Experiments in Answer Sets Planning (extended abstract)

M. Balduccini, G. Brignoli, G.A. Lanzarone, F. Magni and A. Provetti *

Centro di ricerca "Informatica Interattiva" - Università degli studi dell'Insubria a Varese, Italy.
Dipartimento di Scienze dell'Informazione - Università degli studi di Milano, Italy.

Abstract The study of formal nonmonotonic reasoning has been motivated to a large degree by the need to solve the frame problem and other problems related to representing actions. New efficient implementations of nonmonotonic reasoning, such as SMODELSand DLV, can be used to solve many computational problems that involve actions, including plan generation. SMODELSand its competitors are essential to implement a new approach to knowledge representation and reasoning: to compute solutions to a problem by computing the stable models (answer sets) of the theory that represents it. Marek and Truszczyński call this paradigm Stable model programming. We are trying to assess the viability of stable logic programming for agent specification and planning in realistic scenarios. To do so, we present an encoding of plan generation within the lines of Lifschitz's Answer set planning and evaluate its performance in the simple scenario of Blocks world. Several optimization techniques stemming from mainstream as well as satisfiability planning are added to our planner, and their impact is discussed.

1 Introduction

Stable Logic Programming (SLP) is an emergent, alternative style of logic programming: each solution to a problem is represented by an answer set of a function–free logic program[1] encoding the problem itself. Several implementations now exist for stable logic programming, and their performance is rapidly improving; among them are SMODELS [Sim97,NieSim98], DeReS [ChoMarTru96], SLG [CheWar96], DLV [ELMPS97], and CCalc [McCTur97].

Recently, Lifschitz has introduced [Lif99] the term Answer set planning to describe which axioms are needed to characterize correct plans and to discuss the applicability of stable model (answer set) computation interpreters to plan generation. Two features of Answer set planning are particularly attractive.

First, the language it uses is very apt to capture planning instances where conventional STRIPS fall short. In fact, Answer set planning allow the user to specify incomplete knowledge about the initial situation, actions with conditional and indirect effects, nondeterministic actions, and interaction between concurrently executed actions.

Second, even tough Answer set planning specifications, like those in this paper, are completely declarative in nature, SMODELSis by now efficient enough to interpret them, i.e., generate valid plans, with times at least comparable to those of on-purpose planners, on which research and optimization has been active for

* Corresponding author, ph: +39-02-55006.290, e-mail: *provetti@dsi.unimi.it*.

[1] Or, via a syntactic transformation, a restricted default theory or even a $DATALOG^{\neg}$ program.

years. For instance, Dimopoulos et al. [DNK97] report that on the *logistics* domains from Kautz and Selman [KauSel96] (which is about package delivery by using trucks and planes) their SMODELSsolution took 18 seconds vis-a-vis with more than 6 hours for GRAPHPLAN.

We would like to develop on Lifschitz's proposal in several directions which are related to, and bring together, our current research in nonmonotonic reasoning and autonomous agents.

Research in Milan [Cos95,BCP99,CosPro99], has so far contributed to stable logic programming by analyzing, in particular, the dependence of stable models existence on the syntax of the program. This brought out [Cos95] several basic results that extend the usual sufficent conditions of [Dun92,Fag94], none of which indeed applies to Answer set planning, as well as suggesting a new stable model computation algorithm, proposed in [BCP99].

Meanwhile, research in Varese has discussed the prospect of applying logic programming to the complex tasks of designing and *animating* an autonomous agent capable of reaction, planning and above all learning an unknown outside environment. A PROLOG program simulating a learning agent and its environment was proposed in [BalLan97]. Now, a physical scenario is being considered, with the goal of implementing a controller for a small mobile robot equipped with sensors and an arm. For a *principled* knowledge representation approach to succeed there, we need a viable computational mechanism, and this paper is the first of a series of experiments towards assessing stable models computation as a valid choice.

This article discusses a set of experiments on the problem of plan generation in a blocks world domain. This is a simple, familiar problem from the literature, which is often taken as a benchmark for comparing planners. Planning in domains amenable of, basically, a propositional fluents encoding has a natural representation in terms of stable models computations. Some of the reasons are:

- encodings are rather concise and easy to understand;
- the cost of generating a plan grows rapidly with the number of blocks considered and
- the type of planning needed for our autonomous robot is not too distant from that considered here.

Our experiments have been carried out using SMODELS , one successful implementation of stable models programming. For our project, the appealing feature of SMODELSover its competitors is the companion grounding program LPARSE, which accepts as input programs with variables, [some types of]functions symbols and constraints, intended as notation shorthands[2].

[2] Technically, this means that the Herbrand universe of the programs remains finite; practically, it is described as a preprocessing phase, carried out by LPARSE that removes functional terms in favor of internally-constructed constants.

2 Background definitions

The answer sets semantics [GelLif88,GelLif91] is a view of logic programs as sets of inference rules (more precisely, default inference rules), where a stable model is a set of literals closed under the program itself. Alternatively, one can see a program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. Consider the simple program $\{q \leftarrow not\, p, not\, c.\;\; p \leftarrow not\, q.\; p \leftarrow c.\}$. For instance, the first *rule* is read as "assuming that both p and c are false, we can *conclude* that q is true." This program has two answer sets. In the first, q is true while p and c are false; in the second, p is true while q and c are false. When all literals are positive, we speak in terms of *stable models.* In this paper we consider, essentially, the language $DATALOG^{\neg}$ for deductive databases, which is more restricted than traditional logic programming. As discussed in [MarTru99], this restriction is not a limitation at this stage.

A rule ρ is defined as usual, and can be seen as composed of a conclusion $head(\rho)$, and a set of conditions $body(\rho)$, the latter divided into positive conditions $pos(\rho)$ and negative conditions $neg(\rho)$. Please refer to [AptBol94] for a thorough presentation of the syntax and semantics of logic programs. For the sake of clarity however, let us report the definition of stable models. We start from the subclass of positive programs, i.e. those where, for every rule ρ, $neg(\rho) = \emptyset$.

Definition 1. *(Stable model of positive programs)*
The stable model $a(\Pi)$ *of a positive program* Π *is the smallest subset of* $\mathbb{B}_{\Pi}$ *such that for any rule* $a \leftarrow a_1, \ldots a_m$ *in* Π: $a_1, \ldots, a_m \in a(\Pi) \Rightarrow a \in a(\Pi)$.

Positive programs are unambiguous, in that they have a unique stable model, which coincides with that obtained applying other semantics.

Definition 2. *(Stable models of programs)*
Let Π *be a logic program. For any set* S *of atoms, let* Π^S *be a program obtained from* Π *by deleting (i) each rule that has a formula 'not* A*' in its body with* $A \in S$*, and (ii) all formulae of the form 'not* A*' in the bodies of the remaining rules.*

Π^S *does not contain "not," so that its stable model is already defined. If this stable model coincides with* S*, then we say that* S *is a* stable model *of* Π*. In other words, the stable models of* Π *are characterized by the equation:* $S = a(\Pi^S)$.

The answer set semantics is defined similarly by allowing the unary operator $\neg$, called explicit negation, to distinguish it from the classical-logic connective. What changes is that we do not allow any two contrary literals a, $\neg a$ to appear in an answer set.

Gelfond and Lifschitz [GelLif91] show how to *compile away* explicit negations by i) introducing extra atoms $a', b' \ldots$ to denote $\neg a, \neg b, \ldots$ and ii) considering only stable models of the resulting program that contain no contrary pair a, a'. This requirement is captured by adding, for each new atom a', the constraint $\leftarrow a, a'$ to the program. In any case, two-valued interpretations can be forced by adding rules $a \leftarrow not\ a'$ and $a' \leftarrow not\ a.$ for each contrary pair of atoms (resp. literals).

2.1 Consistency conditions

Unlike with other semantics, a program may have no stable model (answer set), i.e., be contradictory, like the following: $\{a \leftarrow not\, b.\ b \leftarrow not\, c.\ c \leftarrow not\, a.\}$, where no set of literals is closed under the rules. Inconsistency may arise, realistically, when programs are combined: if they share atoms, a subprogram like that above may *surface* in the resulting program.

In the literature, the main (sufficient) condition to ensure the existence of stable models is call–consistency [Dun92], which is summarized as follows: no atom *depends* on itself via an odd number of negative conditions. This condition is quite restrictive, e.g., it applies to almost no program for reasoning about actions and planning seen in the literature (see the examples in [Lif99]). Indeed, note how in the translation above the definition of a/a' is not stratified and that the consistency constraint is mapped into rule $false \leftarrow a, a', not\ false$, which is not call-consistent either.

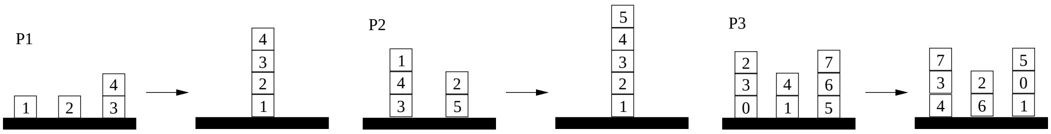
However, some of these authors have shown that this feature does not compromise program's consistency for a large class of cases. The *safe cycle* condition of [CosPro99] applies to all programs considered here.

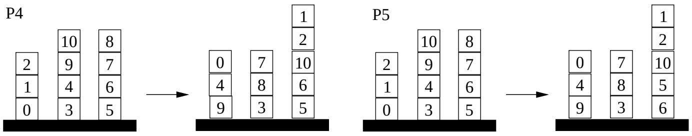
3 Plan specification

The formalization of the Blocks world as a logic program is the main example used by Lifschitz [Lif99] for introducing answer set planning. Two implementations have stem from Lifschitz's definition, Erdem's [Erd99] and the one introduced hereby.

Erdem's solution, which is the closest to Lifschitz's axioms, uses *action and fluent atoms* indexed by time, i.e., *on(B,B1,T)* is a fluent atom, read as "block B is on block B1 at time T" while *move(B,L,T)* is an action, read as "block B is moved on location (meaning another block or the table surface) L at time T."

Our solution is closer to standard situation calculus, since it employs fluent and action terms. In fact, our version of the two atoms presented above is $holds(on(B, B1, T)$ and $occurs(move(B, L), T)$, respectively. Unlike in standard Situation Calculus, we do not have function symbols to denote 'next situations'. This is due in part to earlier limitations of LPARSE, which until recently allowed only functions on integers. However, by having fluents represented as terms, we need only two inertia axioms (see listing below); hence, our solution is more apt than Erdem's for treating complex planning domains, involving hundreds of fluents. The illustrations below are courtesy of W. Faber.





Problem	blocks	steps
P1	4	4
P2	5	6
P3	8	8
P4	11	9
P5	11	11

The price we pay for the convenience of using fluent and action terms is that whereas Erdem's rules can easily be transformed to make them suitable for DLV or CCALC computation, our programs are not easily rephrased for interpreters other than SMODELS. On the other hand, [FabLeoPfe99] have used planning in the Blocks world to experiment with their **dlv** system. However, we felt that this disadvantage was only transient, as DLV and CCALC are actively pursuing the development of their system.

To describe our planner, let us start from the specification of the instance. Basically, we specify the initial and the goal situation[3]. The initial situation here is completely described but it is easy to allow for incomplete knowledge and –if needed– add default assumptions.

```
%%%%%%%%%%%%%%%%%% Domain description: P3
block(b0).
block(b1).
[...]
block(b7).

%%%%%%%% initial situation
holds(on(b2,b3),0).
holds(on(b3,b0),0).
holds(on(b0,table),0).
holds(on(b4,b1),0).
holds(on(b1,table),0).
holds(on(b7,b6),0).
holds(on(b6,b5),0).
holds(on(b5,table),0).

holds(top(b2),0).
holds(top(b4),0).
holds(top(b7),0).
holds(neg(top(b3)),0).
holds(neg(top(b0)),0).
holds(neg(top(b1)),0).
holds(neg(top(b6)),0).
holds(neg(top(b5)),0).
```

[3] The instance below is from [Erd99].

The goal is given in terms of a constraint to be satisfied *at the latest* at time *t=depth,* where *depth* is either passed with the SMODELScall or assigned within the input instance.

```
%%%%%%%% goal state
:-not goal(depth).

goal(T) :- time(T), holds(on(b7,b3),T),
holds(on(b3,b4),T),
holds(on(b4,table),T),
holds(on(b2,b6),T),
holds(on(b6,table),T),
holds(on(b5,b0),T),
holds(on(b0,b1),T),
holds(on(b1,table),T).
```

Let us now see the general axioms for the Blocks world domain (some base predicate definitions are omitted but easy to understand by their use).
The next set of rules describes the [direct] effect of actions.

```
holds(on(B,L),T1) :- next(T,T1),
                     block(B),
                     location(L),
                     occurs(move(B,L),T).

holds(top(B),T1) :- next(T,T1),
                    block(B),
                    location(L),
                    occurs(move(B,L),T).

holds(top(B),T1) :- next(T,T1),
                    block(B), block(B1),
                    holds(on(B1,B),T),
                    location(L),
                    occurs(move(B1,L),T).

holds(neg(top(B)),T1) :- next(T,T1),
                         block(B), block(B1),
                         occurs(move(B1,B),T).
```

The next set of actions provide static constraints, i.e., make no reference to next/past state in order to derive the value of fluents at T (`neq` is the built-in inequality test).

```
holds(neg(on(B,L)),T) :- time(T),
                         block(B),
                         location(L),
                         location(L1),
                         neq(L,L1),
                         holds(on(B,L1),T).

holds(neg(on(table,L)),T) :- time(T),
                             location(L).

holds(top(table),T) :- time(T).
```

The action and fluent description part is ended by the inertia axioms. Note the slight simplification of using semi-normal defaults *in lieu* of abnormalities.

```
    holds(F,T1) :- fluent(F), next(T,T1), holds(F,T), not holds(neg(F),T1).

holds(neg(F),T1) :- fluent(F), next(T,T1), holds(neg(F),T), not holds(F,T1).
```

The following set of rules, called the control module, is crucial for the performance of the planner. It establishes the fact that in each answer set exactly one action is performed at each time $0 \leq T \leq depth - 1$ (no action is performed at the last time). As a consequence, there are in principle $|\mathcal{A}|^{depth}$ stable models of this set of rules (where $|\mathcal{A}|$ denotes the number of possible actions). This is not the case in practice since we have inserted several extra conditions that avoid generating *hopeless* candidate actions.

```
%%%%%%%% Control
occurs(move(B,L),T) :- next(T,T1),
                       block(B),
                       location(L),
                       neq(B,L),        %%% prevents 'moving onto itself'
                       holds(top(B),T), %%% prevents moving a covered block.
                       holds(top(L),T), %%% prevents moving onto an already-occupied bl.
                       not diff_occurs_than(move(B,L),T).

diff_occurs_than(move(B,L),T) :- next(T,T1),
                                 block(B),
                                 location(L),
                                 block(B1),
                                 location(L1),
                                 occurs(move(B1,L1),T),
                                 neq(B,B1).

diff_occurs_than(move(B,L),T) :- next(T,T1),
                                 block(B),
                                 block(B1),
                                 location(L),
                                 location(L1),
                                 occurs(move(B1,L1),T),
                                 neq(L,L1).
```

The rules above are an application of the nondeterministic choice operator by [SacZan97]. Finally, we have some constraints which further guarantee properties of the plan.

```
%%%%%%%%%%% Consistency
:- fluent(F),time(T), holds(F,T),holds(neg(F),T).
%%%%%%% Impossibility laws
:- time(T),block(B),location(L),occurs(move(B,L),T),holds(neg(top(B)),T).
:- time(T),block(B),block(B1),occurs(move(B,B1),T),holds(neg(top(B1)),T).
%%%%%%% can't move on the same block where it's already on
:- time(T),block(B),location(L),occurs(move(B,L),T),holds(on(B,L),T).
```

4 Computational results

As it employs function symbols, the ground version of our planner result much larger than the comparable Erdem's version. As a result, computation time is also longer. The user may or may not want to trade performance for the convenience of using function symbols. The table below reports the timing of finding the minimal plan for each of the Blocks world problems of [Erd99].

Prob.	DEPTH	ERDEM	SITCALC-STYLE
P1	4	0.011	0.100
P2	6	1.480	1.900
P3	8	42.950	1071.300
P4	9	137.560	—
P5	11	—	—

Table1. Times with DEPTH=length of the minimal solution on a Pentium III 500MHz with SunOS 5.7.

4.1 Linearization

Linearization has been proposed in [KauSel96] to improve performance of SAT-based planning by reducing the number of atoms of a given instance, i.e., essentially, its search space. Even tough the search space of SMODELSis defined differently than that of SATPLAN, viz. it corresponds to literal appearing under negation as failure *only* [Sim97], we have proved that it is very effective also in answer set planning.

The idea is to break up the three-parameters predicate $occurs(move(A, B), T$ into two predicates: $move\_obj(B, T)$ and $move\_dest(L, T)$. Let $|B|$ be the number of blocks and $|\mathcal{A}|$ the number of actions. While *occurs* has $|B|^2 \cdot |T| + |B| \cdot |T|$ instances in the normal case, with linearization we consider only $2|B| \cdot |T| + |T|$ atoms overall.

The changes to be made to our planner are concentrated in the control module, listed below. the renaming changes consist in substituting each occurrence of *occurs* (no pun intended) with either *move_obj* or *move_dest* or both.

```
%%%%%%%%%%%% Linearized control module
move_obj(B,T) :- time(T),
block(B),
holds(top(B),T),
not diff_obj(B,T).

diff_obj(B,T) :- time(T),
block(B),
block(B1),
neq(B,B1),
move_obj(B1,T).
```

```
diff_obj(B,T) :- time(T),
block(B),
not move_obj(B,T).

move_dest(L,T) :- time(T),
location(L),
holds(top(L),T),
block(B),      %
move_obj(B,T), % cascading choice
neq(B,L),      %
not diff_dest(L,T).

diff_dest(L,T) :- time(T),
location(L),
location(L1),
neq(L,L1),
move_dest(L1,T).
```

The linearized planner has much appealing performance[4]:

Prob.	DEPTH	LINEAR
P1	4	0.05
P2	6	0.40
P3	8	23.15
P4	9	60.15
P5	11	189.70

Table2. Times with linearized module and *depth* = length of the minimal solution.

What is the trade off here? Linearization makes specifying parallel execution of actions at least awkward [DNK97].

4.2 Overconstraining

Overcostraining is an optimization technique (w.r.t. time) consisting of adding constraint that are logical consequence of the program rules in order to make the search backtrack at earlier points. It has been discussed in [KauSel96] and in the context of SMODELSinterpretation by [DNK97]. It is also present in our first planner: it remains easy to check that, apart from consistency, the constraints are subsumed by the extra conditions in the body of *occurs*.

Even tough overconstraining works for optimization of SMODELSinterpretation, there are still some doubts about whether it can be applied successfully in all cases. In fact, during the experiments with our planner, we noticed that, by adding certain constraints, we would obtain a dramatic performance improvement (about 50%) without altering the semantics of the program. Of course,

[4] From now on results represent the average over 2 runs on a Pentium II 400MHz with 300MB RAM running NetBSD 1.4.1, SMODELS2.24 and LPARSE 0.99.20.

overconstraining could explain it, except that *the constraints are now satisfied regardless.* As an example, let us consider the first additional constraint:

```
:- time(T),block(B),location(L),occurs(move(B,L),T),holds(neg(top(B)),T).
```

Since the predicate occurs/2 is not defined in the (linearized) program anymore, the conjunction is false, independently from the other predicates. This means that the constraint can never be applied. Intuitively, this fact should cause, in the best case, a slight increase in the computation time of the program, due to the larger size of the ground instance. On the contrary, we experienced the dramatic performance improvement shown in Table 3. The constraints to which the results refer to are list below:

```
:- time(T),block(B),location(L),occurs(move(B,L),T),holds(neg(top(B)),T). % Constr. 1

:- time(T),block(B),block(B1),occurs(move(B,B1),T),holds(neg(top(B1)),T). % Constr. 2

:- time(T),block(B),location(L),occurs(move(B,L),T),holds(on(B,L),T). % Constr. 3
```

It is evident from the time results that one constraint is enough to produce the performance improvement. The times of the versions using more than one additional constraint seem to be slightly greater than the one-constraint version, but this issue should be further investigated, since the differences are within experimental error.
We have no convincing explanation for the phenomenon described in this section yet.

test type	file name	LPARSE time	SMODELStime	total time
no constraints	no-occurs-noc.p3	0.70s	36.15s	36.85s
constraint 1	no-occurs-noc2.p3	0.70s	22.50s	23.20s
constraint 2	no-occurs-noc4.p3	0.70s	22.20s	23.90s
constraint 3	no-occurs-noc5.p3	0.70s	22.50s	23.25s
constraints 1 and 2	no-occurs-noc3.p3	0.70s	22.10s	23.80s
all constraints	no-occurs.p3	0.70s	22.45s	23.15s

Table3. Experimental results on P3 with/without additional constraints.

4.3 Improving performance further

Experimental results (and common intuition) show that the performance of SMODELSis, roughly, inversely proportional to the size of the ground instance passed to the interpreter. So, it would be good practice to reduce as much as possible the size of the input program by removing any unnecessary rule/atom. On the other hand, adding constraints may effectively speed up the computation by forcing SMODELSto backtrack at an earlier stage.

We found a good solution to this trade-off which applies to our planner and produces about 10% gain on the computational time of SMODELS. The two constraints achieving this improvement are shown below. Their purpose is two prevent the planner from trying to perform any move after the goal is reached.

```
:- time(T),block(B),goal(T),move_obj(B,T).

:- time(T),location(L),goal(T),move_dest(LT).
```

Suppose that, at time $t_0 < depth$, the planner achieves its goal. Since the definitions of the *move_obj* and *move_dest* predicates do not take into consideration the truth of *goal(T)*, a sequence of useless actions would be generated covering times $t >= t_0$. This approach has several drawbacks. First of all, performing the additional action choices requires computation time, which is, after all, wasted. Second, since the goal was already reached at time t_0, any later action sequence achieves the goal; this means that a large number of models are generated which differ only for the actions performed after reaching the goal.

The set of constraints that we propose simply prevents any action from being performed after the goal has been achieved. The experimental results of the planner with and without the constraints are shown below.

type	file name	LPARSE time	SMODELStime	total time
w/o constraints	no-occurs.p3	0.70s	22.45s	23.15s
with constraints	no-occurs-e.p3	0.70s	20.77s	21.47s

Table4. Running times for P3 with/without constraints on post-goal actions.

However, this solution does not mean that we are able to capture minimal plan generation within stable logic programming. Deciding whether a program has stable models is an NP-complete problem [MarTru99], while generating a minimal plan is in Δ_2^p [Lib99]. All we can hope to achieve, for minimal planning, it to optimize an algorithm that calls SMODELSas an NP-oracle *at most* a logarithmic number of times. See Liberatore's work ([Lib99] and references therein) for a discussion on these crucial aspects.

Acknowledgments

S. Costantini has greatly motivated us to pursue this work. E. Erdem's work set the framework of our experiments and her lively comments have prompted and motivated us. I. Niemelä and P. Simons have graciously helped us on-line several times with their SMODELS implementation.

References

[AptBol94] Apt, K. R. and Bol, R., 1994. *Logic programming and negation: a survey,* J. of Logic Programming, 19/20.

[BalLan97] M. Balduccini and G. A. Lanzarone, 1997. *Autonomous semi–reactive agent design based on incremental inductive learning in logic programming.* Proc. of the ESSLI'97 Symp. on Logical Approaches to Agent Modeling and Design, pages 1–12. Utrecht University, 1997.

[BarGel94] Baral, C. and Gelfond. M., 1994. *Logic programming and knowledge representation,* J. of Logic Programming, 19/20.

[BCP99] Brignoli G., Costantini S. and Provetti A., 1999. *A Graph Coloring algorithm for stable models generation.* Univ. of Milan Technical Report, submitted for publication.

[CosPro99] Costantini S. and Provetti A., 1999. *A new method, and new results, for detecting consistency of knowledge bases under answer sets semantics.* Univ. of Milan Technical Report, submitted for publication.

[Cos95] Costantini S., 1995. *Contributions to the stable model semantics of logic programs with negation,* Theoretical Computer Science, 149.

[CheWar96] Chen W., and Warren D.S., 1996. *Computation of stable models and its integration with logical query processing,* IEEE Trans. on Data and Knowledge Engineering, 8(5):742–747.

[ChoMarTru96] Cholewiński P., Marek W. and Truszczyński M., 1996. *Default reasoning system DeReS.* Proc. of KR96, Morgan-Kauffman, pp. 518-528.

[DNK97] Dimopoulos Y., Nebel B. and Koehler J., 1997. *Encoding Planning Problems in Nonmonotonic Logic Programs,* Proc. of European Conference on Planning, pp. 169–181.

[Dun92] Dung P.M., 1992. *On the Relation between Stable and Well–Founded Semantics of Logic Programs,* Theoretical Computer Science, 105.

[ELMPS97] Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F., 1997. *A deductive system for non-monotonic reasoning.* Proc. Of the 4th LPNMR Conference, Springer Verlag, LNCS 1265, pp. 363–374.

[Erd99] Erdem E., 1999. *Application of Logic Programming to Planning: Computational Experiments.* Proc. of LPNMR'99 Conference, LNAI.

[FabLeoPfe99] Faber W., Leone N. and Pfeifer G., 1999. *Pushing Goal Derivation in DLP Computations.* Proc. of LPNMR'99 Conference, LNAI.

[Fag94] Fages F., 1994. *Consistency of Clark's completion and existence of stable models.* Proc. of 5th ILPS conference.

[GelLif88] Gelfond, M. and Lifschitz, V., 1988. *The stable model semantics for logic programming,* Proc. of 5th ILPS conference, pp. 1070–1080.

[GelLif91] M. Gelfond and V. Lifschitz., 1991. *Classical negation in logic programs and disjunctive databases.* New Generation Computing, pp. 365–387.

[KauSel96] Kautz H. and Selman B., 1996. *Pushing the envelope: Planning, Propositional Logic and Stochastic Search* Proc. of AAAI'96.

[Lib99] Liberatore P., 1999. *Algorithms and Experiments on Finding Minimal Models.* Technical Report of University of Rome "La Sapienza."

[Lif99] Lifschitz V., 1999. *Answer Set Planning.* Proc. of LPNMR'99 Conference.

[MarTru99] Marek, W., and Truszczyński M., 1999. *Stable models and an alternative logic programming paradigm.* The Journal of Logic Programming.

[NieSim98] Niemelä I. and Simons P., 1998. *Logic programs with stable model semantics as a constraint programming paradigm.* Proc. of NM'98 workshop. Extended version submitted for publication.

[McCTur97] McCain N. and Turner H., 1997. *Causal theories of actions and change.* Proc. of AAAI'97 Conference, pp. 460–465.

[SacZan97] Saccà D. and Zaniolo C., 1997. *Deterministic and Non-Deterministic Stable Models.* J. of Logic and Computation.

[Sim97] Simons P., 1997. *Towards Constraint Satisfaction through Logic Programs and the Stable Models Semantics,* Helsinki Univ. of Technology R.R. A:47.

[SubNauVag95] Subrahmanian, V.S., Nau D., and Vago C., 1995. *WFS + branch and bound = stable models,* IEEE Trans. on Knowledge and Data Engineering, 7(3):362–377.