. Then, we have already computed $Atmost(P, \emptyset) = \{a, b\}$, and we want to find $Atmost(P, A)$. If r is the rule $a \leftarrow \mathit{not}\ d$, then the counter of r is at first zero and then changes to one as d becomes a member of A. Therefore, we deduce that a is possibly not a part of the new closure. The basic algorithm proceeds to increment the counters of $b \leftarrow a$, removing b, and $a \leftarrow b$, where it stops. At this point the counter of the rule $a \leftarrow \mathit{not}\ c$ is still zero, and we note that a must be part of the closure. Including a causes the counter of $b \leftarrow a$ to decrease to zero. Consequently, b is added to the closure and the counter of $a \leftarrow b$ is decremented. Since nothing more remains to be done, the final closure is $\{a, b\}$.

One can argue, in this particular example, that a follows from the rule $a \leftarrow \mathit{not}\ c$ and need not be removed in the first stage of the procedure. However, in general it is not possible to decide whether an atom is in the final closure by inspecting the rules of which it is a head. Notwithstanding, we can make improvements based upon this observation.

For every atom a, create a source pointer whose mission is to point to the first rule that causes a to be included in the closure. During the portion of the computation when atoms are removed from the closure, we only remove atoms which are to be removed due to a rule in a source pointer. For if the rule in a source pointer does not justify the removal of an atom, then the atom is reentered into the closure in the second phase of the computation. In practice, this simple trick yields a substantial speedup of the computation of $Atmost(P, A)$.

5 Experiments

We will search for sets of binary words of length n such that the Hamming distance between any two words is at least d. The size of the largest of these sets is denoted by $A(n, d)$. For example, $A(5, 3) = 4$ and any 5-bit one-error-correcting code contains at most 4 words. One such code is $\{00000, 00111, 11001, 11110\} = \{0, 7, 25, 30\}$. Finding codes becomes very quickly very hard. For instance, $A(10, 3) = 72$ was only recently proved [10].

Construct a program that includes a rule $w_i \leftarrow \mathit{not}\ w_{j_1}, \ldots, \mathit{not}\ w_{j_k}$ for every word $i = 0, \ldots, 2^n$ such that $j_1, \ldots, j_k$ are the words whose distance to i is positive and less than d. Then, the stable models of the program are the maximal codes with Hamming distance d. Add the rule $\mathit{true} \leftarrow m\,\{w_0, \ldots, w_{2^n}\}$ and every model containing *true* is a code of size at least m. For the purpose of making the problem a bit more tractable, we only consider codes that include the zero word.

The test results are tabulated below. The minimum, maximum, and average times are given in seconds and are calculated from ten runs on randomly shuffled instances of the program. All tests where run under Linux 2.2.6 on a 233MHz Pentium II with 128MB of memory.

Problem	Min	Max	Average
$A(6,3) \geq 8$	0.02	0.04	0.03
$A(6,3) < 9$	0.16	0.18	0.17
$A(7,3) \geq 16$	0.14	14.19	6.77
$A(7,3) < 17$	69.08	72.29	70.55
$A(8,3) \geq 20$	6.39	202.41	55.98
$A(8,3) < 21$	> 1 week		

Problem	Min	Max	Average
$A(7,5) \geq 2$	0.05	0.07	0.06
$A(7,5) < 3$	0.04	0.07	0.06
$A(8,5) \geq 4$	0.29	0.36	0.34
$A(8,5) < 5$	2.64	2.75	2.71
$A(9,5) \geq 6$	3.18	8.71	4.81
$A(9,5) < 7$	1127.03	1162.10	1145.85

6 Conclusion

We have presented some new and more expressive propositional rules for the stable model semantics. A decision procedure, which has been used as a base for an efficient implementation, has also been described.

The literals that $\mathit{smodels}(P, A)$ can branch on are, in this paper, the literals that do not cover $\mathit{Atoms}(P) - \mathit{Atoms}(A)$. In previous work, for instance in Niemelä and Simons [6, 8], the eligible literals have also been required to appear in the form of not-atoms in the program. This additional restriction can reduce the search space, and a similar requirement is, of course, also

possible here. The question of which literals one necessarily must consider as branch points is left to future research.

References

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

[2] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.

[3] K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *Lecture Notes in Computer Science*, pages 240–254, Amsterdam, The Netherlands, March 1999. Springer-Verlag.

[4] V.W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. The Computing Research Repository, http://xxx.lanl.gov/archive/cs/, September 1998. cs.LO/9809032.

[5] I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. In *Proceedings of the Workshop on Computational Aspects of Nonmonotonic Reasoning*, pages 72–79. Research Report A52, Helsinki University of Technology, May 1998.

[6] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[7] C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning*, 13:145–172, 1994.

[8] P. Simons. Towards constraint satisfaction through logic programs and the stable model semantics. Research Report A47, Helsinki University of Technology, August 1997.

[9] P. Simons. Smodels 2.10. http://www.tcs.hut.fi/pub/smodels/, 1999. A system for computing the stable models of logic programs.

[10] P. Östergård, T. Baicheva, and E. Kolev. Optimal binary one-error-correcting codes of length 10 have 72 codewords. *IEEE Transactions on Information Theory*, 45(4):1229–1231, May 1999.