Automating Default Reasoning*

Ilkka Niemelä
Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O. Box 5400, FIN-02015 HUT, Finland
`Ilkka.Niemela@hut.fi`
www.tcs.hut.fi/~ini

Contents

*The work has been supported by the Academy of Finland through Project 43963.

1 Introduction

This article addresses the question of automating Reiter's default logic [Rei80] which is one of the oldest and most well-studied nonmonotonic logics. Default logic is an extension of classical logic where new kinds of inference rules, *default rules*, are added to classical reasoning. The novelty in default rules is that they are *nonmonotonic*, i.e., a default rule which is applicable for a set of premises might become inapplicable when the set of premises is extended. Such nonmonotonic rules are useful for many kinds of knowledge representation tasks such as capturing rules with exceptions, frame axioms and default values. Default logic is closely related to other major nonmonotonic logics such as circumscription [McC80, Lif85], nonmonotonic modal logics [MD80, MST93] and autoepistemic logic [Moo85]. It turns out that default logic is quite expressive and can capture other nonmonotonic logics in an intuitive and modular way, see e.g. [Jan98] and the references there for interesting translations.

The rest of the article is organized as follows. First we introduce Reiter's default logic. Then we briefly survey known decidability and complexity results for default logic and discuss the implications of the results on automated default reasoning. Given this background we move towards automated theorem proving methods. The first issue is the problem of developing *finitary* characterizations of default reasoning. This forms the basis of the decision methods studied in Section 5. For many practical applications a limited subclass of default logic corresponding to logic programs is sufficient. Section 6 discuss disjunctive logic programs and their relation to default logic. In Section 7 we show how the decision method developed for default logic in Section 5 can be further optimized to provide the basis for one of the most efficient current implementations of the stable model semantics for normal (non-disjunctive) logic programs, the **smodels** system [NS96, NS97].

2 Default Logic

Default logic introduced by Reiter [Rei80] is one of the most well-known nonmonotonic logics. Default logic assumes knowledge to be represented in terms of a default theory. A default theory is a pair $(\mathcal{D}, W)$ where W is a set of first-order formulae representing the facts which are known to be true with certainty and $\mathcal{D}$ is a set of defaults of the form

$$\frac{\varphi : \psi_1, ..., \psi_n}{\gamma}$$

where φ, ψ_i and γ are classical formulae. We will also frequently use the alternative, less space consuming notation $\varphi{:}\psi_1, \ldots, \psi_n/\gamma$ for a default. The default has the intuitive reading:

if φ is derivable and, for all i $(1 \leq i \leq n)$, $\neg\psi_i$ is not derivable, then derive γ.

The formula φ is called the *prerequisite*, ψ_i a *consistency condition* or *justification*, and γ the *consequent* of the default. For a default d we use $\mathrm{pre}(d)$, $\mathrm{just}(d)$, and $\mathrm{cons}(d)$ to denote the prerequisite, the set of justifications, and the consequent of d, respectively. Open defaults, i.e., defaults with free variables, are usually interpreted as schemata representing all of their closed instances.[1]

A default theory induces so-called *extensions* which represent acceptable belief sets a reasoner may adopt based on the theory. A formula φ is called a *sceptical* consequence of a

[1] Reiter treats open defaults somewhat differently and uses a more complicated method to define extensions for them. We refer to [Kam95] for a detailed investigation.

default theory $(\mathcal{D}, W)$ *if and only if* φ is contained in all extensions of $(\mathcal{D}, W)$ and a *credulous* consequence of $(\mathcal{D}, W)$ *if and only if* φ is contained in at least one extension of $(\mathcal{D}, W)$.

We will first present a definition of extensions due to Marek and Truszczyński [MT93] which is slightly different from (but equivalent to) Reiter's original definition. This definition seems to be a bit easier to apply than the original one which we will also present.

Intuitively, E is an extension of $(\mathcal{D}, W)$ *if and only if* E is a deductively closed (in the sense of classical logic) set containing W and satisfying the following two properties

1. all defaults that are "applicable" with respect to E have been applied,
2. every formula in E has a "derivation" from W and applicable defaults.

To make the two requirements more precise we introduce the following notion of a closure with respect to classical consequence and a set R of monotonic inference rules of the form

$$\frac{\varphi}{\gamma}.$$

For such a rule r we denote φ by $\mathrm{pre}(r)$ and γ by $\mathrm{cons}(r)$. By $\mathrm{Cn}_R(W)$ we mean the closure of W with respect to classical consequence and the monotonic rules R. This closure can be captured using a notion of a proof as follows. A formula φ belongs to the closure if and only if it has a proof from R with W, i.e., there is a finite sequence of rules $r_1, \ldots, r_n$ in R such that

1. $W \cup \{\mathrm{cons}(r_1), \ldots, \mathrm{cons}(r_{i-1})\} \models \mathrm{pre}(r_i)$, for $i \in \{1, \ldots, n\}$,
2. $W \cup \{\mathrm{cons}(r_1), \ldots, \mathrm{cons}(r_n)\} \models \varphi$.

As usual $\models$ denotes the classical consequence relation. We now can state the definition of extensions formally:

Definition 2.1 (Extension 1)
Let $(\mathcal{D}, W)$ be a default theory. Then E is an extension of $(\mathcal{D}, W)$ if and only if $E = \mathrm{Cn}_{\mathcal{D}_E}(W)$ where

$$\mathcal{D}_E = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D}, \textit{for all } \psi \in \mathrm{just}(d), \neg\psi \notin E\}.$$

The set $\mathcal{D}_E$ can be seen as (the monotonic part of) the potentially applicable rules in $\mathcal{D}$ given E. Hence, the equation for E requires that every formula derivable by the potentially applicable rules must be included in the extension and every formula in the extension is derivable by such rules.

Reiter's [Rei80] equivalent original definition gives extensions as fixed points of a certain other operator which combines classical logical closure and default rules. We denote the classical closure of W by $\mathrm{Cn}(W) = \{\varphi \mid W \models \varphi\}$.

Definition 2.2 (Extension 2)
Let $(\mathcal{D}, W)$ be a default theory, S a set of formulae. Let $\Gamma(S)$ be the smallest set such that:

1. $W \subseteq \Gamma(S)$,
2. $\mathrm{Cn}(\Gamma(S)) = \Gamma(S)$,
3. *if* $\varphi{:}\psi_1, \ldots, \psi_n/\gamma \in \mathcal{D}$, $\varphi \in \Gamma(S)$, $\neg\psi_i \notin S$ $(1 \leq i \leq n)$, *then* $\gamma \in \Gamma(S)$.

E is an extension of $(\mathcal{D}, W)$ if and only if *$E = \Gamma(E)$, that is, if E is a fixed point of Γ.*

We finally give a third, quasi-inductive characterization of extensions, also due to Reiter [Rei80]. This version is often used in proofs about default logic and it makes the way in which formulae have to be grounded in the premises more explicit. Let E be a set of formulae and define, for a given default theory $(\mathcal{D}, W)$, a sequence of sets of formulae as follows:

$$E_0 = W, \text{ and for } i \geq 0$$
$$E_{i+1} = \mathrm{Cn}(E_i) \cup \{\gamma \mid \varphi{:}\psi_1, \ldots, \psi_n/\gamma \in \mathcal{D}, \varphi \in E_i, \neg\psi_i \notin E\}.$$

It can be shown that E is an extension of $(\mathcal{D}, W)$ *if and only if* $E = \bigcup_{i=0}^{\infty} E_i$. The appearance of E in the definition of E_{i+1} is what renders this alternative definition of extensions non-constructive.

Default theories may have several extensions or even none at all. Extensions are always consistent if W is and if there are no defaults without consistency conditions. If W is inconsistent, then the single extension of $(\mathcal{D}, W)$ is the set of all formulae. Extensions are minimal in the following sense: if E is an extension, then there is no extension E' such that $E' \subset E$.

3 Decidability and Complexity

The fundamental decision problems for default logic are *sceptical reasoning* (i.e. determining sceptical consequences), *credulous reasoning* (i.e. determining credulous consequences) and the problem of deciding whether a set of premises has at least one extension (the *extension existence problem*).

For first-order default logic these problems are not even semi-decidable [Rei80]. For propositional default logic, extension existence problem and credulous reasoning are $\mathbf{\Sigma_2^P}$-complete problems in the polynomial time hierarchy and sceptical default reasoning (the dual problem) is $\mathbf{\Pi_2^P}$-complete [Got92, Sti92]. Balcázar et al. [BDG88] and Garey and Johnson [GJ79] provide good sources of basic definitions and results on the polynomial time hierarchy and related issues in complexity theory. Kautz and Selman [KS91], Stillman [Sti90], and Ben-Eliyahu and Dechter [BED91, BED92] have obtained further complexity results on restricted subclasses of default theories.

Implications of the Results

The complexity results have interesting implications on theorem proving methods for default logic which we elaborate next. As mentioned above, the full first-order case is not even semi-decidable as is the case for classical first-order logic. Hence, theorem proving systems for first-order default logic cannot provide a similar level of completeness as classical theorem provers: a query can be a nonmonotonic consequence of the premises but no nonmonotonic theorem prover is able to establish this. This should be compared to first-order theorem proving where it can be guaranteed that for each valid formula a proof is eventually found.

If we consider the propositional case, default logic is decidable and the most important decision problems (sceptical and credulous consequence) are complete problems with respect to the second level of the polynomial time hierarchy. First, this implies that this kind of nonmonotonic reasoning can be implemented in *polynomial space*. Second, this indicates that nonmonotonic reasoning is strictly harder than classical reasoning unless the polynomial time hierarchy collapses which is regarded unlikely by the experts. This means that two orthogonal sources of complexity are involved in general nonmonotonic reasoning. One source originates from the classical reasoning on top of which nonmonotonic reasoning is built. The other source is related to the handling of the nonmonotonic constructs. These sources are independent of each other. In fact, even if we assume that we can use an oracle for classical reasoning, i.e., we are able to decide classical consequence in one computation step, solving a nonmonotonic reasoning problem remains on the difficulty level of an **NP/co-NP**-complete problem and no polynomial time algorithms are known even under the oracle assumption. Hence, it is highly unlikely that general nonmonotonic reasoning can be implemented on top of a classical theorem proving method with only a polynomial overhead.

Tractable Subclasses

The complexity results provide important information when searching for tractable subclasses of nonmonotonic reasoning problems. In classical logics the standard approach to achieving tractability is to restrict the syntactic form of formulae, e.g., to consider only Horn clauses or clauses with only two literals. The complexity results indicate that this is not enough for nonmonotonic reasoning because only the complexity of the required classical reasoning is affected. The complexity related to the handling of the nonmonotonic constructs is orthogonal to this and can still lead to intractability. Hence, in order to achieve tractability both sources of complexity have to be addressed. A standard technique is to restrict the syntactic form of the premises (for guaranteeing the tractability of the required classical reasoning) and to limit the interaction of the nonmonotonic constructs. The interaction can be controlled by syntactic restrictions such as stratification introduced in the logic programming setting [CH85, ABW88, VG88]. We refer to [NR94] for a detailed investigation of the effects of stratification on the complexity of the nonmonotonic reasoning and for the associated decision methods and algorithms. Another approach to control the interaction is to use a weaker semantics. Examples of this approach are the well-founded semantics for logic programs [VGRS91] and the stationary semantics for default logic [PP94, Got95]. These kinds of restrictions cut the cost of implementing nonmonotonic reasoning but also limit the expressive power. This raises interesting questions about suitable trade-offs between expressive power and computational complexity because recent results show that general nonmonotonic reasoning techniques lead to very compact representation of knowledge and, e.g., exponentially more succinct representations than when using classical logic [GPSK95, CDSS97].

4 Finitary Characterization of Default Reasoning

The definition of default reasoning is given in terms of infinite objects: a fixed point equation over infinite sets of formulae. For developing a computational treatment of the logic, it is necessary to find techniques for characterizing correct reasoning using finitary means. These characterizations provide the basis for the decidability and complexity results as well as for the automated theorem proving methods. In this section we elaborate the definition of default logic and present finitary characterizations of extensions with the aim of providing the basis for automating default reasoning. In the following section we demonstrate how they lead to efficient decision methods.

The key objects of interest in default logic are extensions of a default theory. Reiter [Rei80] made an important observation already in his original article: extensions can be captured in terms of the consequents of the default rules. Every extension of a default theory $(\mathcal{D}, W)$ can be represented as the logical closure of the classical formulae W and the consequents of a subset GD of the defaults $\mathcal{D}$ (the generating defaults).

Theorem 4.1 (Generating Defaults)
Each extension of a default theory $(\mathcal{D}, W)$ *has a set of generating defaults* $\mathrm{GD} \subseteq \mathcal{D}$*, i.e., if*

E is an extension of $(\mathcal{D}, W)$, then $E = \mathrm{Cn}(W \cup \{\mathrm{cons}(d) \mid d \in \mathrm{GD}\})$ *where* $\mathrm{GD} = \{d \in \mathcal{D} \mid \mathrm{pre}(d) \in E$ *and for all* $\psi \in \mathrm{just}(d), \neg\psi \notin E\}$.

Hence, to find an extension we need to identify the generating defaults. We present two approaches to characterizing the generating defaults. We will use the notation Cn_R which stands for the consequence relation induced by a set R of monotonic rules.

The first characterization is based on examining default rules directly. This approach has been presented in several papers [MT93, RS94, BH95]. The idea is to project the conditions fulfilled by an extension to individual default rules. First, extensions are grounded which means that the corresponding set of generating defaults must be grounded in the following sense. A set of defaults $\mathcal{D}$ is grounded in a set formulae W if and only if for all $d \in \mathcal{D}$, $\mathrm{pre}(d) \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D})}(W)$ where $\mathrm{Mon}(\mathcal{D}) = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D}\}$. Second, extensions are closed under default rules, i.e. all applicable defaults are applied.

Theorem 4.2 (Extensions in Terms of Generating Defaults)
A set of formulae E is an extension of a default theory $(\mathcal{D}, W)$ if and only if $E = \mathrm{Cn}(W \cup \{\mathrm{cons}(d) \mid d \in \mathcal{D}'\})$ *for a set of defaults $\mathcal{D}' \subseteq \mathcal{D}$ such that*

1. *$\mathcal{D}'$ is grounded in W and*
2. *for all $d \in \mathcal{D}$:*

$$\begin{aligned} d \in \mathcal{D}' \text{ if and only if } & \mathrm{pre}(d) \in \mathrm{Cn}(W \cup \{\mathrm{cons}(d) \mid d \in \mathcal{D}'\}) \textit{ and} \\ & \textit{for all } \psi \in \mathrm{just}(d), \neg\psi \notin \mathrm{Cn}(W \cup \{\mathrm{cons}(d) \mid d \in \mathcal{D}'\}). \end{aligned}$$

The second characterization introduced in [Nie95] is based on justifications. Here the idea is that default rules are inference rules guarded with consistency conditions given by the justifications. Hence, it is the set of justifications that determines the extension and the rest is just monotonic rule-based derivation.

We denote by $\mathrm{just}(\mathcal{D})$ the set of all justifications in the set of defaults $\mathcal{D}$ and for a set of formulae F, $\mathrm{Mon}(\mathcal{D}, F) = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D}, \mathrm{just}(d) \subseteq F\}$. A set of justifications corresponding to an extension is called *full* with respect to the default theory.

Definition 4.3 (Full Sets)
For a default theory $(\mathcal{D}, W)$, a set of justifications $F \subseteq \mathrm{just}(\mathcal{D})$ is called $(\mathcal{D}, W)$-full if and only if the following condition holds:

$$\textit{for all } \psi \in \mathrm{just}(\mathcal{D}){:}\quad \psi \in F \quad \textit{if and only if } \neg\psi \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$$

For each full set there is a corresponding extension and for each extension a full set that induces it.

Theorem 4.4 (Extensions in Terms of Full Sets)
Let $(\mathcal{D}, W)$ a default theory.

1. *If $F \subseteq \mathrm{just}(\mathcal{D})$ is $(\mathcal{D}, W)$-full, then $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$ is an extension of $(\mathcal{D}, W)$.*
2. *If E is an extension of $(\mathcal{D}, W)$, then $F = \{\psi \in \mathrm{just}(\mathcal{D}) \mid \neg\psi \notin E\}$ is $(\mathcal{D}, W)$-full and $E = \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$.*

Example 1 *Consider a default theory* $(\mathcal{D}, W)$ *where* $\mathcal{D} = \{p{:}\neg s/r, q{:}\neg r/s\}$ *and* $W = \{q\}$. *The* $(\mathcal{D}, W)$*-full sets are subsets of* $\mathrm{just}(\mathcal{D}) = \{\neg s, \neg r\}$. *Consider* $F = \{\neg r\}$. *It is* $(\mathcal{D}, W)$*-full because* $\mathrm{Mon}(\mathcal{D}, F) = \{\frac{q}{s}\}$, $\neg\neg r \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$ *but* $\neg\neg s \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$. *Hence,* $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W) = \mathrm{Cn}(W \cup \{s\})$ *is an extension of* $(\mathcal{D}, W)$. *For instance,* $F' = \{\neg s\}$ *is not* $(\mathcal{D}, W)$*-full, because* $\mathrm{Mon}(\mathcal{D}, F') = \{\frac{p}{r}\}$ *and, e.g.,* $\neg\neg r \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F')}(W)$ *but* $\neg r \notin F'$. *It is easy to verify that* F *is the only* $(\mathcal{D}, W)$*-full set and, thus,* $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},F)}(W)$ *is the unique extension of* $(\mathcal{D}, W)$.

5 A Decision Method for Default Logic

There are already quite a number of decision methods for default reasoning. Methods based on the characterization of extensions in terms of generating defaults (Theorem 4.1) can be found, e.g., in [MT93, BH95, RS94]. There are approaches where default reasoning is reduced into another problem like a truth maintenance problem [JK90] or a constraint satisfaction problem [BED91]. An interesting approach to provide proof theory for default reasoning based on sequent calculus is proposed by [Bon96, BO97]. A novelty in this approach is that it includes an explicit proof theoretic treatment of non-validity needed in default reasoning.

The development in this section is based on the second characterization of extensions in terms of full sets and originates from the work in [Nie95]. The aim is to develop a decision method for default reasoning by employing ideas similar those used in the Davis-Putnam procedure which is one of the most successful methods for deciding propositional satisfiability.

We start by presenting first a naive decision procedure for default logic and then show how to optimize it in a systematic way. Fig. 1 presents the naive decision procedure for both credulous and sceptical default reasoning. The idea is that when the procedure terminates, then for a default theory $(\mathcal{D}, W)$ and formula φ, the variable S-answer is set to Yes if and only if φ is a sceptical consequence of $(\mathcal{D}, W)$, i.e., in every extension of $(\mathcal{D}, W)$ and the variable C-answer is set to Yes if and only if φ is a credulous consequence of $(\mathcal{D}, W)$. Using Theorem 4.4 this is easy to show to be the case. Under what conditions is the procedure implementable, i.e., when are all subtasks decidable? It turns out that it is enough to require that (i) the default theory is finite and (ii) the first-order logic of W and the first-order formulae appearing in the defaults is decidable. Then there is a finite number of justifications and, thus, finitely many subsets of justifications. Hence, the main loop is iterated only a finite number of times. Decidability of the first-order logic implies the decidability of the membership in the closure $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},S)}(\cdot)$ and this implies the decidability of the fullness test. This means that each iteration of the main loop is computable. These two conditions imply the implementability of the optimized version of the decision procedure to be developed and they are assumed for the rest of the section.

The naive decision procedure is not very efficient because for a default theory with n justifications, the main loop is iterated 2^n times and each iteration involves several calls to a classical theorem prover. We start optimizing the procedure using the following ideas which are similar to those found in the Davis-Putnam procedure for propositional satisfiability (see, e.g., [Fit90]). (i) We employ a bottom-up backtracking search over candidate full sets, (ii) we prune the search space by making assumptions about the justifications one at a time and (iii) before making a new assumption, an *approximation* of the extensions agreeing with the current assumptions is computed and *conflicts* are detected. Fig. 2 presents the skeleton of the optimized decision procedure given as a function **extensions**. It employs four functions

- **expand** (for approximations),
- **conflict** (for conflict detection),

```
S-answer:= Yes; C-answer:= No;
For every subset S of just(D) do
  If S is (D,W)-full then
    If φ ∈ Cn_Mon(D,S)(W) then
      C-answer:= Yes
    else
      S-answer:= No
    end if
  end if
```

Figure 1: A Naive Decision Procedure for Credulous and Sceptical DL

```
function extensions((D,W), FS, φ)
FS' := expand((D,W), FS);
if conflict((D,W), FS') returns Yes then return No
else if FS' covers the negation of each justification in D then
    return test((D,W), FS', φ)
else
    χ := choose((D,W), FS');
    if extensions((D,W), FS' ∪ {χ}, φ) returns Yes then return Yes
    else return extensions((D,W), FS' ∪ {compl(χ)}, φ)
    end if
end if
```

Figure 2: An optimized decision procedure for default logic

- **choose** (for search heuristics) and
- **test** (for testing extensions).

The first three functions identify the key subtasks of the procedure where optimizations are applied. By changing the **test** function, the procedure can be adapted to yield decision procedures for sceptical and credulous reasoning as well as for checking the existence of extensions. We will explain the role of the functions in more detail but first we discuss the representation of a full set in the procedure and its overall idea.

In the algorithm we need to represent a partially constructed full set and also explicate assumptions about which justifications are not included in the full set. Here we reverse the representation and describe a full set by speaking about the membership in the extension. With this representation it is easier to discuss the optimizations and the application of the method to logic programming semantics becomes more straightforward in Section 7.

As given in Definition 4.3, a full set F characterizing an extension E is a set of justifications such that for each justification ψ in the default theory, $\psi \in F$ if and only if $\neg\psi \notin E$. Our idea is to represent a partially constructed full set by using a set FS which is built from the *negations* of justifications and which explicates the current assumptions about their membership in the extension under consideration. For representing the fact that a formula φ is not in the extension we employ what we call a not-formula which is just an expression of the form $\mathrm{not}(\varphi)$. So an expression $\mathrm{not}(\neg\chi)$ in FS indicates that χ belongs to the full set to be constructed (and $\neg\chi$ does not belong to the corresponding extension). On the other hand, a formula $\neg\chi$ in FS

indicates that χ does not belong to the full set under consideration (and $\neg\chi$ belongs to the corresponding extension).

Hence, a full set containing the justifications $\neg p, q$ but not containing the justifications $\neg r, s$ is represented as the set $\{\text{not}(\neg\neg p), \text{not}(\neg q), \neg\neg r, \neg s\}$. Note that for the corresponding extension E holds that, e.g., $\neg\neg p \notin E$ but $\neg s \in E$. On the other hand, when we speak about a full set represented in this way, we naturally mean the set of justifications which are the complements of formulae inside the not$(\cdot)$ operator. For instance, when we discuss the full set

$$FS = \{\text{not}(\neg\neg p), \text{not}(\neg q), \text{not}(\neg\neg r), \neg s\},$$

what is actually meant is the set $F = \{\neg p, q, \neg r\}$. Hence, we occasionally use a full set FS represented in this way instead of a set F represented in the original way. For instance, we write $\text{Mon}(\mathcal{D}, FS)$ which denotes the set

$$\{\frac{pre(d)}{cons(d)} \mid d \in \mathcal{D}, \text{ for all } \psi \in \text{just}(d), \text{not}(\neg\psi) \in FS\}.$$

For discussing the procedure **extensions** we employ the following concepts:

- We say that a set of formulae S *agrees* with a set of formulae and not-formulae FS if for every ordinary formula $\varphi \in FS$, $\varphi \in S$ and for all $\text{not}(\varphi) \in FS$, $\varphi \notin S$. For example, the set $\{a, b\}$ agrees with $\{a, \text{not}(c)\}$ but not with $\{a, \text{not}(b)\}$.

- A set of formulae and not-formulae FS *covers* a formula φ if either $\varphi \in FS$ or $\text{not}(\varphi) \in FS$. For example, the set $\{\text{not}(a), b\}$ covers a. A set of formulae S is covered by FS if each formula in S is covered by FS.

- By $\text{compl}(\chi)$ we denote the not-complement of an expression: for an ordinary formula χ, $\text{compl}(\chi) = \text{not}(\chi)$ and $\text{compl}(\text{not}(\chi)) = \chi$.

The function **extensions** takes as input a default theory $(\mathcal{D}, W)$, and a set FS which determines the constraints on the extensions to be considered and a formula φ which is just passed as an argument to the function **test**. The function **extensions** performs *focused extension search*, i.e., given some conditions on the extensions of interest in terms of a set FS of ordinary formulae (to be included in the extension) and not-formulae (to be excluded from the extension), **extensions** focuses its search to these extensions to find one that passes the **test** function. For a default theory $(\mathcal{D}, W)$ and a formula φ, **extensions**$((\mathcal{D}, W), FS, \varphi)$ returns Yes if and only if there is an extension E agreeing with FS such that **test**$((\mathcal{D}, W), FS', \varphi)$ returns Yes where FS' is the full set of E. This is accomplished by constructing $(\mathcal{D}, W)$-full sets agreeing with FS until a full set FS' is found such that **test**$((\mathcal{D}, W), FS', \varphi)$ returns Yes in the following way.

The function **extensions** starts by expanding cautiously the set FS. The idea is that no extensions are lost in the expansion: every extension agreeing with the original FS agrees also with FS'. Then a conflict test is performed. If there are no conflicts and FS' covers the negations of justifications in $(\mathcal{D}, W)$, FS' is a full set for an extension agreeing with FS and **extensions** returns what **test**$((\mathcal{D}, W), FS', \varphi)$ returns. If there are negations of justifications in $(\mathcal{D}, W)$ not covered by FS', then **choose** selects one of them, say ψ, and returns either ψ or $\text{not}(\psi)$. For ψ there are two possibilities: either ψ belongs to the extension to be constructed or not. The two alternatives are covered by backtracking search. The decision in **choose** whether ψ or $\text{not}(\psi)$ is returned, determines in which order the alternatives are examined. Extensions containing ψ are covered by calling **extensions**$((\mathcal{D}, W), FS' \cup \{\psi\}, \varphi)$ and extensions not containing ψ are covered by calling **extensions**$((\mathcal{D}, W), FS' \cup \{\text{not}(\psi)\}, \varphi)$.

Soundness and completeness of the procedure can be guaranteed under quite weak assumptions about the three functions.

choose: $\mathbf{choose}((\mathcal{D},W),FS)$ returns either $\neg\psi$ or $\mathrm{not}(\neg\psi)$ for some justification ψ in $(\mathcal{D},W)$ for which $\neg\psi$ is not covered by FS.

expand: $\mathbf{expand}((\mathcal{D},W),FS)$ returns a set FS' of formulae and not-formulae which extends FS and with which every extension of $(\mathcal{D},W)$ agreeing with FS agrees.

conflict: (i) If $\mathbf{conflict}((\mathcal{D},W),FS)$ returns Yes, there is no extension agreeing with FS and (ii) if FS covers the negation of each justification in $(\mathcal{D},W)$, and for some $\mathrm{not}(\chi)\in FS$, $\chi\in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W)$ or for some $\chi\in FS$, $\chi\notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W)$, $\mathbf{conflict}((\mathcal{D},W),FS)$ returns Yes.

Under these assumptions we can establish the following theorem.

Theorem 5.1 (Soundness and completeness)
Let $(\mathcal{D},W)$ be a default theory and FS a set of formulae and not*-formulae. Then the function* $\mathbf{extensions}((\mathcal{D},W),FS,\varphi)$ *returns Yes if and only if there exists an extension of $(\mathcal{D},W)$ agreeing with FS such that* $\mathbf{test}((\mathcal{D},W),FS',\varphi)$ *returns Yes where FS' is the corresponding full set.*

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 extensions to be constructed and (ii) choosing an appropriate **test** function as shown in the following corollary.

Corollary 5.2 *Let $(\mathcal{D},W)$ be a default theory and φ a formula.*

1. *Let* $\mathbf{test}((\mathcal{D},W),FS,\varphi)$ *return Yes for all $(\mathcal{D},W)$, FS, and φ.*
   Then $\mathbf{extensions}((\mathcal{D},W),\emptyset,\varphi)$ *returns Yes if and only if $(\mathcal{D},W)$ has an extension.*
2. *Let* $\mathbf{test}((\mathcal{D},W),FS,\varphi)$ *return Yes if and only if $\varphi\in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W)$.*
   Then $\mathbf{extensions}((\mathcal{D},W),\emptyset,\varphi)$ *returns Yes if and only if there exists an extension of $(\mathcal{D},W)$ containing φ.*
3. *Let* $\mathbf{test}((\mathcal{D},W),FS,\varphi)$ *return Yes if and only if $\varphi\notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W)$.*
   Then $\mathbf{extensions}((\mathcal{D},W),\emptyset,\varphi)$ *returns Yes if and only if there exists an extension of $(\mathcal{D},W)$ not containing φ (i.e.,* $\mathbf{extensions}((\mathcal{D},W),\emptyset,\varphi)$ *returns No if and only if φ is contained in every extension of $(\mathcal{D},W)$).*

In order to provide a systematic basis for developing implementations of **expand** and **conflict** we introduce two concepts: a lower bound and an upper bound of a set of extensions. The idea is that given a default theory $(\mathcal{D},W)$ and some conditions FS on the extensions, all the extensions satisfying the conditions agree with the lower bound and are contained in the upper bound:

lower bound: $\mathbf{LB}((\mathcal{D},W),FS)$ is characterized with the property: every extension E of $(\mathcal{D},W)$ agreeing with FS agrees with $\mathbf{LB}((\mathcal{D},W),FS)$ and

upper bound: $\mathbf{UB}((\mathcal{D},W),FS)$ satisfies the property: for every extension E of $(\mathcal{D},W)$ which agrees with FS, $E\subseteq \mathbf{UB}(P,FS)$.

Using these concepts we can implement the two functions in a unified way. In **expand** the idea is to extend the constraints FS by all negations of the justifications which are in the lower bound and by $\text{not}(\neg\varphi)$ for all $\neg\varphi$ not in the upper bound. Of course, this can be iterated which leads to the following implementation:

> $\mathbf{expand}((\mathcal{D}, W), FS)$ returns the smallest set of formulae FS' extending FS and closed under the following rules:
>
> - if $\neg\varphi \in \mathbf{LB}((\mathcal{D}, W), FS')$ for $\varphi \in \text{just}(\mathcal{D})$, then include $\neg\varphi$ in FS' and
> - if $\neg\varphi \notin \mathbf{UB}((\mathcal{D}, W), FS')$ for $\varphi \in \text{just}(\mathcal{D})$, then include $\text{not}(\neg\varphi)$ in FS'.

In **conflict** the idea is to check the requirements in FS against the lower and upper bounds:

> $\mathbf{conflict}((\mathcal{D}, W), FS)$ returns Yes if and only if for some formula φ,
>
> $$\begin{array}{l} \text{not}(\varphi) \in FS \text{ but } \varphi \in \mathbf{LB}((\mathcal{D}, W), FS) \text{ or} \\ \varphi \in FS \text{ but } \varphi \notin \mathbf{UB}((\mathcal{D}, W), FS). \end{array}$$

It should be noted that the soundness and completeness of the decision procedure using the above implementations of the two functions is not guaranteed by the characterizing properties of the bounds. This is because the properties of the bounds do not imply the second soundness and completeness condition for **conflict**.

Example 2 *Consider the trivial bounds where $\mathbf{LB}((\mathcal{D}, W), FS)$ is the empty set and the upper bound $\mathbf{UB}((\mathcal{D}, W), FS)$ is the whole language, a default theory $\mathcal{D} = \{:p/p\}$, $W = \{\neg p\}$ and a set $FS = \{\text{not}(\neg p)\}$. Clearly, the second condition for* **conflict** *does not hold: FS covers negations of the justifications in $\mathcal{D}$, $\text{not}(\neg p) \in FS$ but $\neg p \in \text{Cn}_{\text{Mon}(\mathcal{D},FS)}(W)$. However, given the trivial bounds, $\mathbf{conflict}((\mathcal{D}, W), FS)$ does not return the answer Yes.*

In order to have full freedom in developing lower and upper bounds, we assume that the implementation of **conflict** performs a *post-check*:

> when FS covers the negation of each justification in $(\mathcal{D}, W)$, $\mathbf{conflict}((\mathcal{D}, W), FS)$ returns Yes if for some $\text{not}(\chi) \in FS$, $\chi \in \text{Cn}_{\text{Mon}(\mathcal{D},FS)}(W)$ or for some $\chi \in FS$, $\chi \notin \text{Cn}_{\text{Mon}(\mathcal{D},FS)}(W)$.

and otherwise it functions as described above. With this addition the soundness and completeness of **extensions** is guaranteed for any upper and lower bound satisfying the respective characterizing properties.

Example 3 *We illustrate the idea of the decision procedure by a simple example using the trivial implementation, i.e., where* **expand** *and* **conflict** *are implemented using the trivial bounds. This actually is an implementation of the naive method in Fig. 1.*

Consider a default theory $(\mathcal{D}, W)$ where

$$\mathcal{D} = \{\frac{:p}{\neg q}, \frac{:q}{\neg p}\}, W = \{\neg p \rightarrow r\}$$

and the task of deciding whether r is a sceptical consequence of $(\mathcal{D}, W)$. Hence, the function $\mathbf{test}((\mathcal{D}, W), FS, \varphi)$ returns Yes if and only if $\varphi \notin \text{Cn}_{\text{Mon}(\mathcal{D},FS)}(W)$ and the decision procedure is started by a call to $\mathbf{extensions}((\mathcal{D}, W), \emptyset, r)$. First $FS = \emptyset$ is extended by $\mathbf{expand}((\mathcal{D}, W), \emptyset)$ but as the trivial bounds are used, nothing new is discovered and, similarly, no conflicts are *detected. Hence,* $\mathbf{choose}((\mathcal{D},W),\emptyset)$ *is called. Here the alternatives are in* $\mathrm{just}((\mathcal{D},W)) = \{p,q\}$*. Assume that* p *is selected and* $\mathrm{not}(\neg p)$ *is returned by* $\mathbf{choose}((\mathcal{D},W),\emptyset)$*. Then*

$$\mathbf{extensions}((\mathcal{D},W),\{\mathrm{not}(\neg p)\},r)$$

is called and because of the trivial bounds the full set is not extended nor are conflicts detected. Hence $\mathbf{choose}((\mathcal{D},W),\{\mathrm{not}(\neg p)\})$ *is called. Assume it returns* $\mathrm{not}(\neg q)$*. Hence,*

$$\mathbf{extensions}((\mathcal{D},W),\{\mathrm{not}(\neg p),\mathrm{not}(\neg q)\},r)$$

is called. Now all the negations of the justifications in $(\mathcal{D},W)$ *are covered and the post-check is done. Here conflicts are found:* $\{\neg p,\neg q\} \subseteq \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W)$ *where* $FS = \{\mathrm{not}(\neg p),\mathrm{not}(\neg q)\}$*.*

Hence, the procedure backtracks and $\mathbf{extensions}((\mathcal{D},W),\{\mathrm{not}(\neg p),\neg q\},r)$ *is called. Now* $\{\mathrm{not}(\neg p),\neg q\}$ *covers the negations of the justifications in* $\mathcal{D}$ *and the post-check is passed. Thus,* $\{\mathrm{not}(\neg p),\neg q\}$ *is a* $(\mathcal{D},W)$*-full set and*

$$\mathbf{test}((\mathcal{D},W),\{\mathrm{not}(\neg p),\neg q\},r)$$

is called. It returns Yes which implies that $\mathbf{extensions}((\mathcal{D},W),\emptyset,r)$ *returns Yes. Hence,* r *is not a sceptical consequence of* $(\mathcal{D},W)$*.*

The efficiency of the decision procedure **extensions** is determined by the implementations of the three functions **expand**, **conflict**, and **choose**. The decision procedure performs search over extensions and the size of this search space has a crucial effect on the performance of the algorithm. The size depends on the implementations of **expand** and **conflict** as well as on the search heuristics (implemented by **choose**). The search heuristics can have a dramatic effect on the efficiency because the right choices can lead to the desired extension immediately. Even in the case where there are no extensions agreeing with the given requirements, the order in which the search space is examined can effect significantly the size of the space. We discuss the choice of a good search heuristics in more detail in Section 7 where we apply this method to logic programming.

The role of the functions **expand** and **conflict** is to prune the search space by removing the choice points (**expand**) and cutting useless search when no solutions are left (**conflict**). Exponential savings can be achieved. For example, if **expand** extends the full set by one formula, this cuts the remaining search space by half. Similarly, detecting a conflict early in the search yields an exponential reduction on the size of the search space. Hence, good implementations of the functions can speed-up the decision procedure considerably. There is an interesting trade-off between the effort spent in the two functions and the size of the remaining search space. This leads to the question of deciding how much computation resources should be invested in the two pruning functions and how much in the actual search.

We have unified the development of the two functions by adopting the concepts of a lower and an upper bound. Hence, instead of discussing the implementations of the two functions separately, we can focus on finding good implementation techniques for the upper and lower bounds. Our approach to the trade-off between pruning efforts and search is to consider implementations of the bounds where a low polynomial number of calls to a decision procedure for classical monotonic derivation are needed. In the following we speak about these calls as theorem prover calls for short. Why is this a reasonable framework for developing the bounds? First, theorem prover calls are needed no matter how weak bounds are used because the soundness and completeness of the method requires the post-check where theorem prover calls are employed. Second, it seems unreasonable to use a high (e.g., exponential) number of classical theorem prover calls in the pruning functions because then the savings compared to performing the actual search are not clear.

Before we go into developing the bounds, let us consider the sources of complexity in the decision procedure and examine whether these are consistent with the complexity results. In the procedure there are two sources of complexity. One is the classical theorem proving task for implementing the upper and lower bounds and the other source is related to the search over the extensions. This is very consistent with the complexity results on default reasoning which indicate two independent sources of complexity. These sources are orthogonal to each other and hence, e.g., the implementation of the classical theorem proving task has little effect on the inherent complexity of the search.

Now we develop methods for implementing upper and lower bounds with the aim of obtaining tight bounds using a low number of theorem prover calls. Consider first the lower bound. For a default theory $(\mathcal{D}, W)$ and a set FS, it can safely contain the logical consequences of W but also those of the ordinary formulae in FS. We denote the set of the ordinary formulae in FS by FS^+. The lower bound can be made even tighter because the closure over the rules in $\mathcal{D}$ enabled by FS, $\mathrm{Mon}(\mathcal{D}, FS) = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D},\ \text{for all } \psi \in \mathrm{just}(\mathcal{D}), \mathrm{not}(\neg\psi) \in FS\}$ can be added without compromising the characterizing property of the lower bound. These considerations lead to quite a tight lower bound:

$$\mathbf{LB}((\mathcal{D}, W), FS) = \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},FS)}(W \cup FS^+) \tag{1}$$

In order to obtain a tight upper bound one should remember that extensions are strongly grounded and everything in the extension is derived from W by classical derivations and the default rules $\mathcal{D}$. Hence, in the upper bound one should include the logical consequences of W but also cover the closure over the applicable rules. This means that one has to include all the possible ways of applying the *potentially applicable rules* in $\mathcal{D}$, i.e., rules that are not disabled by FS

$$\{d \in \mathcal{D} \mid \ \text{for all } \psi \in \mathrm{just}(\mathcal{D}), \neg\psi \notin FS\}.$$

One idea to find an upper approximation of the possible ways of applying the rules is to consider the consequents of the potentially applicable rules. This leads to an upper bound on extensions which is the logical consequences of W and the consequents of the potentially applicable rules. This can be made even tighter by restricting attention to the closure of the monotonic rules given by the potentially applicable ones. We denote this set of monotonic rules by

$$\mathcal{D}^{FS} = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D},\ \text{for all } \psi \in \mathrm{just}(\mathcal{D}), \neg\psi \notin FS\}.$$

This development leads to a rather tight upper bound:

$$\mathbf{UB}((\mathcal{D}, W), FS) = \mathrm{Cn}_{\mathcal{D}^{FS}}(W) \tag{2}$$

The bounds given in equations (1) and (2) are so tight that the soundness and completeness of the decision methods is guaranteed without the post-check.

Notice that membership in the lower and upper bounds given in equations (1) and (2) can be determined by deciding membership in the closure over classical consequences and some monotonic rules. This problem is **co-NP**-complete [Got95] and hence on the same level of complexity as deciding classical consequence.

The decision method **extensions** is essentially a bottom-up procedure for generating extensions but it includes ingredients with which a fair amount of goal-directedness can be built in, i.e., the computation can be made to concentrate on answering a given query. One technique is to design the search heuristics to focus on choices relevant to the query. Another one is to include the query into the constraints FS given as the initial input to the procedure. So when the task is to decide whether φ is a credulous consequence of a default theory, start the method with $FS = \{\varphi\}$ instead of $FS = \emptyset$ and when sceptical consequence needs to be decided, use $FS = \{\text{not}(\varphi)\}$. This has two effects. First, it can prune the search space considerably and, second, all extensions need not be generated fully in order to answer a query and sometimes answers can be given even without generating any of the extensions.

Notice that for general default reasoning it seems unfeasible to develop a fully goal-directed procedure, i.e., a procedure which would examine only those parts of the default theory which are somehow syntactically relevant to the query. This is because extensions are defined with a global condition on the whole theory requiring that each applicable default rule should be applied. Hence, there are theories with no extensions and in the worst case it is necessary to examine every default rule in order to guarantee the existence of an extension. For achieving a goal-directed decision method, one can consider a weaker notion of extensions or syntactically restricted subclasses of default theories such as normal defaults. See, e.g. [Sch95, SB96] for work in this direction.

6 Disjunctive Logic Programs

Disjunctive logic programs with the stable model semantics [Prz91] turn out to be a special case of Reiter's default logic and methods presented in Section 5 can be employed to compute disjunctive stable models. For example, the following translation proposed by Sakama and Inoue [SI93] maps logic programs to default theories such that extensions correspond to stable models. Given a disjunctive program P the corresponding default theory D_P contains the following default rules.

1. For each rule $A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$ in P, the default

$$\frac{: \neg C_1, \ldots, \neg C_n}{B_1 \wedge \cdots \wedge B_m \rightarrow A_1 \vee \ldots \vee A_k}$$

2. For each atom A appearing in P, the default

$$\frac{: \neg A}{\neg A}$$

Then the atomic parts of the extensions of the corresponding default theory provide the stable models of the original program [SI93], i.e.,

(i) each stable model of P is the set of atoms in some extension of D_P and

(ii) the set of atoms in an extension of D_P is a stable model of P.

Hence, any method for computing extensions of default theories can be used for computing disjunctive stable models.

Recently special purpose methods for computing disjunctive stable models have been developed for a knowledge representation system based on disjunctive logic programming, `dlv` system [ELM+97, ELM+98]. The computational kernel of `dlv` is an efficient engine for computing stable models of a disjunctive logic program using algorithms presented in [LRS97] together with various improvements by suitable optimization techniques.

7 Normal Logic Programs

The computational complexity of disjunctive logic programs is still on the same level as that of the original default logic: the problem of deciding whether a disjunctive program has a stable model is $\mathbf{\Sigma_2^P}$-complete and the complexity remains essentially the same even if negative body literals are not allowed [EG95]. However, if disjunctions are not allowed, the complexity drops one level: deciding whether a normal (non-disjunctive) program has a stable model is **NP**-complete [MT91].

For many practical purposes this subclass of default theories is sufficient. Our aim is to employ the techniques devised for default logic and show how they can be further developed to exploit the more limited structure of logic programs. This is the approach used in a system called **smodels** [NS96] which provides an implementation of the stable model semantics in C++. The system is publicly available (see http://www.tcs.hut.fi/Software/smodels) and seems to outperform clearly other current approaches to computing stable models. We discuss first the propositional case, i.e., ground programs. In the end of the section we indicate how non-ground programs can be handled.

Several methods for computing stable models have already been proposed (see, e.g., [BG94] for a survey). An interesting feature of the approach presented in this section is that for ground programs it enables *polynomial space implementation* which is essential when working with larger programs. This is different from some recent advanced approaches [BNNS94, SNV95] which have exponential worst-case space requirements. The SLG system developed by Chen and Warren [CW95] is closest to the method presented in this section and we compare the two approaches in the end of the section.

In order to get the most out of the limited structure of normal logic programs we treat them as a special subclass of default theories where only atomic facts and default rules with atomic prerequisites and conclusions and negative literals as justifications are allowed. This means that a logic program rule of the form

$$A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$$

corresponds to a default rule

$$\frac{B_1 \wedge \cdots \wedge B_m : \neg C_1, \ldots, \neg C_n}{A}$$

Under this mapping, the stable models of a logic program coincide with the extensions of the corresponding default theory [GL90]. This is a simplification of the mapping in the previous section where implications were used. Notice that the difference between logic programs and default rules is that in logic program rules the *not* operator speaks directly about membership in the extension whereas in default rules justifications state a consistency requirement (non-membership of the complement of the justification).

The close correspondence between stable models for normal logic programs and extensions of default theories becomes evident when we rephrase the original definition of stable models [GL88] in terms of the machinery developed for default logic. The original definition uses the notion of a reduct of program P with respect to a model S (a set of atoms) which is the set of definite rules

$$\begin{aligned}\{A \leftarrow B_1, \ldots, B_m \mid\ & A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \in P, \\ & \{C_1, \ldots, C_n\} \cap S = \emptyset\}.\end{aligned}$$

We denote this set by P^S and it can be seen as a set of monotonic inference rules. In this section we are interested in atomic consequences of such rules. We denote by $\mathrm{Cn}_R(\emptyset)$ the atomic formulae derivable from a set of such monotonic rules R. A set of such rules can also be taken as a set of definite clauses and the set of atomic consequences coincides with the unique least model of the clauses. Now a set of atoms S is a stable model of a program P if and only if $S = \mathrm{Cn}_{P^S}(\emptyset)$ [GL88]. Hence, P^S plays the role of $\mathcal{D}_E$ in Definition 2.1 of default extensions.

Our treatment of default logic in Section 5 is based on the notion of full sets (see Definition 4.3). When working with logic programs we use the representation of full sets introduced in Section 5 which employs the *not* operator and membership in extensions as the fundamental concepts. Hence, we can forget the "extra" negation in the justifications and work directly with the atoms appearing in the scope of the *not* operator. For a logic program P, we denote the set of atoms in the the scope of the *not* operator by $\mathrm{NA}(P)$. To illustrate the notation we show how to rewrite the definition of full sets using these concepts directly.

Definition 7.1 (P-full Sets)
For a logic program P, a set $F \subseteq \{not\ A \mid A \in \mathrm{NA}(P)\}$ is P-full if and only if

$$\textit{for all } A \in \mathrm{NA}(P),\ not\ A \in F \ \textit{ if and only if } A \notin \mathrm{Cn}_{\mathrm{Mon}(P,F)}(\emptyset)$$

where

$$\mathrm{Mon}(P,F) = \{A \leftarrow B_1, \ldots, B_m \mid \begin{array}{l} A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n \in P, \\ \{not\ C_1, \ldots, not\ C_n\} \subseteq F\} \end{array}$$

and $\mathrm{Cn}_{\mathrm{Mon}(P,F)}(\emptyset)$ is the deductive closure of the rules $\mathrm{Mon}(P,F)$.

Full sets capture stable models in the following way:

1. for every P-full F, $\mathrm{Cn}_{\mathrm{Mon}(P,F)}(\emptyset)$ is a stable model of P and
2. for every stable model M, $F = \{not\ A \mid A \in \mathrm{NA}(P) - M\}$ is the unique corresponding full set such that $M = \mathrm{Cn}_{\mathrm{Mon}(P,F)}(\emptyset)$.

Example 4 *Consider the program P*

$$\begin{array}{l} p \leftarrow not\ q, r \\ q \leftarrow not\ p, r \\ r \leftarrow not\ q \end{array}$$

Now $\mathrm{NA}(P) = \{q, p\}$ and $F = \{not\ q\}$ is P-full as $\mathrm{Mon}(P,F)$ has two rules $p \leftarrow r$ and $r \leftarrow$ and, hence, $\mathrm{Cn}_{\mathrm{Mon}(P,F)}(\emptyset) = \{p, r\}$. This is also a stable model of P. For instance, $F' = \{not\ p\}$ is not P-full since $\mathrm{Cn}_{\mathrm{Mon}(P,F')}(\emptyset) = \emptyset$ but $not\ q \notin F'$.

We exploit the connection between logic programming and default logic in order to develop an efficient implementation of the stable model semantics using the decision method for default logic presented in Section 5 (Fig. 2). In order to clarify the starting point we repeat the method in Fig. 3 with the modification that the input is a logic program P instead of a default theory and the current search assumptions FS is just a set of literals (formulae of the form A or *not* A where A is atomic). The function **extensions** employs four subroutines. The function **test** is used for testing whether the extension (stable model) found by the procedure fulfills some given property, e.g., contains a given formula. The other three (**expand**, **conflict**, and **choose**) identify the key subtasks of the function **extensions** and we develop optimized implementations of these subroutines by exploiting the special properties of normal logic programs.

function extensions(P, FS, φ)
$FS' :=$ **expand**(P, FS);
if conflict(P, FS') returns true **then** return false
else if FS' covers $\mathrm{NA}(P)$ **then** return **test**(P, FS', φ)
else
  $\chi :=$ **choose**(P, FS');
  if extensions$(P, FS' \cup \{\chi\}, \varphi)$ returns true **then** return true
  else return **extensions**$(P, FS' \cup \{\mathrm{compl}(\chi)\}, \varphi)$
  end if
end if

Figure 3: A decision procedure for the stable model semantics

expand

In order to guarantee the soundness and completeness of the function **extensions**, the function **expand**(P, FS) should return a set of literals FS' which extends FS and with which every stable model of P agreeing with FS agrees. In Section 5 we introduced a technique for developing such functions by employing the notions of a lower and upper bound:

- For the lower bound $\mathbf{LB}(P, FS)$ it holds that every stable model of P agreeing with FS, agrees with $\mathbf{LB}(P, FS)$.
- For the upper bound $\mathbf{UB}(P, FS)$ it holds that for every stable model M of P which agrees with FS, $M \subseteq \mathbf{UB}(P, FS)$.

The implementation of **expand** is then based on the implementations of the bounds in the following way: **expand**(P, FS) returns the smallest set of literals FS' extending FS and closed under the following rules:

- if $\varphi \in \mathbf{LB}(P, FS')$, then include φ in FS' and
- if $A \notin \mathbf{UB}(P, FS')$ for $A \in \mathrm{NA}(P)$, then include *not* A in FS'.

Hence, for implementing **expand**, it remains to devise the implementations of the bounds. In Section 5 we presented a technique for developing tight bounds for default logic. Here we show how even tighter bounds can be devised in the case of logic programs. First we generalize the notion of potentially applicable rules in a program P under a set of assumptions FS discussed in Section 5. We denote this set of rules by P/FS and require that a rule is potentially applicable only when the positive as well as negative body literals are not in conflict with FS.

$$\begin{aligned} P/FS = \{\ & A \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \mid \\ & \{\mathit{not}\ B_1, \ldots, \mathit{not}\ B_m, C_1, \ldots, C_n\} \cap FS = \emptyset\} \end{aligned}$$

Consider the program P

$$\begin{aligned} & p \leftarrow q \\ & q \leftarrow p \\ & p \leftarrow s, \mathit{not}\ r \\ & s \leftarrow \mathit{not}\ t, \mathit{not}\ p \\ & r \leftarrow \mathit{not}\ t, \mathit{not}\ q \\ & t \leftarrow \mathit{not}\ r \end{aligned} \tag{3}$$

For example, the program $P/\{not\ s, t\}$ consists of the first two and the last rule of P.

We define the lower bound $\mathbf{LB}(P, FS)$ as the smallest set FS' including FS and closed under the following rules:

1. If for $A \leftarrow L_1, \ldots, L_n \in P/FS'$, $\{L_1, \ldots, L_n\} \subseteq FS'$, then $A \in FS'$
2. If an atom A is not a head of any rule in P/FS', then $not\ A \in FS'$.
3. If $A \in FS'$ is the head of only one rule $A \leftarrow L_1, \ldots, L_n$ in P/FS', then $\{L_1, \ldots, L_n\} \subseteq FS'$.
4. If $not\ A \in FS'$ and A is the head of a rule $A \leftarrow L_1, \ldots, L_n$ in P/FS' such that $\{L_1, \ldots, L_{i-i}, L_{i+1}, \ldots, L_n\} \subseteq FS'$, then $\mathrm{compl}(L_i) \in FS'$.

For the upper bound $\mathbf{UB}(P, FS)$ we use the least model of the definite program obtained by removing all negative literals from P/FS.

Example 5 *Consider the program P defined in (3) and the execution of* $\mathbf{expand}(P, \{p\})$. *First* $\mathbf{LB}(P, \{p\}) = \{p, q, not\ s, not\ r, t\}$ *where q is obtained by Rule (1), $not\ s$ and $not\ r$ by Rule (2) and finally t by Rule (1). Now* $\mathbf{UB}(P, \{p, q, not\ s, not\ r, t\}) = \{t\}$. *Hence,* **expand** *adds $not\ p, not\ q$ to the assumptions. No new assumptions are derivable and* $\mathbf{expand}(P, \{p\})$ *returns* $\{p, q, not\ s, not\ r, t, not\ p, not\ q\}$.

As another example, consider $\mathbf{expand}(P, \{not\ s\})$. *Now* $\mathbf{LB}(P, \{not\ s\}) = \{not\ s\}$ *but* $\mathbf{UB}(P, \{not\ s\}) = \{s, r, t\}$. *Thus,* **expand** *adds $not\ p, not\ q$ as new assumptions. Then* $\mathbf{LB}(P, \{not\ s, not\ p, not\ q\}) = \{not\ s, not\ p, not\ q, t, not\ r\}$ *where t is obtained by Rule (4) and $not\ r$ by Rule (3). No new assumptions are derivable. Hence,* $\mathbf{expand}(P, \{not\ s\})$ *returns* $\{not\ s, not\ p, not\ q, t, not\ r\}$.

Notice that Rules 1–2 for the lower bound correspond to the Fitting operator Φ_P and Rules 3–4 generalize the backward propagation of assumed truth values presented by [CW95]. When employing the bounds given above the function **expand** computes a partial model which is closely related to the well-founded model of the program: if there are no initial assumptions, then $\mathbf{expand}(P, \emptyset)$ computes the well-founded model of P but when assumptions are present the partial model computed by **expand** provides more information. For instance, as we demonstrated in the previous example for the program P from (3) $\mathbf{expand}(P, \{p\})$ returns $\{p, q, not\ s, not\ r, t, not\ p, not\ q\}$ but the well-founded model of $P \cup \{p\}$ is $\{p, q, not\ s, not\ r, t\}$.

It turns out that the upper and lower bound are implementable in linear time using, e.g., data structures similar to those employed in a linear time algorithm for Horn clause satisfiability problem proposed in [DG84]. This leads to quadratic time worst case performance for **expand**.

conflict

For soundness and completeness it is sufficient that

1. if $\mathbf{conflict}(P, FS)$ returns true, there is no stable model agreeing with FS and
2. if FS covers $\mathrm{NA}(P)$ and for some $not\ \chi \in FS$, $\chi \in \mathrm{Cn}_{\mathrm{Mon}(P,FS)}(\emptyset)$ or for some $\chi \in FS$, $\chi \notin \mathrm{Cn}_{\mathrm{Mon}(P,FS)}(\emptyset)$, then $\mathbf{conflict}(P, FS)$ returns true.

These conditions are easy to fulfill because of the tightness of the bounds. As the conflict test is performed for a set of assumptions computed by **expand** it is sufficient to implement **conflict** as follows:

> $\mathbf{conflict}(P, FS)$ returns true if and only if there is some atom A such that $\{A, not\ A\} \subseteq FS$.

choose

The function **choose** implements the search heuristics and **choose**(P, FS) should return either A or *not* A for some atom $A \in \mathrm{NA}(P)$ which is not covered by FS. The heuristics for choosing the next choice point has a strong influence on the performance of the decision method. For example, the following two approaches could be used.

- Start from the lowest strongly connected component in the dependency graph of the program.

  This kind of a heuristics has been proposed, e.g., in [SNV95] and it is computationally inexpensive: a fixed order for atoms in $\mathrm{NA}(P)$ can be computed once in linear time and this order can be used for selecting the next choice point. The disadvantage of the approach is that is it quite static and insensitive to the current search assumptions and can lead to a large search space.

- Maximum Occurrences in rules of Minimum size (Mom's) heuristics

  This approach is one of the most successful heuristics for the Davis-Putnam type procedure for propositional satisfiability. It can be adjusted to stable model computation and made quite dynamic, e.g., using the following approach [NS96]. Given a set of current assumptions FS, select a negative literal from a body of a potentially applicable rule (i.e. included in P/FS) with the least number of negative literals not in FS. This heuristics is dynamic and needs to be re-evaluated each time the set of the assumptions FS changes. This re-evaluation can be done in linear time. Hence, the heuristics is more expensive than the previous static one but is more sensitive to the current search assumptions.

test

Notice that the use of the **test** function is not necessary when the queries are, e.g., literals. For such queries we can employ a function **test** which always returns true and implement the usual nonmonotonic reasoning tasks in the following way by employing the assumptions FS given as input to the function **extensions**.

- Finding a stable model:

  extensions$(P, \emptyset, \varphi)$ returns true *if and only if* there is an stable model of P.

- Credulous reasoning, i.e., deciding whether there is a stable model agreeing with a set of literals S:

  extensions(P, S, φ) returns true *if and only if* there is a stable model agreeing with S.

- Sceptical reasoning, i.e., deciding whether a literal L is in every stable model of P:

  extensions$(P, \{\mathrm{compl}(L)\}, \varphi)$ returns false *if and only if* L is in every stable model of P.

For more complicated queries and reasoning tasks the **test** function provides the necessary flexibility.

Example 6 *We illustrate the method with two examples. Consider the following program P.*

$$
\begin{array}{lllll l}
p \leftarrow \mathit{not}\ q, \mathit{not}\ u & & 1.\ \mathit{not}\ u & & & 1.\ \mathit{not}\ s \\
q \leftarrow \mathit{not}\ p, \mathit{not}\ u & & \swarrow\searrow & & & 2.\ \mathit{not}\ u \\
r \leftarrow p & 2.\ \mathit{not}\ t & & & 3.\ t & 3.\ t \\
s \leftarrow \mathit{not}\ q & 4.\ s & & & 11.\ \mathit{not}\ s & 4.\ q \\
s \leftarrow \mathit{not}\ r, q & \swarrow\searrow & & & 12.\ q & 5.\ \mathit{not}\ p \\
t \leftarrow \mathit{not}\ s & 5.\ \mathit{not}\ r & 6.\ r & & 13.\ \mathit{not}\ p & 6.\ \mathit{not}\ r \\
s \leftarrow \mathit{not}\ t & 7.\ \mathit{not}\ p & 9.\ p & & 14.\ \mathit{not}\ r & 7.\ s \\
 & 8.\ q & 10.\ \mathit{not}\ q & & 15.\ \mathit{not}\ q & \times \\
 & & & & \times &
\end{array}
$$

First we discuss the problem of determining all stable models of P. For finding one stable model, it is enough to let **test** *return always true and start with an empty set of assumptions FS but the method can also compute all stable models by letting* **test** *return always false and recording the full sets found.*

We illustrate the execution of **extensions**$(P, \emptyset, \varphi)$ *with the tableau like structure on the left-hand side which can be read as follows. Formula 1 is obtained in* **expand** *by Rule 1, then* **choose** *is called and it returns not t. Hence, the tableau splits into two branches corresponding to the two recursive calls of* **extensions**. *Formula 4 is obtained in* **expand** *by Rule 1 etc. Thus, all formulae in the tableau are derived in* **expand** *(using the four rules for the lower bound and the upper bound rule) except those immediately after the splitting of the tableau which are provided by the function* **choose**. *When applying the rules, the current set of assumptions FS is given by the branch starting from the root of the tableau to the current leaf node. For example, consider how the formula 14 derived. The current set of assumptions (up to the formula 13) $FS = \{\mathit{not}\ u, t, \mathit{not}\ s, q, \mathit{not}\ p\}$ implies that the set of potentially application rules P/FS does not contain any rule for r and not r is obtained by Rule 2. A branch ending with $\times$ contains a conflict, i.e., a pair A, not A for some atom A.*

The execution of **extensions** *calls the* **test** *function for the first time after the formula 8 when*

$$\mathrm{NA}(P) = \{p, q, u, r, s, t\}$$

is covered by the current assumptions $FS = \{\mathit{not}\ u, \mathit{not}\ t, s, \mathit{not}\ r, \mathit{not}\ p, q\}$ and no conflicts are detected. Hence, a stable model of P has been found. This can be obtained by computing the deductive closure $\mathrm{Cn}_{\mathrm{Mon}(P,FS)}(\emptyset)$ which in this case happens to coincide with the positive assumptions on the branch $\{s, q\}$.

If we were interested in finding one stable model, it would be enough to let **test** *to return true. However, for finding all stable models of P we let* **test** *return false. Then* **extensions** *backtracks to the previous choice point and builds another branch. This does not contain any conflicts either and provides another stable model $\{s, r, p\}$. The function* **test** *returns false and* **extensions** *backtracks to the first choice point and builds the third branch which ends up with a conflict. No more backtracking is possible and* **extensions** *returns false. So P has two stable models $\{s, q\}$ and $\{s, r, p\}$ corresponding to the completed branches that contain no conflicts.*

As another example, consider the problem of determining whether every stable model of P contains the atom s. Hence, we start with the initial assumptions $FS = \{\mathit{not}\ s\}$ and let **test** *to return always true. The tableau on the right hand side illustrates the execution of* **extensions**$(P, \{\mathit{not}\ s\}, \varphi)$. *A branch is built but not completed because a conflict is detected. Hence,* **extensions**$(P, \{\mathit{not}\ s\}, \varphi)$ *returns false implying that s is in every stable model of P. This example shows that it is not necessary to build any stable models in order to answer this kind of a sceptical query.*

The approach presented here for computing stable models is quite similar to that used in the SLG system developed by Chen and Warren [CW95]. The SLG system performs goal-directed query-evaluation and the well-founded semantics is employed for computing the residual program for a given query using SLG-resolution. The query can then be evaluated with respect to the well-founded semantics against the residual program. The SLG system evaluates the query with respect to the stable model semantics by computing the stable models of the ground residual program with an *assume-and-reduced* algorithm which assumes truth values of negative body literals and propagates these in order to prune the search space. However, the propagation is weaker than that in the function **expand**: assumed positive truth values are not propagated, backward propagation seems to be less dynamic and the upper bound is not employed. Hence, the assume-and-reduce algorithm prunes the search space less efficiently. This difference can also be observed experimentally when comparing the two approaches using larger ground programs [NS96].

Notice that the SLG system can handle the stable model semantics also for non-ground programs provided that the resulting residual program is finite. This seems to be a promising approach for non-ground programs: given a query, use an efficient implementation of SLG-resolution like XSB [SSW96] and compute the residual program and then use another method, like the one presented in this section, for evaluating the query with respect to stable models on the basis of the residual program. However, the use of the residual program can lead to unsound results: the stable models of the residual program are not necessarily stable models of the original program [CW95]. For function free programs which are range restricted, soundness can be guaranteed by employing a bottom-up method for computing a grounded version of a non-ground program such that the stable models are preserved.This approach has been employed, e.g., in the **smodels** system [NS96].

Acknowledgements

Parts of the material in this article have been developed together with Jürgen Dix and Ulrich Furbach and the author would like to thank them for interesting discussions, comments and feedback.

References

[ABW88] K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Los Altos, 1988.

[BDG88] J.L. Balcázar, I. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, Berlin, 1988.

[BED91] 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.

[BED92] R. Ben-Eliyahu and R. Dechter. Propositional semantics for default logic. In *Working Notes of the 4th International Workshop on on Nonmonotonic Reasoning*, pages 13–27, Plymouth, Vermont, USA, May 1992.

[BG94] Chitta Baral and Michael Gelfond. Logic Programming and Knowlege Representation. *Journal of Logic Programming*, 19-20:73–148, 1994.

[BH95] F. Baader and B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning*, 14:149–180, 1995.

[BNNS94] C. Bell, A. Nerode, R.T. Ng, and V.S. Subrahmanian. Mixed integer programming methods for computing nonmonotonic deductive databases. *Journal of the Association for Computing Machinery*, 41(6):1178–1215, November 1994.

[BO97] P.A. Bonatti and N. Olivetti. A sequent calculus for skeptical default logic. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 107–121, Pont-à-Mousson, France, May 1997. Springer-Verlag. Lecture Notes in Artificial Intelligence 1227.

[Bon96] P.A. Bonatti. Sequent calculi for default and autoepistemic logics. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, pages 127–142, Terrasini, Italy, May 1996. Springer-Verlag. Lecture Notes in Artificial Intelligence 1071.

[CDSS97] Marco Cadoli, Francesco M. Donini, Marco Schaerf, and Riccardo Silvestri. On compact representations of propositional circumscription. *Theoretical Computer Science*, 182:183–202, 1997. (Extended abstract appeared in: On Compact Representations of Propositional Circumscription. *STACS '95*, pages 205–216, 1995.).

[CH85] A. Chandra and D. Harel. Horn clause queries and generalizations. *Journal of Logic Programming*, 2:1–15, 1985.

[CW95] Weidong Chen and David S. Warren. Computing of Stable Models and its Integration with Logical Query Processing. *IEEE Transactions on Knowledge and Data Engineering*, 17:279–300, 1995.

[DG84] 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.

[EG95] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[ELM+97] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A Deductive System for Nonmonotonic Reasoning. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97)*, number 1265 in Lecture Notes in AI (LNAI), Berlin, 1997. Springer.

[ELM+98] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 406–417, 1998.

[Fit90] M. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer-Verlag, New York, 1990.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman and Company, San Francisco, 1979.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic Program with Classical Negation. In David H.D. Warren and Peter Szeredi, editors, *Proceedings of the 7th Int. Conf. on Logic Programming*, pages 579–597. MIT, June 1990.

[Got92] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, 1992.

[Got95] G. Gottlob. The complexity of default reasoning under the stationary fixed point semantics. *Information and Computation*, 121:81–92, 1995.

[GPSK95] Goran Gogic, Christos Papadimitriou, Bart Selman, and Henry Kautz. The Comparative Linguistics of Knowledge Representation. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 862–869, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.

[Jan98] T. Janhunen. On the intertranslatability of autoepistemic, default and priority logics, and parallel circumscription. In J. Dix, L. Farias del Cerro, and U. Furbach, editors, *Proceedings of the 6th European Workshop on Logics in Artificial Intelligence*, pages 216–232, Dagstuhl, Germany, October 1998. Springer-Verlag. LNAI 1489.

[JK90] 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. MIT Press.

[Kam95] M. Kaminski. A Comparative Study of Open Default Theories. *Artificial Intelligence*, 77:285–319, 1995.

[KS91] H. Kautz and B. Selman. Hard problems for simple default logics. *Artificial Intelligence*, 49:243–279, 1991.

[Lif85] Vladimir Lifschitz. Computing Circumscription. In *Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, California*, pages 121–127, 1985.

[LRS97] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135(2):69–112, June 1997.

[McC80] John McCarthy. Circumscription: A Form of Nonmonotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[MD80] D. McDermott and J. Doyle. Non-monotonic logic I. *Artificial Intelligence*, 13:41–72, 1980.

[Moo85] R.C. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25:75–94, 1985.

[MST93] W. Marek, G.F. Schwarz, and M. Truszczyński. Modal nonmonotonic logics: Ranges, characterization, computation. *Journal of the Association for Computing Machinery*, 40(4):963–990, 1993.

[MT91] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the Association for Computing Machinery*, 38:588–619, 1991.

[MT93] Wiktor Marek and Mirek Truszczyński. *Nonmonotonic Logics; Context-Dependent Reasoning*. Springer, Berlin, 1st edition, 1993.

[Nie95] Ilkka Niemelä. Towards efficient default reasoning. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 312–318, Montreal, Canada, August 1995. Morgan Kaufmann Publishers.

[NR94] Ilkka Niemelä and Jussi Rintanen. On the impact of stratification on the complexity of nonmonotonic reasoning. *Journal of Applied Non-Classical Logics*, 4(2):141–179, 1994.

[NS96] Ilkka Niemelä and Patrik Simons. Efficient Implementation of the Well-founded and Stable Model Semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, September 1996. The MIT Press.

[NS97] Ilkka Niemelä and Patrik Simons. Smodels – an implementation of the stable model and well-founded semantics for normal logic programs. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 420–429, Dagstuhl, Germany, July 1997. Springer-Verlag.

[PP94] H. Przymusinska and T. Przymusinski. Stationary default extensions. *Fund. Informat.*, 21 (1,2):67–87, 1994.

[Prz91] T.C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing*, 9:401–424, 1991.

[Rei80] Raymond Reiter. A Logic for Default-Reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[RS94] V. Risch and C. Schwind. Tableau-based characterization and theorem proving for default logic. *Journal of Automated Reasoning*, 13:223–242, 1994.

[SSW96] K. Sagonas, T. Swift, and D.S. Warren. An abstract machine for computing the well-founded semantics. In M. Maher, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 274–288, Bonn, Germany, September 1996. The MIT Press.

[SB96] T. Schaub and S. Brüning. Prolog technology for default reasoning. In *Proceedings of the European Conference on Artificial Intelligence*, pages 105–109, Budapest, Hungary, August 1996. John Wiley.

[Sch95] T. Schaub. A new methodology for query-answering in default logics via structure-oriented theorem proving. *Journal of Automated Reasoning*, 15(1):95–165, 1995.

[SI93] C. Sakama and K. Inoue. Relating disjunctive logic programs to default theories. In *Proceedings of the 2nd International Conference on Logic Programming and Non-Monotonic Reasoning*, pages 266–282. MIT Press, 1993.

[SNV95] V.S. Subrahmanian, Nau, and Vago. WFS + Branch and Bound = Stable Models. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):362–377, 1995.

[Sti90] J. Stillman. It's not my default: the complexity of membership problems in restricted propositional default logics. In *AAAI-90 — Proceedings of the 9^{th} AAAI National Conference on Artificial Intelligence*, pages 571–578, Boston, Massachusetts, USA, July 1990. The MIT Press.

[Sti92] J. Stillman. The complexity of propositional default logics. In *AAAI-92 — Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 794–800, San Jose, California, USA, July 1992. The MIT Press.

[VG88] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann Publishers, Los Altos, 1988.

[VGRS91] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the Association for Computing Machinery*, 38(3):620–650, July 1991.