# Evaluating an Algorithm for Default Reasoning

**Ilkka Niemelä and Patrik Simons**
Department of Computer Science
Helsinki University of Technology
Otakaari 1, FIN–02150 Espoo, Finland
Email: {Ilkka.Niemela,Patrik.Simons}@hut.fi

## Abstract

In this paper performance of a lately proposed decision procedure for Reiter's default logic is evaluated. Recent complexity results indicate that there are two orthogonal sources of complexity in default reasoning: classical (first-order) reasoning and conflict resolution (i.e. choosing an appropriate set of applicable non-conflicting default rules). For classical reasoning well-known techniques exist and conflict resolution presents the new computational challenge in default reasoning. However, efficient conflict resolution methods have been lacking. The recently introduced default reasoning algorithm pays special attention to conflict resolution and in this paper the efficiency of the conflict resolution technique in the new algorithm is examined. To rule out classical reasoning, a fragment of default logic with only atomic formulae is considered. This fragment corresponds to the stable model semantics of propositional logic programs. As an interesting test domain for nonmonotonic reasoning, combinatorial problems are put forward. In this paper combinatorial graph problems such as graph colorings and Hamiltonian circuits are used. In order to contrast the new algorithm with existing approaches, it is compared with an advanced special purpose implementation of the stable model semantics, the SLG system. The performance of the new algorithm compares favorably to that of SLG and the new algorithm is able to compute stable models quite efficiently for non-stratified (propositional) logic programs consisting of several hundreds of rules.

## 1 Introduction

We study automated theorem proving methods for default logic of Reiter [1980]. Although there are some approaches to automating default reasoning [Junker and Konolige, 1990; Ben-Eliyahu and Dechter, 1991; Baader and Hollunder, 1992; Risch and Schwind, 1994], relatively little attention has been paid to efficiency considerations. Also test cases of realistic size are lacking. In this paper we put forward combinatorial problems as an application domain with rich and interesting patterns of test cases. Using this domain we study the performance of a novel decision method [Niemelä, 1995] for default logic. In particular, we are interested in evaluating the efficiency of the conflict resolution technique in the novel method.

The computational complexity of nonmonotonic reasoning has been studied intensively during the last years. Propositional default reasoning has been shown to be complete with respect to the second level of the polynomial time hierarchy [Gottlob, 1992]. This indicates that there are two independent sources of complexity in default reasoning: (i) *classical (first-order) reasoning* and (ii) *conflict resolution* (i.e. choosing an appropriate set of applicable non-conflicting default rules). In order to develop efficient implementations of default reasoning both sources have to be addressed.

For classical reasoning well-known techniques exist and conflict resolution presents the new computational challenge in default reasoning. The initial search space for conflict resolution is exponential in the number of default rules and brute force search methods are able to handle only very modest sized sets of default rules. However, existing approaches to automating default reasoning have paid little attention to conflict resolution. For stratified default theories, where the interaction of default rules is limited, there exist techniques to solve the conflict resolution task in polynomial time [Niemelä and Rintanen, 1992]. For the non-stratified case we have developed a novel decision method [Niemelä, 1995] that focuses on the conflict resolution problem.

In this paper our aim is to evaluate the conflict resolution capabilities of the novel method. To rule out (as much as possible) the classical reasoning task we consider a fragment of default logic with only atomic formulae. In effect this leads us to consider the stable model semantics of propositional logic programs. As an interesting test domain we put forward combinatorial problems[1]. It turns out that combinatorial problems can be described as logic programs in such a way that a stable model of the program provides a solution to the original combinatorial problem. Furthermore, there is a wealth of practically

[1]This domain was initially suggested to us by V. Marek and M. Truszczyński.

relevant combinatorial problems with a lot of results, literature, and algorithms available for comparisons. In this paper we consider combinatorial graph problems. To generate test cases we have exploited the Stanford GraphBase developed by Donald E. Knuth [1993] that is a portable software package for generating and processing graphs.

There are several other proposals for computing stable models [Bell *et al.*, 1994; Chen and Warren, 1994; Eshghi, 1990; Satoh and Iwayama, 1991]. In order to contrast the novel algorithm to existing approaches we wanted to use a special purpose implementation of the stable model semantics. As such a challenge system we employ the SLG system developed by Chen and Warren [1993] because it is freely available and because it seems to be a good representative of the current state of the art in computing stable models [Chen and Warren, 1994].

The paper is organized as follows. Section 2 introduces the stable model semantics and develops a concise characterization of stable models. This characterization forms the basis for the novel algorithm. Section 3 presents the new algorithm, which was originally developed for default logic, directly in terms of stable models of logic programs. Section 4 briefly describes our implementation, Section 5 the experiments and Section 6 contains the concluding remarks.

## 2 Stable Model Semantics

In this section we briefly introduce the stable model semantics and present a characterization of stable models that serves as the basis of the novel algorithm for computing stable models. We consider general (propositional) logic programs consisting of rules of the form

$$c \leftarrow a_1, \ldots, a_n, \mathrm{not}(b_1), \ldots, \mathrm{not}(b_m). \tag{1}$$

where $a_i, b_j$ and $c$ are atoms. For general logic programs the *stable model semantics* is defined as follows [Gelfond and Lifschitz, 1988]. The *reduct* $P^S$ of $P$ with respect to a set of atoms $S$ is obtained from $P$ by deleting (i) each rule that has a formula $\mathrm{not}(b)$ in its body with $b \in S$ and (ii) all formulae of the form $\mathrm{not}(l)$ in the bodies of the remaining rules. Now a set of atoms $S$ is a stable model of $P$ iff $S$ is the deductive closure of $P^S$ when rules in $P^S$ are seen as inference rules.

We reformulate the definition of a stable model in terms of a deductive closure operator $\mathrm{Dcl}(\cdot,\cdot)$ that we use for describing our approach to computing stable models. For a set of formulae $S$, let $\mathrm{not}(S) = \{\mathrm{not}(\phi) \mid \phi \in S\}$. By $S^+$ ($S^-$) we denote the set of formulae (not-formulae) in $S$. For example, if $S = \{a, \mathrm{not}(b), \mathrm{not}(c)\}$, $S^+ = \{a\}$ and $S^- = \{\mathrm{not}(b), \mathrm{not}(c)\}$. We denote by $\mathrm{Dcl}(P, L)$ the *deductive closure* of a set of rules $P$ of the form (1) and a set of formulae and not-formulae $L$. $\mathrm{Dcl}(P, L)$ is the smallest set of formulae which contains $L^+$ and which is closed under $\mathrm{R}(P, L)$ where

$$\begin{aligned}\mathrm{R}(P, L) \quad = \quad & \{c \leftarrow a_1, \ldots, a_n \mid \\ & c \leftarrow a_1, \ldots, a_n, \mathrm{not}(b_1), \ldots, \mathrm{not}(b_m) \in P \\ & \text{and for all } i = 1, \ldots, m, \mathrm{not}(b_i) \in L^-\}.\end{aligned}$$

For example, let $P = \{q \leftarrow \mathrm{not}(p);\ r \leftarrow q\}$. Then $\mathrm{R}(P, \{p, \mathrm{not}(p)\}) = \{q \leftarrow\ ;\ r \leftarrow q\}$ and $\mathrm{Dcl}(P, \{p, \mathrm{not}(p)\}) = \{p, q, r\}$).

It is straightforward to show that $\Delta$ is a stable model of a set of rules $P$ iff

$$\Delta = \mathrm{Dcl}(P, \mathrm{not}(\overline{\Delta}))$$

where $\overline{\Delta}$ is the complement of $\Delta$, i.e., the atoms not in $\Delta$.

In [Niemelä, 1995] we presented a compact characterization of extensions in default logic. Because of the close relationship between extensions and stable models, this approach is directly applicable to stable models. We present the characterization here directly in terms of stable models of general logic programs. The formulae inside the $\mathrm{not}(\cdot)$ operator play a crucial role. For a set of rules $P$, we denote by $\mathrm{NAnt}(P)$ the *negative antecedents* in $P$, i.e., the set of formulae $b$ such that $\mathrm{not}(b)$ appears in $P$. For example, $\mathrm{NAnt}(\{q \leftarrow \mathrm{not}(p); r \leftarrow q\}) = \{p\}$.

For the deductive closure of a set of rules $P$ the formulae in $\mathrm{NAnt}(P)$ are of importance, i.e. $\mathrm{Dcl}(P, \mathrm{not}(\overline{\Delta})) = \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - \Delta))$. Thus, for a set of rules $P$, a set of atoms $\Delta$ is a stable model of $P$ iff

$$\Delta = \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - \Delta)).$$

The characterizing sets for stable models are called *full sets* and they are sets of not-formulae built from formulae in $\mathrm{NAnt}(P)$.

**Definition 2.1** *A set $\Lambda$ of not-formulae is called $P$-full iff the following condition holds for all $\phi \in \mathrm{NAnt}(P)$, $\mathrm{not}(\phi) \in \Lambda$ iff $\phi \notin \mathrm{Dcl}(P, \Lambda)$.*

The following characterization theorem provides the basis for the novel algorithm.

**Theorem 2.2** *(i) If a set $\Lambda$ is $P$-full, then $\mathrm{Dcl}(P, \Lambda)$ is a stable model of $P$.*

*(ii) If there is a stable model $\Delta$ of $P$, then $\Lambda = \mathrm{not}(\mathrm{NAnt}(P) - \Delta)$ is a $P$-full set such that $\Delta = \mathrm{Dcl}(P, \Lambda)$.*

Theorem 2.2 suggests the following straightforward method for finding all stable models. For each subset $S$ of $\mathrm{NAnt}(P)$, test whether $\mathrm{not}(S)$ is $P$-full. If a full set $\Lambda$ is found, $\mathrm{Dcl}(P, \Lambda)$ is a stable model of $P$ by Theorem 2.2 (i) and by Theorem 2.2 (ii) for each stable model $\Delta$, there is a corresponding full set $\Lambda$ such that $\Delta = \mathrm{Dcl}(P, \Lambda)$. However, this method is not very practical: if there are $n$ not-formulae in $P$, $2^n$ fullness tests are needed. The novel algorithm presented in the next section is also based on finding full sets for the stable models but it offers significant improvements over the straightforward method.

## 3 A Procedure for Computing Stable Models

In [Niemelä, 1995] we presented a decision procedure for default logic. Besides computing extensions of a default theory, this algorithm handles two basic query-answering tasks: *brave reasoning* (whether a default theory has an extension containing a formula) and *cautious reasoning* (whether a formula is contained in every extension of a default theory). The algorithm is based on ideas introduced in [Niemelä, 1994] in the context of autoepistemic reasoning. In this paper we use this algorithm to implement the stable model semantics. In this section we briefly describe the algorithm from [Niemelä, 1995] directly in terms of stable models of general logic programs. The algorithm is presented in Figure 1 as a function **smodels** that is the skeleton for the decision procedures for brave and cautious reasoning as well as for computing stable models.

When describing the algorithm we use the following three concepts.

- A set $\Lambda$ of formulae and not-formulae is *grounded* in a set of rules $P$ if $\Lambda^+ \subseteq \mathrm{Dcl}(P, \Lambda^-)$. For example, $\{\mathrm{not}(b)\}$ and $\{b, \mathrm{not}(a)\}$ are grounded in $\{b \leftarrow \mathrm{not}(a);\ a \leftarrow a\}$ but $\{a\}$ is not.
- We say that a set of formulae $S$ *agrees* with a set of formulae and not-formulae $\Lambda$ if for all formulae $\phi \in \Lambda^+$, $\phi \in S$ and for all $\mathrm{not}(\phi) \in \Lambda^-$, $\phi \notin S$. For example, the set $\{a, b\}$ agrees with $\{a, \mathrm{not}(c)\}$ but not with $\{a, \mathrm{not}(b)\}$.
- A set $\Lambda$ of formulae and not-formulae *covers* a formula $\phi$ if either $\phi \in \Lambda$ or $\mathrm{not}(\phi) \in \Lambda$. For example, the set $\{\mathrm{not}(a), b\}$ covers $a$. A set of formulae $S$ is covered by $\Lambda$ if each formula in $S$ is covered by $\Lambda$.

The function **smodels** takes as input a set of rules $P$, sets $B$ and $F$ which represent the common part of the stable model to be considered and a formula $\phi$ which is just passed as an argument to the function **test**. The aim of **smodels** is to return true iff there is a stable model $\Delta$ of $P$ agreeing with $B \cup F$ such that **test**$(\Delta, \phi)$ returns true. This is accomplished by constructing $P$-full sets agreeing with $B \cup F$ until a full set $\Lambda$ is found such that for the corresponding stable model $\Delta = \mathrm{Dcl}(P, \Lambda)$, **test**$(\Delta, \phi)$ returns true.

We represent a partially constructed full set using the set $B$ that contains not-formulae and ordinary formulae. The not-formulae $B^-$ are the formulae included in the partially constructed full set. An ordinary formula $\phi$ in $B$ indicates that the corresponding not-formula $\mathrm{not}(\phi)$ cannot be included in the full set as $\phi \in \mathrm{Dcl}(P, B^-)$. The set $F$ contains formulae $\phi$ for which $\mathrm{not}(\phi)$ is excluded from the full set to be constructed (and $\phi$ should be included in $B$), but which are "frozen": $\phi \in F$ is added to $B$ only when the groundedness condition is satisfied, i.e., $\phi \in \mathrm{Dcl}(P, B^-)$.

The function **smodels** uses three additional functions: **expand** (for expanding $B$), **conflict** (for detecting conflicts), and **test** (for testing stable models). By changing the function **test** the various decision procedures are obtained. The first two functions form the crucial subtasks where optimization techniques can be applied. In [Niemelä, 1995] we have developed optimized versions of the functions **expand** and **conflict** presented in Figure 2. We employ these versions in our implementation.

The function **smodels** starts by expanding cautiously the set $B$. For all negative antecedents $\phi$ of $P$ not covered by $B$, the optimized version of the function **expand** adds (i) $\phi$ to $B$ if $\phi$ is derivable from $P$ and $B^-$, i.e., $\phi \in \mathrm{Dcl}(P, B^-)$ and (ii) $\mathrm{not}(\phi)$ to $B$ if $\phi$ is no longer even potentially derivable, i.e., $\phi \notin \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$. Then a conflict test is performed. The optimized version of the function **conflict** captures two kinds

**function smodels**$(P, B, F, \phi)$
$B :=$ **expand**$(P, B, F)$;
**if conflict**$(P, B, F)$ returns true **then**
    return false
**else if** $\mathrm{NAnt}(P)$ is covered by $B \cup F$ **then**
    **if** $F \subseteq B$ **then**
        return **test**$(\mathrm{Dcl}(P, B^-), \phi)$
    **else**
        return false
    **endif**
**else**
    take some $\chi \in \mathrm{NAnt}(P)$ not covered by $B \cup F$;
    **if smodels**$(P, B \cup \{\mathrm{not}(\chi)\}, F, \phi)$ returns true **then**
        return true
    **else**
        return **smodels**$(P, B, F \cup \{\chi\}, \phi)$
    **endif**
**endif**

Figure 1: The Skeleton for the Decision Procedures for Logic Programs with the Stable Model Semantics

**function expand**$(P, B, F)$
**repeat**
    $B' := B$;
    $B := B \cup \{\phi \in \mathrm{NAnt}(P) \mid \phi$ is not covered by $B$, $\phi \in \mathrm{Dcl}(P, B^-)\}$;
    $B := B \cup \{\mathrm{not}(\phi) \mid \phi \in \mathrm{NAnt}(P), \phi$ is not covered by $B, \phi \notin \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))\}$;
**until** $B = B'$;
return $B$.

**function conflict**$(P, B, F)$
**if** for some $\mathrm{not}(\phi) \in B$, $\phi \in \mathrm{Dcl}(P, F \cup B)$ or for some $\phi \in F$, $\phi \notin \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$ **then**
    return true
**else**
    return false.

Figure 2: Optimized Versions of the Functions **expand** and **conflict**

of conflicts: (i) there is some $\mathrm{not}(\phi) \in B$ for which $\phi$ will be derivable when the frozen formulae are included, i.e., $\phi \in \mathrm{Dcl}(P, F \cup B)$ and (ii) there is some frozen formula $\phi \in F$ that is no longer even potentially derivable, i.e., $\phi \notin \mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$.

If there are no conflicts and $B \cup F$ covers every not-formula in the premises, then the status of the frozen formulae $F$ is examined. If all of them have been included in $B$, $B^-$ is a full set for the stable model $\mathrm{Dcl}(P, B^-)$ and **smodels** returns what $\mathbf{test}(\mathrm{Dcl}(P, B^-), \phi)$ returns. If there is a formula $\mathrm{not}(\chi)$ in the premises such that $\chi$ is not covered by $B \cup F$, then either $\chi$ belongs to the stable model to be constructed or not. The two alternatives are handled by backtracking. First $B$ is extended by $\mathrm{not}(\chi)$ (assuming that $\chi$ is not in the stable model) but if $\mathbf{smodels}(P, B \cup \{\mathrm{not}(\chi)\}, F, \phi)$ returns false, $\chi$ must belong to the stable model and $\chi$ is taken as a frozen formula. Then **smodels** returns what $\mathbf{smodels}(P, B, F \cup \{\chi\}, \phi)$ returns. It can be shown that **smodels** is sound and complete.

**Theorem 3.1** *Let $B$ be a set of formulae and not-formulae, $F$ a set of atoms, and $B$ be grounded in $P$. Then* $\mathbf{smodels}(P, B, F, \phi)$ *returns true iff there exists a stable model $\Delta$ of $P$ which agrees with $B \cup F$ and for which* $\mathbf{test}(\Delta, \phi)$ *returns true.*

The theorem implies that decision procedures for the different reasoning tasks can be obtained by (i) taking the empty set as the common part of the stable models to be constructed and (ii) choosing an appropriate **test** function as shown in the following corollary.

**Corollary 3.2** *Let $P$ be a set of rules and $\phi$ an atom.*

1. *If* $\mathbf{test}(\Delta, \phi)$ *returns true for all $\Delta$ and $\phi$, then* $\mathbf{smodels}(P, \emptyset, \emptyset, \phi)$ *returns true iff $P$ has a stable model.*
2. *If* $\mathbf{test}(\Delta, \phi)$ *returns true iff $\phi \in \Delta$, then* $\mathbf{smodels}(P, \emptyset, \emptyset, \phi)$ *returns true iff there exists a stable model of $P$ containing $\phi$.*
3. *If* $\mathbf{test}(\Delta, \phi)$ *returns true iff $\phi \notin \Delta$, then* $\mathbf{smodels}(P, \emptyset, \emptyset, \phi)$ *returns true iff there exists a stable model of $P$ not containing $\phi$ (and, in other words,* $\mathbf{smodels}(P, \emptyset, \emptyset, \phi)$ *returns false iff $\phi$ is contained in every stable model of $P$).*

**Example 3.1** Let

$$P = \{b \leftarrow \mathrm{not}(a); a \leftarrow \mathrm{not}(b); a \leftarrow b\}.$$

We use the decision procedure to determine whether $P$ has a stable model containing $b$. Hence, we use a **test** function such that $\mathbf{test}(\Delta, \phi)$ returns true iff $\phi \in \Delta$ and take the empty set as the initial common part of the stable models to be considered. So we consider the execution of

$$\mathbf{smodels}(P, \emptyset, \emptyset, b).$$

First $\mathbf{expand}(P, \emptyset, \emptyset)$ returns $\emptyset$ and $\mathbf{conflict}(P, \emptyset, \emptyset)$ returns false. Since $\mathrm{NAnt}(P) = \{a, b\}$ is not covered by $\emptyset$, we must choose either $a$ or $b$ in order to continue. Let us take $a$. Then

$$\mathbf{smodels}(P, \{\mathrm{not}(a)\}, \emptyset, b)$$

is executed. Now $\mathbf{expand}(P, \{\mathrm{not}(a)\}, \emptyset)$ returns $\{\mathrm{not}(a), b\}$. Then $\mathbf{conflict}(P, \{\mathrm{not}(a), b\}, \emptyset)$ returns true as $a \in \mathrm{Dcl}(P, \{\mathrm{not}(a), b\})$. So $\mathbf{smodels}(P, \{\mathrm{not}(a)\}, \emptyset, b)$ returns false and

$$\mathbf{smodels}(P, \emptyset, \{a\}, b)$$

is executed. Now $\mathbf{expand}(P, \emptyset, \{a\})$ returns $\{\mathrm{not}(b), a\}$ because $\mathrm{not}(\mathrm{NAnt}(P) - B \cup F) = \{\mathrm{not}(b)\}$ and $b \notin \mathrm{Dcl}(P, \{\mathrm{not}(b)\})$. Then $\mathbf{conflict}(P, \{\mathrm{not}(b), a\}, \{a\})$ returns false. Because $\mathrm{NAnt}(P)$ is covered by $B \cup F = \{\mathrm{not}(b), a\}$ and $F = \{a\} \subseteq \{\mathrm{not}(b), a\} = B$, we have found a stable model $\mathrm{Dcl}(P, B^-) = \mathrm{Dcl}(P, \{\mathrm{not}(b)\}) = \{a\}$ of $P$ and

$$\mathbf{test}(\mathrm{Dcl}(P, \{\mathrm{not}(b)\}), b)$$

is performed but false is returned. This implies that $\mathbf{smodels}(P, \emptyset, \{a\}, b)$ returns false and $\mathbf{smodels}(P, \emptyset, \emptyset, b)$ returns false. Hence, there is no stable model of $P$ containing $b$. ∎

## 4 Implementation

The algorithm **smodels** presented in this paper has been implemented in C++. There are a few implementation decisions that affect the performance of the algorithm: which data structures are used to implement a set of atoms and a set of rules, how backtracking information is handled, what algorithm is used to calculate the deductive closure, and what kind of search heuristic is used. A brief description of our choices follows.

Propositional atoms are stored as numbers. The rules are stored consecutively in an array, and an index array is used to tell where each rule resides and where the consequent, the positive part of the body, and the negative part of the body are stored. These arrays are henceforth treated as a single object, a $P$-set.

For implementing the sets $B$ and $F$ we use a special data structure which we call an atom-set. For $B$ we use two atom-sets $B^+$ and $B^-$ that contain atoms and not-atoms, respectively. The set $F$ containing frozen atoms is implemented as a single atom-set.

An atom-set consists of two arrays and two indices. The first array presents the actual set of propositional atoms in question: the nth element of the array is one, if atom number n belongs to the set, and zero otherwise. In order to avoid unnecessary copying of this array when making recursive calls, the second array is used to hold the atoms that have been added to the set, i.e. it holds backtracking information. The indices point to the end-points of the segment in the second array that contain the atoms that should be removed when backtracking. In this way, for backtracking we only have to pass the two indices along to make it possible to restore the initial state of the atom array. The array containing backtracking information also provides a convenient way to iterate over the propositional atoms in the set.

The deductive closure $\mathrm{Dcl}(P, L)$ is computed by taking the reduct $\mathrm{R}(P, L)$ and then using the first linear-time algorithm of Dowling and Gallier [1984] to compute the closure of the reduct. We have to compute three different deductive closures: $\mathrm{Dcl}(P, B^-)$, $\mathrm{Dcl}(P, F \cup B)$, and $\mathrm{Dcl}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$. But as the linear algorithm computes the deductive closure from the reduct of $P$ and as $\mathrm{R}(P, B^-) = \mathrm{R}(P, F \cup B)$, we only have to initialize the algorithm with two different reducts. The two reducts could be constructed from $P$ every time a closure is needed. However, one can avoid some unnecessary work by storing backtracking information in the following way.

When the function **smodels** recursively calls itself, the set $\mathrm{not}(\mathrm{NAnt}(P) - (B \cup F))$ diminishes as long as no backtracking occurs. This means that the rules removed from $P$, to get $\mathrm{R}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$, constitutes a growing set. Therefore, an object can be used to keep track of the set of rules in $\mathrm{R}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$ in a similar fashion to what was done in the case of the atom-set. More precisely, an array with indices into the $P$-set, keeps tab on the rules in $\mathrm{R}(P, \mathrm{not}(\mathrm{NAnt}(P) - (B \cup F)))$, and an array with backtracking information stores the indices to the rules removed from this array.

One can also observe that the set $B^-$ grows during recursive calls with no backtracking. Thus, in this case, the rules in $\mathrm{R}(P, B^-)$ constitutes a growing set, and therefore one can define an object that remembers which rules are not in $\mathrm{R}(P, B^-)$. Although, some of these rules possibly have to be moved to $\mathrm{R}(P, B^-)$ as $B^-$ grows, the number of them decreases as $B^-$ grows, and therefore the time spent testing them for membership in $\mathrm{R}(P, B^-)$ gets shorter.

In the algorithm the search can be guided by the choices made when selecting new atoms not yet covered by $B \cup F$ from $\mathrm{NAnt}(P)$. In the implementation we are using the following *dynamic* search heuristic. Given $B$ and $F$, we select when possible an atom $\chi$ for which $\mathrm{not}(\chi)$ appears in a rule which has the least number of not-formulae not in $B$ among the set of currently potentially applicable rules (i.e. rules without a formula $\mathrm{not}(b)$ for which $b \in B \cup F$).

## 5 Experimental Results

The execution times of the algorithm **smodels** have been compared to those of SLG [Chen and Warren, 1994; 1993] by using the two algorithms to search for 3-colorings and Hamiltonian circuits in different graphs. The graphs were created using the Stanford GraphBase [Knuth, 1993], which is a highly portable collection of programs and data that serves as a platform for combinatorial algorithms. In addition, **smodels** has been tested on the two benchmark programs used in [Chen and Warren, 1994].

Two types of graphs have been used in the tests. A special type of planar graph, created by the *plane* function found in the Stanford GraphBase, and a 'ladder'-graph, see Figure 3. The planar graphs are constructed by Delaunay triangulation of randomly inserted points in a plane.

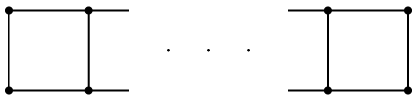

Figure 3: Ladder Graph.

Test cases are generated from the graphs by translating a given graph to a propositional logic program in such a way that a stable model of the program provides a solution to the problem.

The 3-coloring problem is translated from a graph into a logic program in the following way. For each vertex $a$ with neighbors $p_1, \ldots, p_j$ and each color $i \in \{0, 1, 2\}$, include the rule

$$\begin{aligned} \mathit{color}(a,i) \quad \leftarrow \quad & \mathrm{not}(\mathit{color}(p_1,i)), \ldots, \mathrm{not}(\mathit{color}(p_j,i)), \\ & \mathrm{not}(\mathit{color}(a, i+1 \pmod 3)), \\ & \mathrm{not}(\mathit{color}(a, i+2 \pmod 3)) \end{aligned}$$

and for each vertex $a$ include the rule

$$h \leftarrow \mathrm{not}(\mathit{color}(a,0)), \mathrm{not}(\mathit{color}(a,1)), \mathrm{not}(\mathit{color}(a,2)).$$

Finally add the rules $h_2 \leftarrow \mathrm{not}(h)$ and $h_2 \leftarrow \mathrm{not}(h_2)$. The number of rules is thus $4V + 2$ if $V$ is the number of vertices.

The translation of the Hamiltonian circuit problem is somewhat more complicated. For each pair $(a, b)$ of vertices in the graph such that there is an edge between $a$ and $b$, and where $b_1, \ldots, b_i$ are the neighbors of $a$ excluding $b$, and $a_1, \ldots, a_j$ are the neighbors of $b$ excluding $a$, include the rule

$$\begin{aligned} \mathit{edge}(b,a) \quad \leftarrow \quad & \mathrm{not}(\mathit{edge}(b_1,a)), \ldots, \mathrm{not}(\mathit{edge}(b_i,a)), \\ & \mathrm{not}(\mathit{edge}(b,a_1)), \ldots, \mathrm{not}(\mathit{edge}(b,a_j)), \\ & \mathrm{not}(\mathit{edge}(a,b)). \end{aligned}$$

Fix a vertex $d$. For each pair $(a, b)$ of vertices in the graph such that there is an edge between $a$ and $b$, if $b \neq d$ include the rule

$$b \leftarrow a, \mathit{edge}(a,b)$$

and if $b = d$ include the rule

$$d' \leftarrow a, \mathit{edge}(a,d).$$

Finally, for each vertex $a \neq d$ add $a \leftarrow \mathrm{not}(a)$, and then add the rules $d \leftarrow$ and $d' \leftarrow \mathrm{not}(d')$. If the number of vertices is denoted by $V$ and the number of edges by $E$, the number of rules in this problem is $4E + V + 1$.

The two benchmark programs found in [Chen and Warren, 1994] have also been used to generate test cases. The first benchmark program consists of the following rules:

$$\begin{array}{ll} z1(X) \leftarrow v1(X), w1(X) & z2(X) \leftarrow v1(X), w2(X) \\ z3(X) \leftarrow v2(X), w1(X) & z4(X) \leftarrow v2(X), w2(X) \\ v1(X) \leftarrow s(X) & v2(X) \leftarrow t(X) \\ w1(X) \leftarrow p(X) & w2(X) \leftarrow q(X) \\ t(X) \leftarrow \mathrm{not}(s(X)) & s(X) \leftarrow \mathrm{not}(t(X)) \\ p(X) \leftarrow \mathrm{not}(q(X)) & q(X) \leftarrow \mathrm{not}(p(X)) \\ m(X) \leftarrow z1(X) & m(X) \leftarrow z2(X) \\ m(X) \leftarrow z3(X) & m(X) \leftarrow z4(X) \end{array}$$

Before the stable models are computed a number of constants are introduced and the program is grounded, i.e. the variables in the rules are instantiated with the constants. If the number of constants is $n$, the number of stable models is $2^{2n}$.

The second benchmark program consists of the following rules:

$$\begin{array}{ll} s(X) \leftarrow p(X), q(X) & s(X) \leftarrow p(X), r(X) \\ s(X) \leftarrow q(X), r(X) & p(X) \leftarrow \mathrm{not}(q(X)) \\ q(X) \leftarrow \mathrm{not}(r(X)) & r(X) \leftarrow \mathrm{not}(p(X)) \end{array}$$

A number of constants are introduced and the program is grounded. This program does not have any stable models.

Both benchmark programs are very simple since there are few interdependencies between the propositional atoms. This fact is clearly reflected in the timing results.

The results are tabulated in Figure 4. They have been calculated as the average of the times for five different runs on a pseudo-randomly shuffled set of rules. The rationale for shuffling the set of rules is that a particular ordering of the rules might help the algorithm to avoid backtracking and thus give a skewed picture of its behavior. A bar (-) in the table indicates that the program ran for more than 10 minutes on each shuffle without returning an answer and was then interrupted. All times are in seconds and they represent the time to find one stable model if one exists, or the time to decide that there are no stable models. As an exception all stable models of the first benchmark program were computed.

All tests were run on a SPARCstation IPX workstation, with 24 MB of memory and the SunOS 4.1.3 operating system. The SLG system was compiled in SICStus 2.1 #5 with the fastcode compilation-flag set. The C++ program **smodels** was compiled using gcc version 2.5.8.

The test results show that our implementation of the new algorithm **smodels** compares favorably to SLG. The results on the benchmark programs 1 and 2 from [Chen and Warren, 1994], where the non-stratified parts are quite small and simple in structure, indicate that our basic implementation is more efficient than that of SLG. This can be explained by the different implementation techniques. We use C++ and a linear time algorithm [Dowling and Gallier, 1984] for deduction whereas SLG has been implemented as a meta-interpreter on top of Prolog.

The other test cases concerning 3-colorings and Hamiltonian circuits are much more difficult since they are highly non-stratified with complicated relationships between propositions. Here advanced conflict resolution methods are needed in order to be able to handle even a few dozens of rules. The conflict resolution technique of the new algorithm [Niemelä, 1995] performs quite well and is able to handle sets of several hundreds of rules. On the other hand, the capacity of SLG seems to be limited to 100–150 rules on examples like these. This seems to be due to difficulties in pruning the search space for conflict resolution adequately when the size of the set of rules grows. Hence, it appears that the novel algorithm advances the state of the art in conflict resolution and enables applications where several hundreds of non-stratified rules are involved.

## 6 Conclusions

In this paper our aim has been to evaluate the performance of a recently proposed decision procedure for Reiter's default logic [Niemelä, 1995]. In particular, we wanted to study the efficiency of the conflict resolution technique in the new algorithm. This led us to consider a fragment of default logic corresponding to the stable model semantics of propositional logic programs. As an

| Ladder graphs, 3-coloring | | | |
|---|---|---|---|
| Vertices | Rules | SLG | **smodels** |
| 10 | 42 | 0.93 | 0.02 |
| 20 | 82 | 212.12 | 0.06 |
| 30 | 122 | - | 0.12 |
| 40 | 162 | - | 0.20 |
| 50 | 202 | - | 0.27 |
| 400 | 1602 | - | 16.60 |

| Planar graphs, 3-coloring | | | |
|---|---|---|---|
| Vertices | Rules | SLG | **smodels** |
| 10 | 42 | 35.60 | 0.05 |
| 12 | 50 | 107.57 | 0.06 |
| 14 | 58 | 424.43 | 0.09 |
| 16 | 66 | - | 0.11 |
| 18 | 74 | - | 0.11 |
| 400 | 1602 | - | 3.80 |

| Ladder graphs, Hamiltonian circuit | | | |
|---|---|---|---|
| Vertices | Rules | SLG | **smodels** |
| 10 | 63 | 1.53 | 0.05 |
| 14 | 91 | 5.42 | 0.16 |
| 18 | 119 | 20.34 | 0.28 |
| 22 | 147 | 93.39 | 0.77 |
| 26 | 175 | 481.43 | 1.24 |
| 30 | 203 | - | 3.05 |
| 40 | 273 | - | 4.94 |
| 50 | 343 | - | 11.81 |

| Planar graphs, Hamiltonian circuit | | | |
|---|---|---|---|
| Vertices | Rules | SLG | **smodels** |
| 10 | 95 | 29.44 | 0.32 |
| 12 | 121 | 78.97 | 1.26 |
| 14 | 147 | 884.65 | 3.18 |
| 16 | 173 | - | 5.92 |
| 18 | 199 | - | 95.12 |

| SLG benchmark 1, all stable models | | |
|---|---|---|
| Constants | SLG | **smodels** |
| 1 | 0.11 | 0.01 |
| 2 | 0.28 | 0.03 |
| 3 | 1.00 | 0.16 |
| 4 | 3.57 | 0.75 |
| 5 | 15.22 | 3.69 |

| SLG benchmark 2 | | |
|---|---|---|
| Constants | SLG | **smodels** |
| 20 | 0.53 | 0.01 |
| 30 | 0.85 | 0.02 |
| 40 | 1.43 | 0.02 |
| 50 | 1.53 | 0.02 |
| 100 | 2.93 | 0.05 |

Figure 4: Test Results

interesting test domain for nonmonotonic reasoning, we put forward combinatorial problems. In this paper we have considered combinatorial graph problems such as graph colorings and Hamiltonian circuits. Combinatorial problems offer a rich domain where test cases of increasing degree of difficulty can be generated and where a lot of literature and results already exist. An interesting area of further research is a comparison between implementations of nonmonotonic reasoning and special purpose algorithms for solving combinatorial problems. Little is known about the nature of typical common sense reasoning tasks and more work is needed to develop representative test cases for nonmonotonic reasoning systems.

As a challenge system with which the novel algorithm is compared we have used an advanced implementation of a stable model algorithm, the SLG system. The new algorithm compares favorably to SLG on combinatorial graph problems and is able to handle quite efficiently non-stratified (propositional) logic programs with several hundreds of rules. Some of the (quite drastic) performance dissimilarities can be explained by differences in implementation techniques: our algorithm is implemented in C++ while SLG is implemented as a meta-interpreter on top of Prolog. However, it seems that the new algorithm advances the state of the art in conflict resolution. Like SLG it is based on the idea of using well-founded models as an approximation of stable models but, unlike in SLG, well-founded models are used recursively to prune the search space for conflict resolution. This recursive use of well-founded models together with a dynamic search heuristic (depending on the current state of the search) leads to performance unmatched by SLG.

## References


[Baader and Hollunder, 1992] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 306–317, Cambridge, MA, USA, October 1992. Morgan Kaufmann Publishers.

[Bell *et al.*, 1994] C. Bell, A. Nerode, R.T. Ng, and V.S. Subrahmanian. Mixed integer programming methods for computing nonmonotonic deductive databases. *Journal of the ACM*, 41(6):1178–1215, November 1994.

[Ben-Eliyahu and Dechter, 1991] R. Ben-Eliyahu and R. Dechter. Default logic, propositional logic and constraints. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 370–385. The MIT Press, July 1991.

[Chen and Warren, 1993] W. Chen and D.S. Warren. The SLG system, 1993. Available by ftp from seas.smu.edu:pub or sbcs.sunysv.edu:pub/XSB.

[Chen and Warren, 1994] W. Chen and D.S. Warren. Computation of stable models and its intergration with logical query processing. Available by ftp from ftp.ms.uky.edu:pub/lpnmr/chen2.ps, 1994.

[Dowling and Gallier, 1984] 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.

[Eshghi, 1990] K. Eshghi. Computing stable models by using the ATMS. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 272–277, Boston, MA, USA, July 1990. The MIT Press.

[Gelfond and Lifschitz, 1988] 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.

[Gottlob, 1992] G. Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Junker and Konolige, 1990] U. Junker and K. Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 278–283, Boston, MA, USA, July 1990. The MIT Press.

[Knuth, 1993] D.E. Knuth. Stanford GraphBase, 1993. Available by ftp from labrea.stanford.edu.

[Niemelä and Rintanen, 1992] I. Niemelä and J. Rintanen. On the impact of stratification on the complexity of nonmonotonic reasoning. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 627–638, Cambridge, MA, USA, October 1992. Morgan Kaufmann Publishers.

[Niemelä, 1994] I. Niemelä. A decision method for nonmonotonic reasoning based on autoepistemic reasoning. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484, Bonn, Germany, May 1994. Morgan Kaufmann Publishers.

[Niemelä, 1995] I. Niemelä. Towards efficient default reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, Montreal, Canada, August 1995. Morgan Kaufmann Publishers. Accepted to be published.

[Reiter, 1980] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[Risch and Schwind, 1994] V. Risch and C. Schwind. Tableau-based characterization and theorem proving for default logic. *Journal of Automated Reasoning*, 13:223–242, 1994.

[Satoh and Iwayama, 1991] K. Satoh and N. Iwayama. Computing abduction by using the TMS. In *Proceedings of the 8th International Conference on Logic Programming*, pages 505–518, Paris, France, June 1991. The MIT Press.