Stable Model Checking Made Easy

Christoph Koch and Nicola Leone
Information Systems Dept.
Technical University of Vienna
A-1040 Vienna, Austria
{koch, leone}@dbai.tuwien.ac.at

Abstract

Disjunctive logic programming (DLP) with stable model semantics is a powerful nonmonotonic formalism for knowledge representation and reasoning. Reasoning with DLP is harder than with normal (V-free) logic programs; liecause *stable model checking* - deciding whether a given model is a stable model of a propositional DLP program is co-NP-complete, while it is polynomial for normal logic programs.

This paper proposes a new transformation $\Gamma_M(\mathcal{P})$, which reduces stable model checking to UNSAT - i.e., to deciding whether a given CNF formula is unsatisfiable. Thus, the stability of a model A/ for a program $\mathcal{P}$ *caw* be verified by calling a Satisfiability Checker on the CNF formula $\Gamma_M(\mathcal{P})$. The transformation is parsimonious and efficiently computable, as it runs in logarithmic space. Moreover, the size of the generated CNF formula never exceeds the size of the input.

The proposed approach to stable model checking lias been implemented in a DLP system, and a number of experiments and benchmarks have been run.

1 Introduction

Disjunctive logic programming (DLP) with stable model semantics is a powerful nonmonotonic formalism for knowledge representation and eommonsense reasoning [Baral and Gelfond, 1994; Lobo *et.* a/., 1992]. DLP has a very high expressive power [Eiter *et.* a/., 1997a], and it allows to represent complex problems in a simple and easy-to-understand fashion [Eiter *et.* a/., 1998]. As for the other main nonmonotonic formalisms, reasoning with DLP (under stable model semantics) is very hard. The high complexity of DLP reasoning depends also on the hardness of *stable model cheeking* - deciding whether a. given model is a stable model of a propositional DLP program which is co-NP-complete. The hardness of this problem ha»s discouraged the implementation of DLP engines. Indeed, at the time being only one system, namely the dlv system (Eiter *et.* a/., 1998], is available which fully supports (function-free) DLP with stable model semantics.

This paper proposes a new transformation, which reduces stable model checking to UNSAT i.e., to deciding whether a given CNF formula is unsatisfiable - the complement of Satisfiability, a problem for which very efficient systems have been developed in AI

Besides providing an elegant characterization of stable models, which sheds new light on their intrinsic nature, the proposed transformation has also a strong practical impact. Indeed, by using this transformation, the huge amount of work done in AI on the design and implementation of efficient algorithms for checking Satisfiability can be profitably used for the implementation of DLP engines supporting stable model semantics. Thus, this transformation opens "new frontiers" in the implementation of Disjunctive Logic Programming which can benefit now from efficient AI techniques and implementations.

We have implemented the proposed technique in the DLP system dlv, and we have run a number of experiments and benchmarks.

In sum, the main contributions of this paper are the following:

• We define a new transformation from stable model checking for general DLP with negation to UNSAT of propositional CNF formulas. The transformation is parsimonious (i.e., it does not add any new symbol) and efficiently computable, since it runs in logspace (and, therefore, in polynomial time). Moreover, the size of the generated CNF formula never exceeds the size of the input (it is usually much smaller).

• We realize our approach in the DLP system dlv, by using an implementation of the Davis-Putnam procedure as the Satisfiability checker.

Our implementation exploits also novel modular evaluation techniques, which follow from our main results.

• We carry out an experimental activity which witnesses the efficiency of our approach to stable model checking.

It is worth noting that, since stable model checking of DLP programs generalizes minimal model checking for Horn CNF formulas, our results can be employed also for

"Research supported by FWF (Austrian Science Fund) under the projects P11580-MAT and Z29-INF.

reasoning with minimal models and with circumscription.

The dlv system, which implements the results described in this paper, can be downloaded from www.dbai.tuwien.ac.at/proj/dlv. From the same Web page, one can also retrieve the benchmark problems that we used in our experiments.

The remainder of the paper is organized as follows: Setions 2 and 3 contain preliminary notions on DLP, stable model semantics, and its characterization in terms of unfounded sets. Section *A* describes our new reduction from stable model checking to UNSAT. Section 5 reports on our experiments.

2 Disjunctive Logic Programming with Stable Model Semantics

In this section, we provide an overview of disjunctive logic programming with stable model semantics. (For further details, see [Lobo *et. al,* 1992].)

The terms of the language are inductively defined. A variable or constant is a *term:* a function symbol with terms as arguments is a term. An *atom* is $a(t_1, ..., t_n)$. where a is a *predicate* of arity // and $t_1, ..., t_n$ are terms. A *literal* is either a *positive literal* p or a *negative literal* -p, where p is an atom.

A *(disjunctive) rule* r is a clause of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \cdots \wedge b_k \wedge \neg b_{k+1} \wedge \cdots \wedge \neg b_m \quad n \geq 1,\ m \geq 0$$

where $a_1, \cdots, a_n, b_1, \cdots, b_m$ are atoms. The disjunction $a_1 \vee \cdots \vee a_n$ is the *head* of r, while the conjunction $b_1 \wedge ... \wedge b_k \wedge \neg b_{k+1} \wedge ... \wedge \neg b_m$ is the *body* of r. We denote by $H(r)$ the set $\{a_1, ..., a_n\}$ of the head atoms, and by $B(r)$ the set $\{b_1, ..., b_k, \neg b_{k+1}, ..., \neg b_m\}$ of the body literals. $B^+(r)$ (resp., $B^-(r)$) denotes the set of atoms occurring positively (resp.. negatively) in E(r). A *(disjunctive) program* (also called DLP program) is a set of rules. A $\neg$-free (resp., $\vee$-free) program is called *positive* (resp., *normal).* A term, an atom, a literal, a rule, or a program is *ground* if no variables appear in it. A finite ground program is also called a *prepositional* program. A *function-free,* program is a finite program where no function symbol occurs.

Let $\mathcal{P}$ be a program. The *Herbrand universe* and the *Herbrand base* of $\mathcal{P}$ are defined in the standard way and denoted by *Up* and $B_{\mathcal{P}}$, respectively. Given a rule r occurring in a $\mathcal{P}$, a *ground instance* of r is a rule obtained from r by replacing every variable X in *r* by $\sigma(X)$, where σ is a mapping from the variables occurring in *r* to the terms in *Up.* We denote by *ground*$(\mathcal{P})$ the set of all the ground instances of the rules occurring in $\mathcal{P}$.

An *interpretation* for $\mathcal{P}$ is a set of ground atoms, that is, an interpretation is a subset / of $B_{\mathcal{P}}$. A ground positive literal *A* is *true* (resp., *false)* w.r.t. I if *A* $\in$ / (resp.. $A \notin I$). A ground negative literal -4 is *true* w.r.t. / if *A* is false w.r.t. I otherwise A is false w.r.t. 7.

Let r be a ground rule in *ground{V).* The head ol r is *true* w.r.t. if $H(r) \cap I \neq \emptyset$. The body of r is *true* w.r.t. / if all body literals of r are true w.r.t. / (i.e., $B^+(r) \subseteq I$ and $B^-(r) \cap I \neq \emptyset$) and is *false* w.r.t. / otherwise. The rule r is *satisfied* (or *true)* w.r.t. / if its head is true w.r.t. / or its body is false w.r.t. /.

A *model* for $\mathcal{P}$ is an interpretation *M* for $\mathcal{P}$ such that every rule $r \in$ *ground*$(\mathcal{P})$ is true w.r.t. A/. A model *M* for $\mathcal{P}$ is *minimal* if no model /V for $\mathcal{P}$ exists such that *N* is a proper subset of *M.* The set of all minimal models for $\mathcal{P}$ is denoted by MM(P).

A generally acknowledged semantics for DLP programs is the extension of the stable model semantics to take into account; disjunction [Gelfond and Lifschitz, 1991: Przymusinski, 1991]. Given a program $\mathcal{P}$ and an interpretation /, the *Gelfond-Lifschitz (GL) transformation* of $\mathcal{P}$ w.r.t. I, denoted $\mathcal{P}^I$, is the set of positive rules defined as follows:

$$\begin{aligned}\mathcal{P}^I = \{\ & a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \cdots \wedge b_k \ \mid \\ & a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \cdots \wedge b_k \wedge \neg b_{k+1} \wedge \cdots \wedge \neg b_m \\ & \text{is in ground}(\mathcal{P}) \text{ and } b_i \notin I, \ \text{ for all } k < i \leq m\ \}\end{aligned}$$

Definition 2 1 [Przymusinski, 1991: Gelfond and Lifschitz. 1991] Let / be an interpretation for a program $\mathcal{P}$. I is a *stable model* for $\mathcal{P}$ if $I \in \mathrm{MM}(\mathcal{P}^I)$ (i.e., *1* is a minimal model of the positive program $\mathcal{P}^I$). □

Example 2.2 Let $P = \{a \vee b \leftarrow c;\quad b \leftarrow \neg a \wedge \neg c;\quad a \vee c \leftarrow \neg b\}$ and $I = \{b\}$. Then, $P^I = \{a \vee b \leftarrow c;\quad b \leftarrow\}$.
It is easy to verify that I is a minimal model for P^I; thus, I is a stable model for P. □

Clearly, if $\mathcal{P}$ is positive, then $\mathcal{P}^I$ coincides with *ground,(V).* It turns out that for n positive program, minimal and stable models coincide.

3 Stable Models and Unfounded Sets

Next, we present a characterization of the stable models of disjunctive logic programs in terms of unfounded sets. This characterization will be used to prove the correctness of our reduction from stable model checking to UNSAT in the next section.

The characterization presented here is obtained by slight modifications of the results presented in [Leone *et. al.* 1997]. In particular, providing the notion of unfounded sets directly for total (2-valued) interpretations, we obtain a simpler characterization than in [Leone *et. a/.,* 1997], where unfounded sets were defined w.r.t. partial (3-valued) interpretations.

Definition 3.1 (Definition 3.1 in [Leone *et.* a/., 1997]) Let / be an interpretation for a program $\mathcal{P}$. A set $X \subseteq B_{\mathcal{P}}$ of ground atoms is an *unfounded set* for *V* w.r.t. *1* if. for each rule $r \in$ *ground*$(\mathcal{P})$ such that A" $\cap H(r) \neq \emptyset$, at least one of the following conditions holds:
1. $(B^+(r) \not\subseteq I) \vee (B^-(r) \cap I \neq \emptyset)$, that is the body of r is false w.r.t. /.
2. $B^+(r) \cap X \neq \emptyset$, that is, some positive body literal belongs to X.
3. $(H(r) - X) \cap I \neq \emptyset$, that is, an atom in the head of r, distinct from the elements in X, is true w.r.t. I. □

```
Input: A ground DLP program $\mathcal{P}$ and a model $M$ for $\mathcal{P}$
Output: A propositional CNF formula $\Gamma_M(V)$ over $M$.
var $\mathcal{P}'$: DLP Program; $S$: Set of Clauses;
begin
1.  Delete from $\mathcal{P}$ each rule whose body is false
      w.r.t. M;
2.  Remove all negative literals from the (bodies of the)
      remaining rules;
3.  Remove all false atoms (w.r.t. $M$) from the heads
      of the resulting rules;
4.  $S := \emptyset$;
5.  Let $\mathcal{P}'$ be the program resulting from steps 1–3;
6.  for each rule $a_1 \vee \cdots \vee a_n \leftarrow b_1 \wedge \cdots \wedge b_m$ in $\mathcal{P}'$ do
7.        $S := S \cup \{\ b_1 \vee \cdots \vee b_m \leftarrow a_1 \wedge \cdots \wedge a_n\ \}$;
8.  end for;
9.  $\Gamma_M(\mathcal{P}) := \bigwedge_{c \in S} c \ \wedge\ (\ \bigvee_{x \in M} x\ )$;
10. output $\Gamma_M(\mathcal{P})$
end.
```

Figure 1: *The transformation* Γ_M *(V)*

Definition 3.2 An interpretation / for a program V is unfounded-free iff no nonempty subset of / is an unfounded set for $\mathcal{P}$ w.r.t. /. □

The unfounded-free condition singles out precisely the stable models.

Theorem 3.3 *(Theorem 4.6 in [Leone* et. al., *1997]) Let M be a model for a program $\mathcal{P}$. M is stable model for V iff M is unfounded-free.*

Example 3.4 Let $\mathcal{P} = \{a \vee b \leftarrow\}$. $M_1 = \{a\}$ is a stable model of $\mathcal{P}$, since there is no nonempty subset of M_1 which is an unfounded set. On the other hand, for $M_2 = \{a, b\}$, due to Condition 3, both $\{a\}$ and $\{b\}$ are unfounded sets. M2 is not unfounded-free, and therefore, it is not a stable model. □

4 From Stable Model Checking to UNSAT

In this section we present a reduction from stable model checking to UNSAT, the complement of Satisfiability, a problem which is better explored in AI and for which efficient algorithms and systems are available.

Recall that a CNF formula, over a set A of atomic propositions is a conjunction of the form $\phi = c_1 \wedge \cdots \wedge c_n$, where $c_1, \cdots, c_n$ are clauses over A. Without loss of generality, in this paper a clause $c = a_1 \vee \cdots \vee a_m \vee \neg b_1 \vee \vee \neg b_r$ will be written as $a_1 \vee \cdots \vee a_m \leftarrow b_1 \wedge \cdots \wedge b_r$; thus, a CNF ϕ will be a conjunction of these implications.

A formula ϕ is *satisfiable* if there exists a truth assignment to the propositions of A which makes ϕ true; otherwise, ϕ is *unsatisfiable* (or *inconsistent).*

UNSAT is the following decision problem.

> Given a CNF formula ϕ, is it true that ϕ is unsatisfiable?

Our reduction from stable model checking to UNSAT is implemented by the algorithm shown in Figure 1.

Example 4.1 Let $\mathcal{P} = \{a \vee b \vee c \leftarrow;\quad a \leftarrow b;\quad a \leftarrow c;\quad b \leftarrow a \wedge \neg c\}$. Consider model $M_1 = \{a, b\}$ of $\mathcal{P}$. In the first step of the algorithm shown in Figure 1, the rule $a \leftarrow c$ is deleted. In the second s t e $\neg c$i s removed from the body of the last rule of $\mathcal{P}$; while the third step removes c from the head of the first rule. Thus, after Step 3, the program becomes $\{a \vee b \leftarrow;\quad a \leftarrow b;\quad b \leftarrow a\}$. Steps 4 to 8 switch the bodies and the heads of the rules, yielding the set of clauses $S = \{\leftarrow a \wedge b;\quad b \leftarrow a;\quad a \leftarrow b\}$. Finally, Step 9 constructs the conjunction of the clauses in 5 plus the clause $a \vee b \leftarrow$. Therefore, the output of the algorithm is $(\leftarrow a \wedge b) \bigwedge (b \leftarrow a) \bigwedge (a \leftarrow b) \bigwedge (a \vee b \leftarrow)$.

Now consider model $M_2 = \{a, c\}$. Here, the first three steps simplify $\mathcal{P}$ to $\{a \vee c \leftarrow;\quad a \leftarrow c.\}$. Steps 4 to 8 swap the heads and bodies of the rules to get $\{\leftarrow a \wedge c;\quad c \leftarrow a\}$, and Step 9 adds $a \vee c \leftarrow$. So the outcome for $M2$ is $(\leftarrow a \wedge c) \bigwedge (c \leftarrow a) \bigwedge (a \vee c \leftarrow)$. □

Theorem 4.2 *Given a model M for a ground DLP program $\mathcal{P}$, let TM (V) be the CNF formula computed by the algorithm of Figure 1 on input $\mathcal{P}$ and M. Then, M is a stable model for $\mathcal{P}$ if and only if Γ_M (V) is unsatisfiable.*

In the remainder of this section we demonstrate Theorem 4.2 (i.e., we show the correctness of our $\Gamma_M(\mathcal{P})$ reduction). For space limitation, we do not include the proofs of the two lemmas here; but we illustrate their validity on a running example.

To better illustrate the $\Gamma_M(\mathcal{P})$ transformation, we proceed in an incremental way, dividing the transformation into three steps, and showing the correctness of each of them.

Definition 4.3 Let $\mathcal{P}$ be a ground DLP program and M be n model for $\mathcal{P}$. Define the *simplified version* $\alpha_M(\mathcal{P})$ of $\mathcal{P}$ w.r.t M as:

$$\alpha_M(\mathcal{P}) = \{\ \bigvee_{x \in (H(r) \cap M)} x \leftarrow \bigwedge_{x \in B^+(r)} x \mid r \in \mathcal{P}, \text{ the body of } r \text{ is true w.r.t. M}\ \}$$ □

Thus, $\alpha_M(\mathcal{P})$ coincides with the program $\mathcal{P}'$ obtained by steps 1-3 of Figure 1. Observe that every rule in AM($\mathcal{P}$) has a non-empty head. Indeed, i$\alpha_M(\mathcal{P})$ u l d contain a rule r with an empty head, then M would not be a model for $\mathcal{P}$, as the rule of $\mathcal{P}$ corresponding to r would have a true body and a false head. Moreover, the simplified program $\alpha_M(\mathcal{P})$ is positive (-free) and it only contains atoms that are true w.r.t. M.

Next, we observe that αM('P) is equivalent to $\mathcal{P}$ as far as the stability of M is concerned.

Lemma 4.4 *Let $\mathcal{P}$ be a DLP program and M be a model for $\mathcal{P}$. Then, M is a stable model for $\mathcal{P}$ if and only if it is a stable model for $\alpha_M(\mathcal{P})$.*

Example 4.5 Take V and the two models $M\backslash$ and $M2$ from Example 4.1. $M\backslash$ is a stable model for $\mathcal{P}$„ while $M2$ is not. Indeed, $M\backslash$ is a stable model for $\alpha_{M_1}(\mathcal{P}) = \{a \vee b \leftarrow;\quad a \leftarrow b;\quad b \leftarrow a\}$ and A/2 is not a stable model for $\alpha_{M_2}(\mathcal{P}) = \{a \vee c \leftarrow;\quad a \leftarrow c\}$ □

Next, we show that by simply swapping the heads and bodies of the rules of the simplified program $\alpha_M(\mathcal{P})$, we get a set of clauses whose models correspond to the unfounded sets of $\mathcal{P}$ w.r.t. M.

Definition 4.6 Let $\mathcal{P}$ be a DLP program and M be a model for $\mathcal{P}$. Define $\beta_M(\mathcal{P})$ as the following set of clauses over M:

$$\beta_M(\mathcal{P}) = \{ \bigvee_{x \in B(r)} x \leftarrow \bigwedge_{x \in H(r)} x \mid r \in \alpha_M(\mathcal{P}) \} \quad \square$$

Observe that $\beta_M(\mathcal{P})$ coincides with the set of clauses S constructed after steps 1-8 of Figure 1.

Lemma 4.7 *Let $\mathcal{P}$ be a ground DLP program, M be a model for $\mathcal{P}$, and $X \subseteq M$. Then X is a model for $\beta_M(\mathcal{P})$ iff it is an unfounded set for $\mathcal{P}$ w.r.t. M.*

Example 4.8 $\beta_{M_1}(\mathcal{P})$ is $\{\leftarrow a \wedge b; \quad b \leftarrow a; \quad a \leftarrow b\}$, The only subset of M_1 which is a model for $\beta_{M_1}(\mathcal{P})$ is $\emptyset$. Indeed, $\emptyset$ is the only unfounded set for $\mathcal{P}$ w.r.t. M_1 (contained in M_1).

$\beta_{M_2}(\mathcal{P})$ is equal to $\{\leftarrow a \wedge c; \quad c \leftarrow a\}$. $\emptyset$ and $\{c\}$ are the subsets of M_2 which are models of $\beta_{M_2}(\mathcal{P})$. Indeed, they are precisely the unfounded sets for $\mathcal{P}$ w.r.t. M_2 (contained in M_2). $\square$

We are now in a position to demonstrate our main theorem.

Proof of Theorem 4.2 *(Sketch)* In the following, we show that $\Gamma_M(\mathcal{P})$ is unsatisfiable iff $\mathcal{P}$ is unfounded-free; the statement will then follow from Theorem 3.3.

It is easy to see that the output $\Gamma_M(\mathcal{P})$ of the algorithm of Figure 1 coincides with the conjunction of all clauses in $\beta_M(\mathcal{P})$ and the clause $\vee_{x \in M} x$. From Lemma 4.7, the models of $\beta_M(\mathcal{P})$ are precisely the unfounded sets of $\mathcal{P}$ w.r.t. M. Therefore, the models of $\Gamma_M(\mathcal{P})$ are exactly the non-empty unfounded sets of $\mathcal{P}$ w.r.t. M, since every model of $\Gamma_M(\mathcal{P})$ must satisfy also the clause $\vee_{x \in M} x$. Thus, M contains no *nonempty* unfounded set for $\mathcal{P}$ (i.e., it is unfounded-free) iff $\Gamma_M(\mathcal{P})$ has no model (i.e., it is unsatisfiable). $\square$

Example 4.9 $M_1 = \{a, b\}$ is a stable model for $\mathcal{P}$. Indeed, $\Gamma_{M_1}(\mathcal{P}) = (\leftarrow a \wedge b) \bigwedge (b \leftarrow a) \bigwedge (a \leftarrow b) \bigwedge (a \vee b \leftarrow)$ is unsatisfiable.

Differently, $M_2 = \{a, c\}$ is not stable for $\mathcal{P}$. Indeed, $\Gamma_{M_2}(\mathcal{P}) = (\leftarrow a \wedge c) \bigwedge (c \leftarrow a) \bigwedge (a \vee c \leftarrow)$ is satisfied by model $\{c\}$. $\square$

The next theorem shows that $\Gamma_M(\mathcal{P})$ is also an efficient transformation.

Theorem 4.10 *Given a model M for a ground DLP program $\mathcal{P}$, let $\Gamma_M(\mathcal{P})$ be the CNF formula computed by the algorithm of Figure 1 on input $\mathcal{P}$ and M. Then, the following fiolds.*

1. *$\Gamma_M(\mathcal{P})$ is logspace computable from $\mathcal{P}$ and M.*
2. *$\Gamma_M(\mathcal{P})$ is a parsimonious transformation.*
   $|\Gamma_M(\mathcal{P})| \leq |\mathcal{P}| + |M|$.

Proof *(Sketch)* $\Gamma_M(\mathcal{P})$ is the conjunction of the clauses in $\beta_M(\mathcal{P})$ plus the disjunction of the propositions in M. The size of $\beta_M(\mathcal{P})$ is equal to the size of $\alpha_M(\mathcal{P})$, which is smaller than the size of $\mathcal{P}$. Thus, $|\Gamma_M(\mathcal{P})| \leq |\mathcal{P}| + |M|$.

$\Gamma_M(\mathcal{P})$ is clearly parsimonious, as it is a formula over the propositions of M only.

Finally, it is easy to see that $\Gamma_M(\mathcal{P})$ can be computed by a logspace Turing Machine. Indeed, $\Gamma_M(\mathcal{P})$ can be generated by dealing with one rule of $\mathcal{P}$ at a time, without storing any intermediate data apart from a (fixed) number of indices. $\square$

5 Implementation and Benchmarks

In order to check the concrete usability of our results, we have implemented our approach to stable model checking in the disjunctive logic programming system dlv [Eiter *et. al.*, 1997b; 1998].

dlv is a knowledge representation system which has been developed at Technische Universitat Wien. Recent comparisons [Eiter *et. al.*, 1998] have shown that dlv is nowadays a state-of-the-art implementation of disjunctive logic programming. To our knowledge, dlv is the only publicly available system which supports full (function-free) disjunctive logic programming under stable model semantics.

The computational engine of dlv implements the theoretical results achieved in [Leone *et. al.*, 1997]. Roughly, the system consists of two main modules: the Model Generator (MG) and the Model Checker (MC). MG produces stable model candidates, whose stability is then checked by the MC.

We have replaced the Model Checker of dlv by new modules implementing the results of the previous section, running some benchmark problems and comparing the execution times.

Overview of the Compared Methods

We have compared the following methods for stable model checking (the labels below are used in Figures 2, 3, and 4).

• (Old Checker.) The old model checker of the dlv system [Leone *et. al.*, 1997; Eiter *et. al.*, 1997b]. Its strong points are the efficient evaluation of head cycle free (HCF) programs [Ben-Eliyahu and Dechter, 1994] and the use of modular evaluation techniques. Indeed, HCF programs are evaluated in polynomial time and, if the program is not HCF, the inefficient part of the computation is limited only to the subprograms[1] which are not HCF (while the polynomial time algorithm is applied to the HCF subprograms). Polynomial space and single exponential time bounds are always guaranteed.

• (Γ_M+DP.) Given a program $\mathcal{P}$ and a model M to be checked for stability, an implementation of the algorithm of Figure 1 generates the CNF formula $\Gamma_M(\mathcal{P})$, which is then submitted to a Satisfiability checker. If the Satisfiability checker returns true ($\Gamma_M(\mathcal{P})$ is satisfiable), then M is not a stable model of $\mathcal{P}$; otherwise ($\Gamma_M(\mathcal{P})$ is unsatisfiable), M is a stable model of $\mathcal{P}$. For checking

[1]A subprogram of $\mathcal{P}$ is the set of rules defining the atoms of the same strongly connected component of the dependency graph of $\mathcal{P}$ [Leone *et. al.*, 1997].

Satisfiability of $\Gamma_M(\mathcal{P})$, we have used an efficient implementation of the Davis-Putnam procedure called SATO [Zhang, 1997].

• (Γ_M+DP+Mod.) This is an improved version of the method above, enhanced by modular evaluation techniques derived from the combination of Lemma 4.4 with the modularity results of [Leone *ct. a/.*, 1997]. Roughly, given $\mathcal{P}$ and M, $\mathcal{P}$ is first simplified (steps 1 3 of Figure 1) computing the program $\alpha_M(\mathcal{P})$. The subprograms of αM($\mathcal{P}$) are then evaluated one-at-a-time. A polynomial time method is applied to HCF subprograms (as in *Old Checker):* while the Γ_M+DP method is applied to non-HCF subprograms.

Benchmark Problem

In order to generate some co-NP-hard model cheeking instances, which could better highlight the differences between the model checking techniques, we needed to run Σ_2^P-hard problems on the DLP system at hand. Thus, we have compared the performance of the different model checking methods by running various instances of the Σ_2^P-complete problem *Strategic Companies* on the dlv system.

The strategic companies problem is from [Cadoli e*t. a/.*, 1997]; it is, to the best of our knowledge, the only Σ_2^P-complete KR problem from the business domain. No experimental results for any Σ_2^P-complete KR problems are known.

Briefly, a holding owns companies, each of which produces some goods. Moreover, several companies may have joint control over another company. Now, some companies should be sold, under the constraint that all goods can be still produced, and that no company is sold which would still be controlled by the holding after the transaction. A company is *strategic,* if it belongs to a *strategic set,* which is a minimal set of companies satisfying these constraints. Those sets are expressed by the following natural program:

```
strategic(Cl) ∨ strategic(C2) ←
      produced by(P,Cl,C2).
strategic(C) ← controlled_by(C,Cl,C2,C3) ∧
      strategic(Cl) ∧ strategic(C2) ∧
      strategic(C3).
```

Here strategic*(C)* means that *C* is strategic, produced..by(7$^)$, Cl, *C2)* that product *P* is produced by companies Cl and C2, and controlledJby(C, Cl, C2,C3) that *C* is jointly controlled by C1,C2 and C3; we have adopted here from [Cadoli *ct. a/.*, 1997] that each product is produced by at most two companies and each company is under joint control of at most, three other companies.

The problem is to find out the set of all strategic companies (i.e., under brave reasoning, for which C the fact strategic(C) is true).

Note that this problem cannot, be expressed by a fixed normal (∨-free) logic program uniformly on all collections of facts produced by($p, c1, c2$) and controlled-by(c, el, e2,c3) (unless NP = Σ_2^P, an unlikely event). Thus, Strategic Companies is an example of a relevant problem where the expressive power of disjunctive logic programming is really needed.

Benchmark Data

We have generated tests with instances for n companies and 3n products.

We have uniform randomly chosen the producedJby and the controlled_by relations, where each company is controlled by one to five companies (Obeying the obvious constraints that no company can control itself, and that two consortia have to have at least one member in common). On average there are 1.5 controlled_by relationships per company.

Discussion of Results

Our experiments were run on a 133 MHz i486-compatible PC under Linux, using egcs-2.91.14 C++ compiler.

The results are displayed in the graphs of Figures 2- 4. Execution times are reported on the vertical axis, while the horizontal axis displays the size of the problem instance (number of companies of the instance of Strategic Companies to be solved). We have run instances of a size increasing a step of 5 companies, up to 100 companies.

The graph of Figure 2 shows the total time employed by the (stable) Model Checkers for the computation of all stable models of the program (i.e., for the generation of all sets of strategic companies). Note that a program having n stable models requires $m \geq n$ calls to the Model Checker ($m - n$ is the number of calls on models which an; not stable); the sum of the times employed for checking the; stability of all m models is displayed in the graph. For each problem size (number of companies) x, we have run 25 instances of size x and reported the average time. Thus, this graph shows the practical impact of the various model checking strategies on the computation.

Γ_M+DP clearly outperforms Old Checker: the time employed by Γ_M+DP to solve an instance of 100 companies is not sufficient for Old Checker even to solve instances of 25 companies. Interestingly, modular evaluation techniques appear very useful: Γ_M+DP+Mod takes less than 5% of +DP to solve problems of size 100.

The graph in Figure 3 refers to the same runs of the previous graph; but it shows the longest times required

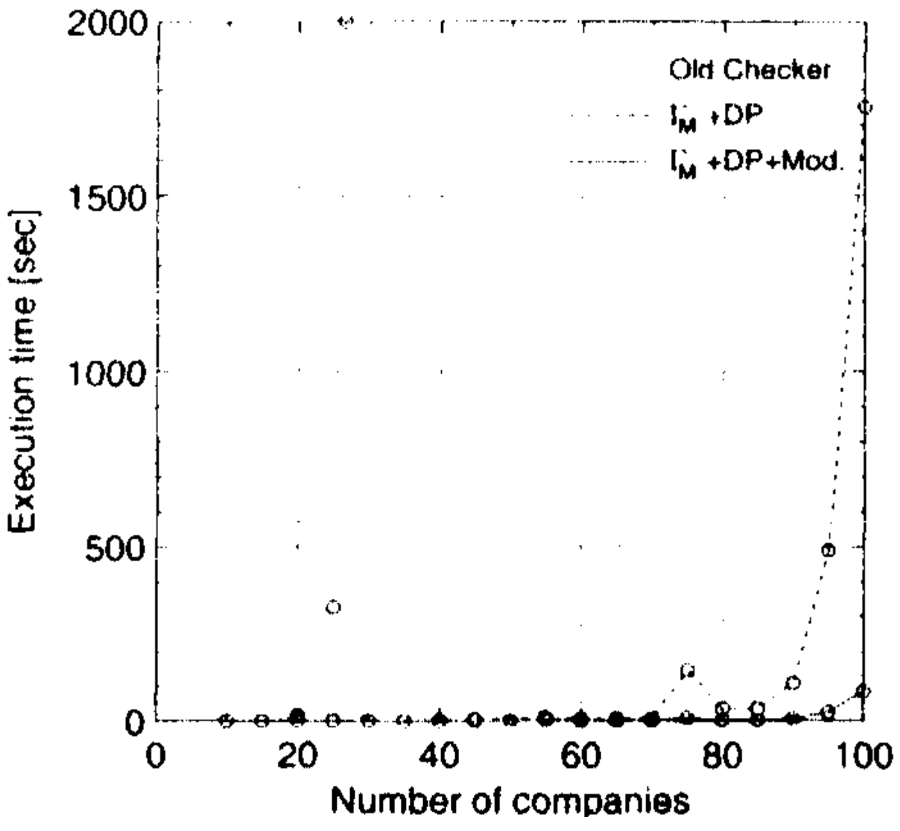

Figure 2: Total model checking time required for the computation of all stable models

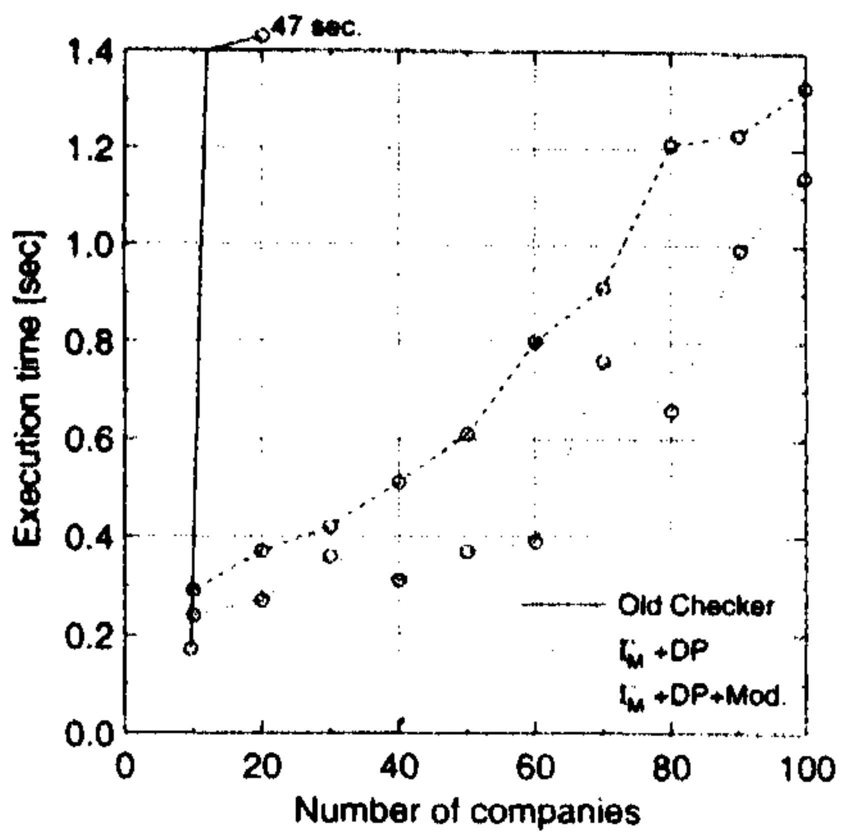

Figure 3: lime required for a single call to the MC (worst case)

by each method for checking the stability of a single model. This time, the curves are not very regular, because they visualize times of single calls (rather than average times as in Figure 2) of randomly generated instances. Also in this graph the new strategies show better performances than the old dlv checker. Surprisingly, the difference between the times of Γ_M+DP and Γ_M+DP+Mod is much smaller than in the previous graph. We have the following explanation for this phenomenon. Modular evaluation techniques speed up the computation of most instances. (Thus, modularity significantly affects total model checking time.) However, very hard instances cannot be "decomposed" by modular evaluation techniques. (Thus, the worst case times for single calls to the model checker are similar.)

Finally, the graph in Figure 4 refers to the computation of *one* stable model (i.e., to the computation of one set of strategic companies sufficient to solve the corresponding Σ_2^P decision problem). It compares the overall execution time (model generation -f model checking time) taken by the dlv system when the (old) model checker is replaced by our new model checkers. The advantage of using Γ_M+DP+Mod is very evident: by using Γ_M+DP+Mod an instance of size 100 is solved more quickly than an instance of size 20 with the old system. This graph also shows the time taken by dlv for model generation alone. Most of the execution time needed for computing one stable model is consumed for the generation of the models; only a very low percentage of the time is consumed by the Model Checker TM+DP+Mod itself.

In conclusion, observe that we have also compared the model checking strategies on a number of problems "easier" than Stategic Companies (at most NP-complete); as expected, the methods behaved very similarly.

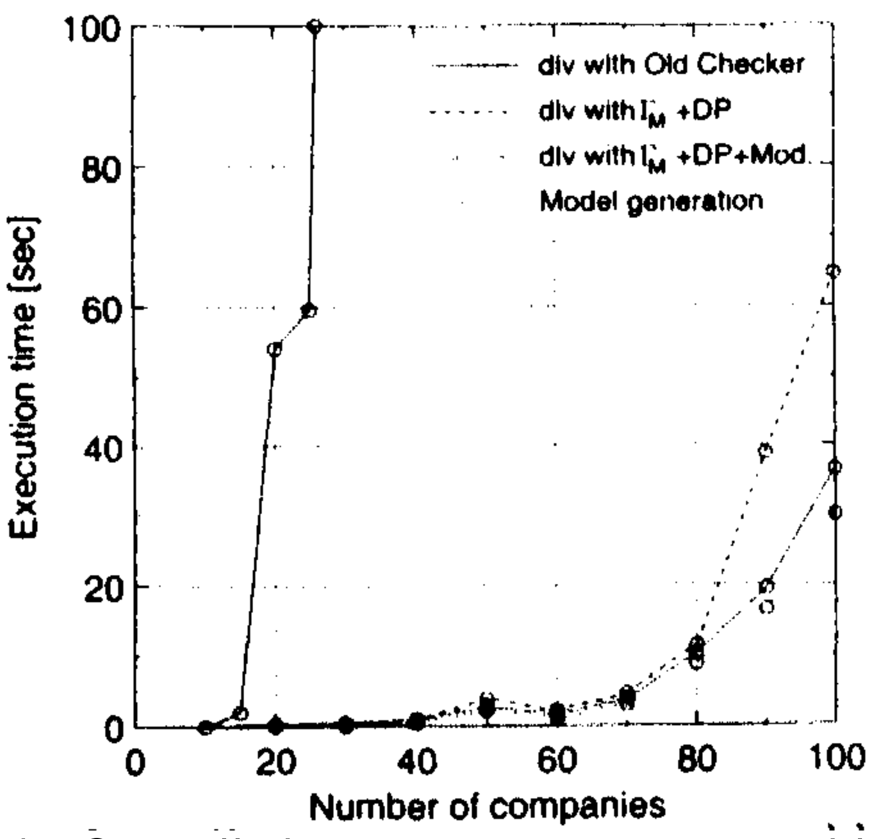

Figure 4: Overall time to compute one stable model

References

[Apt and Bol, 1994] K.R. Apt, R.N. Bol (1994), Logic Programming and Negation: A Survey, *Journal of Logic Programming,* **19/20,** 9-71.

[Baral and Gelfond, 1994] C. Baral, M. Gelfond (1994), Logic Programming and Knowledge Representation *Journal of Logic Programming,* **19/20,** 73 148.

[Ben-Eliyahu and Dechter, 1994] R. Ben-Eliyahu, R. Dechter (1994), Propositional Semantics for Disjunctive Logic Programs, *Annals of Mathematics and Artificial Intelligence,* Baltzer, **12,** pp. 53-87.

[Cadoli *el* a/., 1997] M. Cadoli, T. Eiter, G. Gottlob. (1997), Default Logic as a Querv Language. *IEEE-TKDE,* 9(3):448-463.

[Eiter *et.* a/., 1997a] T. Eiter, G. Gottlob, H. Mannila. (1997) Disjunctive Datalog. *ACM-TODS,* 22(3):315-363.

[Eiter *et.* a/., 1997b] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, F. Scarccllo. (1997) A Deductive System for Nonmonotonic Reasoning. *Proc. LPNMR '97,* pp. 363-374.

[Eiter *et.* a/., 1998] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, F. Scarcello. The KR System dlv: Progress Report, Comparisons and Benchmarks. *Proc. KR'98,* pp. 406-417.

(Gelfond and Lifschitz, 1991] M. Gelfond, V. Lifschitz (1991), Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing,* 9, 365-385.

[Gottlob, 1992] G. Gottlob. (1992) Complexity Results for Nonmonotonic Logics. *Journal of Logic and Corn-putatwn,* 2(3):397 425.

[Leone *et.* a/., 1997] N. Leone, P. Rullo, F. Scarcello. (1997) Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation,* 135(2):69-112.

[Lobo *et. al,* 1992] J. Lobo, .1. Minker, A. Rajasekar (1992), "Foundations of disjunctive logic programming," The MIT Press.

[Przymusinski, 1991] T. Przymusinski. (1991), Stable Semantics for Disjunctive Programs, *New Generation Computing,* 9, 401-424.

[Zhang, 1997] H. Zhang. (1997), SATO: An Efficient Propositional Prover. *Proc. CADE-97.*