CHAPTER 19

Nonmonotonic Reasoning: Towards Efficient Calculi and Implementations

Jürgen Dix

Ulrich Furbach

Ilkka Niemelä

SECOND READERS: Gerd Brewka and Mirek Truszczyński.

Contents

HANDBOOK OF AUTOMATED REASONING
Edited by Alan Robinson and Andrei Voronkov

Preface

Research in nonmonotonic reasoning has begun at the end of the seventies. One of the major motivations came from reasoning about actions and events. John McCarthy and Patrick Hayes had proposed their situation calculus as a means of representing changing environments in logic. The basic idea is to use an extra situation argument for each fact which describes the situation in which the fact holds. Situations, basically, are the results of performing sequences of actions. It soon turned out that the problem was not so much to represent what changes but *to represent what does not change* when an event occurs. This is the so-called *frame problem*. The idea was to handle the frame problem by using a default rule of the form

> *If a property P holds in situation S then P typically also holds in the situation obtained by performing action A in S.*

Given such a rule it is only necessary to explicitly describe the changes induced by a particular action. All non-changes, for instance that the *real colour* of the kitchen wall does not change when the light is turned on, are handled implicitly. Although it turned out that a straightforward formulation of this rule in some of the most popular nonmonotonic formalisms may lead to unintended results, the frame problem was certainly the challenge motivating many people to join the field.

In the meantime a large number of different nonmonotonic logics have been proposed and we have many results that clarify the close relationship as well as the different aspects of such logics. There is quite an agreement in the field that now, after almost twenty years, the conceptual phase has ended and there is enough knowledge available to start implementing nonmonotonic reasoning systems and applying them to non-trivial problems.

Overall Organization

In this chapter we do not want to give a detailed overview of the various formalizations of nonmonotonic reasoning that have evolved (those can be found in [Brewka, Dix and Konolige 1997, Brewka 1996, Marek and Truszczyński 1993]), but we want to give an overview of the main computational techniques and methods leading to implementations of nonmonotonic reasoning. However, to make this chapter self-contained, we give compact overviews (parts of which are taken from [Brewka et al. 1997, Brewka and Dix 2001]) of the main nonmonotonic logics (in Section 1) and semantics of logic programs (in Section 4).

It turns out that logic programming techniques as well as methods that originated in automated reasoning, like *resolution* or *tableau calculi*, can be successfully applied. We can compare this with the programming language PROLOG, that proved to be successful because it is based on a subset of first-order predicate logic, so called *Horn-clauses*, or *definite logic programs*. Although already definite programs give rise to nonmonotonic semantics, full nonmonotonicity comes in when we extend definite programs by *negation* and/or *disjunction*. Consequently, we will distinguish in this chapter between general nonmonotonic logics and semantics of logic programs with negation and disjunction. The latter class is syntactically more restricted (no negation in the head) but has also features which makes it incomparable to general theories.

The chapter is organized as follows. In Section 1 we first introduce the main nonmonotonic logics: *Default Logic*, *Circumscription* and *Autoepistemic Logic*. They form the basis of most nonmonotonic reasoning systems that are in use today. We also consider the abstract approach of Kraus, Lehmann and Magidor to associate with any reasoning system an *abstract consequence relation*. Abstract properties of these relations often help to better understand the overall behaviour of the reasoning system and irregularizes in the induced relations give sometimes rise to define new systems, like the *rational closure* $\mathcal{K}^{\mathrm{rat}}$ of Lehmann and Magidor (which was motivated by shortcomings the *cumulative* and *preferential* system.

In Section 2 we investigate universal methods for computing in general nonmonotonic logics. We do this with a special eye on the underlying complexity and show how this lead to automated theorem proving in such logics.

Finding efficient computation mechanisms for the logics introduced in the former section is the aim of Section 3. There we consider techniques that originated from automated reasoning in first-order predicate calculus. We depict how these techniques can be applied for disjunctive logic programming with programs with variables but only limited use of negation. In particular, we handle GCWA as a basis for nonmonotonic negation therein.

Section 4 gives a declarative overview on nonmonotonicity in logic programming. We introduce (nonmonotonic) semantics of logic programs with negation and disjunction, notably the well-founded and the stable semantics and their extensions to programs containing disjunction— they constitute the most important semantics and are in close relation to the logics introduced in Section 1.

While in Section 3 we considered techniques that can be successfully applied for programs with variables and only limited use of negation, we treat in Section 5 propositional programs with full negation and disjunction. In particular, we provide implementations of D-WFS and D-STABLE in polynomial space.

We end with Section 6, where we treat the problem of finding good benchmarks to test and compare nonmonotonic systems against.

1. General Nonmonotonic Logics

Nonmonotonic reasoning as a form of reasoning contrasts with standard deductive reasoning in the following way. Suppose we know that all men are mortal and that Socrates is a man. Then it follows, by simple syllogistic reasoning, that Socrates is mortal. Further, nothing we can add to our knowledge will change that conclusion, given that we still hold the premises. Deductive reasoning within a theory is “local” in the sense that, having derived a conclusion from premises, we need not worry about any other sentences in the theory. To use the technical term, deductive reasoning is *monotonic*:

$$\text{If } S \vdash \varphi \text{ then } S \cup \{\psi\} \vdash \varphi\,, \tag{1.1}$$

where $S \vdash \varphi$ means that proposition φ follows from the propositions S.

Obviously, nonmonotonic reasoning does not share this property. Why would we ever want a logic that is nonmonotonic? Historically, the need for nonmonotonic reasoning was identified in the course of trying to solve knowledge representation problems in several application areas of Artificial Intelligence (AI). For more detailed discussions on the need of nonmonotonicity we refer to the literature ([Brewka et al. 1997, Brewka 1996, Marek and Truszczyński 1993]). Here it suffices as a first example to consider the *Closed World-Assumption* (CWA). It intuitively means that any information not mentioned in a database is taken to be false. More precisely, if a positive instance of a predicate is not contained in the database, its negation is assumed to hold. For example, if an airline flight database does not contain any information about a flight from Chicago to San Francisco on May 22, 1998, it is assumed that such a flight does not exist.

For a database that is a simple set of facts, the closed-world assumption can be implemented by a search of the database. If the database includes a deductive component, then the CWA can be phrased in terms of the models of the database. Generally the theories considered for the CWA are restricted to finite languages with only predicates and constant symbols. [Reiter 1978] defines the CWA as:

1.1. Definition *(CWA)*.
The closed world assumption of a database T is the theory

$$\mathrm{CWA}(T) = \mathrm{Cn}(T \cup \{\neg p : \ T \not\vdash p\})\,,$$

where p is a ground predicate instance.

That is, if a ground atom cannot be inferred from the database, its negation is added to the closure. The CWA is a strong nonmonotonic closure operation since any ground predicate not deduced from the database has its negation added to the closure. A problem with strong closures is that they risk being inconsistent for perfectly reasonable databases. For example, consider a database containing just the disjunctive statement $p \vee q$. Since neither p nor q is deduced, both $\neg p$ and $\neg q$ are in the closure, which is then inconsistent with the original database.

The CWA served us as one particular simple example of a nonmonotonic logic. There exist other systems and one usually can distinguish between four major types of such logics:

Section 1.2: Logics using nonstandard inference rules with an additional consistency check to represent default rules. Reiter's *Default Logic* and its variants are of this type.

Section 1.3: *Circumscription* and its variants. These approaches are based on a preference relation on models. A formula is a consequence *if and only if* it is true in all most preferred models of the premises. Syntactically, a second order formula is used to eliminate all non-preferred models.

Section 1.4: Nonmonotonic modal logics using a modal operator to represent consistency or (dis-) belief. These logics are nonmonotonic since conclusions may depend on disbelief. The most prominent example is Moore's autoepistemic logic.

Section 1.5: Conditional approaches which use a non truth-functional connective $\mid\!\sim$ to represent defaults. A particularly interesting way of using such conditionals was proposed by Kraus, Lehmann and Magidor. They consider φ as a default consequence of ψ *if and only if* the conditional $\psi\mid\!\sim\varphi$ is in the closure of a given conditional knowledge base under a collection of rules. Each of the rules directly corresponds to a desirable property of a nonmonotonic inference relation.

The various logics are intended to handle different intuitions about nonmonotonic reasoning in a most general way. On the other hand, the generality leads to problems, at least from the point of view of implementations and applications. In the first-order case the approaches are not even semi-decidable since an implicit consistency check is needed. In the propositional case we still have tremendous complexity problems. For instance, the complexity of determining whether a formula is contained in all extensions of a propositional default theory is on the second level of the polynomial hierarchy. As mentioned earlier we believe that logic programming techniques can help to overcome these difficulties (see Sections 4, 3).

Originally, nonmonotonic reasoning was intended to provide us with a *fast* but *unsound* approximation of classical reasoning in the presence of incomplete knowledge. Therefore one might ask whether the higher complexity of NMR-formalisms (compared to classical logic) is not a serious drawback of this aim?

The answer is that NMR-systems allow us to formulate a problem in a very *compact* way as a theory T. It turns out that there are theories T, where any equivalent formulation in classical logic (if possible at all) as a theory T' is much larger: the size of T' is exponential in the size of T! We refer to [Gogic, Papadimitriou, Selman and Kautz 1995] and [Cadoli, Donini and Schaerf 1996, Cadoli, Donini, Schaerf and Silvestri 1997, Cadoli, Donini, Liberatore and Schaerf 1995] where such problems are investigated.

1.1. Herbrand vs. General Models

The alert reader may have taken a second glance at the "ground term" phrase in the definition of CWA. In first-order logic, ground terms that are syntactically different can refer to the same individual, e.g., $a = b$ means that the individual referred to by a is the same as the one referred to by b. In deductive databases, one generally assumes that different terms refer to different individuals: the names are unique. For most database applications, this makes sense: *United* is a different airline than *Delta*, for example. The standard name schema is the **U**nique **N**ames **A**ssumption:

$$\text{UNA}: \quad \{t_i \neq t_j \mid t_i \text{ and } t_j \text{ are different ground terms}\} \tag{1.2}$$

A second assumption is that the number of individuals is finite and that every individual has a name in the underlying language. Again, in the context of databases, this is justified because there exist only a finite number of airlines, cities, etc..

The assumption of a finite domain is formalized by assuming a finite set $\{t_i | 1 \leq i \leq n\}$ of ground terms and formulating the **D**omain **C**losure **A**ssumption:

$$\text{DCA}: \quad \forall x\,(t_1 \doteq x \vee t_2 \doteq x \vee t_3 \doteq x \vee \cdots \vee t_n \doteq x) \tag{1.3}$$

The schema UNA is a consequence of the CWA, and the schema DCA is consistent with it. We will consider the importance of these schemas when comparing the CWA to circumscription.

When function-symbols are available, then restricting to finite domains would be too restrictive in general. Therefore one often restricts the set of all models to *Herbrand* models. Herbrand models have as the underlying domain exactly the term-algebra generated by the underlying language. The domain closure assumption in this setting means that we restrict to those models $\mathcal{I}$ where for any element a in $\mathcal{I}$ there is a $\mathcal{L}$-term t that represents this element: $a = t^{\mathcal{I}}$.

The difference between Herbrand models and models satisfying UNA and DCA is that the interpretation of terms is uniquely determined in Herbrand models. It is required that a term "$f(t_1, \ldots, f_n)$" is interpreted in a Herbrand model $\mathcal{I}$ as "$f(t_1^{\mathcal{I}}, \ldots, t_n^{\mathcal{I}})$".

1.2. DEFINITION *(Herbrand Models).*
Given a theory T in a language $\mathcal{L}$, we define

Herbrand Universe U_T: it is the algebra consisting of all terms that can be built using the constants and function symbols from $\mathcal{L}$, and

Herbrand Base B_T: it is the set of all atomic formulae of the form $r(t_1, \ldots, t_n)$, where r is a relation symbol of $\mathcal{L}$ and $t_1, \ldots, t_n$ are ground terms from U_T.

It is well-known that restricting to Herbrand models (as opposed to truth in all models) is a much more complex task—namely Π^1_1-complete (while Π^0_1-complete, i.e. recursively enumerable, for all models). See also Section 2, Subsection 2.1.

1.2. Default Logic (DL)

Reiter's default logic [Reiter 1980] is one of the most prominent nonmonotonic logics. Default logic assumes knowledge to be represented in terms of a default theory. A default theory is a pair $(\mathcal{D}, W)$. W is a set of first-order formulas representing the facts which are known to be true with certainty. $\mathcal{D}$ is a set of defaults of the form

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

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

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

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]

Default theories induce so-called *extensions* which represent acceptable belief sets a reasoner may adopt based on the available information. A formula φ is called a *sceptical* consequence of $(\mathcal{D}, W)$ *if and only if* φ is contained in all extensions of $(\mathcal{D}, W)$. φ is called 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 which is slightly different from (but equivalent to) Reiter's original definition. We have found that this definition is somewhat easier to digest. The original definition will be presented later.

Intuitively, E is an extension of $(\mathcal{D}, W)$ *if and only if* E is a deductively closed (in the sense of classical logic) superset of W 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:

1.3. DEFINITION *(Default Proof, $Cn_{\mathrm{Mon}(\mathcal{D})}(W)$).*
Let $(\mathcal{D}, W)$ be a default theory, S a set of formulas, and φ a formula. A $(\mathcal{D}, W)$-default proof for φ is a finite sequence $\Delta = (d_1, \ldots, d_n)$ of defaults in $\mathcal{D}$ such that:

1. $W \cup \{\mathrm{cons}(d_1), \ldots, \mathrm{cons}(d_{i-1})\} \vdash \mathrm{pre}(d_i)$, for $i \in \{1, \ldots, n\}$,
2. $W \cup \{\mathrm{cons}(d_1), \ldots, \mathrm{cons}(d_n)\} \vdash \varphi$.

Δ is valid in S *if and only if* S does not contain the negation of a justification of a default in Δ.

A slightly different, but completely equivalent formulation is to associate to any set of defaults $\mathcal{D}$ a set $\mathrm{Mon}(\mathcal{D})$ of associated *monotonic* inference rules:

$$\mathrm{Mon}(\mathcal{D}) := \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} : d \in \mathcal{D}\}.$$

Let us denote by $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D})}(W)$ the closure of W with respect to classical derivability and the rules $\mathrm{Mon}(\mathcal{D})$. Then formulae having a $(\mathcal{D}, W)$-default proof are exactly those contained in $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D})}(W)$. Valid proofs are those that also respect the justifications of the original defaults.

As usual $\vdash$ denotes classical provability. We now can state the definition of extensions formally:

1.4. DEFINITION *(Extension 1).*
Let $(\mathcal{D}, W)$ be a default theory. E is an extension of $(\mathcal{D}, W)$ *if and only if* E is a deductively closed superset of W satisfying the conditions

[1]Reiter treats open defaults somewhat differently and uses a more complicated method to define extensions for them. We refer to [Kaminski 1995] for a detailed investigation.

1. if $\varphi{:}\psi_1, \ldots, \psi_n/\gamma \in \mathcal{D}$, $\varphi \in E$ and for all i $(1 \leq i \leq n)$ $\neg\psi_i \notin E$, then $\gamma \in E$, and
2. $\varphi \in E$ implies there is a $(\mathcal{D}, W)$-default proof for φ valid in E. This condition is also called *groundedness.*

Reiter's equivalent original definition is more compact. It defines extensions as fixed points of a certain operator.

1.5. DEFINITION *(Extension 2).*
Let $(\mathcal{D}, W)$ be a default theory, S a set of formulas. 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. This version is often used in proofs about default logic and makes the way in which formulas have to be grounded in the premises more explicit. Let E be a set of formulas and define, for a given default theory $(\mathcal{D}, W)$, a sequence of sets of formulas as follows:

$E_0 = W$, 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 an arbitrary number of extensions (including zero). Extensions are always consistent if W is and if there are no degenerate defaults without consistency conditions. If W is inconsistent then the single extension of $(\mathcal{D}, W)$ is the set of all formulas. Extensions are maximal in the following sense: if E is an extension then there is no extension E' such that $E' \subset E$.

Let us consider a few simple examples from the birds domain. Let

$\mathcal{D}_1 = \{bird(x){:}fly(x)/fly(x)\}$

$W_1 = \{bird(tw)\}$.

Since extensions are always deductively closed sets "generated" by W and some of the available default consequents, there are only two candidates to be tested here, namely $S_1 = \mathrm{Cn}(W_1)$ and $S_2 = \mathrm{Cn}(W_1 \cup \{fly(tw)\})$.[2] It is easy to see that $\Gamma(S_1)$ must contain $fly(tw)$. Since S_1 does not contain this formula, it is not a fixed point of Γ. S_2, on the other hand, is a fixed point, as can easily be verified.

Now let

$\mathcal{D}_2 = \mathcal{D}_1$

$W_2 = \{bird(tw), peng(tw), \forall x(peng(x) \leftarrow \neg fly(x))\}$.

[2] Other instances of the default rule need not be considered, since there is no way of deriving $bird(t)$ for any ground term t different from tw.

$\mathcal{D}$	W	Fixed Point(s)
$\frac{bird(x):fly(x)}{fly(x)}$	$bird(tw)$	$\mathrm{Cn}(W \cup \{fly(tw)\})$
$\frac{bird(x):fly(x)}{fly(x)}$	$bird(tw)$ $peng(tw)$ $\forall x(peng(x) \to \neg fly(x)$	$\mathrm{Cn}(W)$
$\frac{bird(x):fly(x)}{fly(x)}$ $\frac{peng(x):\neg fly(x)}{\neg fly(x)}$	$bird(tw)$ $peng(tw)$	$\mathrm{Cn}(W \cup \{fly(tw)\})$ $\mathrm{Cn}(W \cup \{\neg fly(tw)\})$
$\frac{bird(x):fly(x)\wedge\neg peng(x)}{fly(x)}$ $\frac{peng(x):\neg fly(x)}{\neg fly(x)}$	$bird(tw)$ $peng(tw)$	$\mathrm{Cn}(W \cup \{\neg fly(tw)\})$

Table 1: Some Default Theories and Their Extensions

Again there are two candidates, namely $S_1 = \mathrm{Cn}(W_2)$, $S_2 = \mathrm{Cn}(W_2 \cup \{fly(tw)\})$. This time, S_2, which is inconsistent, is not a fixed point of Γ since $\Gamma(S_2) = \mathrm{Cn}(W_2)$. The only fixed point is S_1.

Consider what happens if the universally quantified formula from W_2 is replaced by a corresponding default rule:

$\mathcal{D}_3 = \{bird(x){:}fly(x)/fly(x), peng(x){:}\neg fly(x)/\neg fly(x)\}$
$W_3 = \{bird(tw), peng(tw)\}$.

If instantiations of both defaults (with $x = tw$) are applied, an inconsistency is generated. $\mathrm{Cn}(W_3 \cup \{fly(tw), \neg fly(tw)\})$ cannot be an extension, since $\Gamma(S) = \mathrm{Cn}(W)$ whenever S is inconsistent. If a candidate contains neither $fly(tw)$ nor $\neg fly(tw)$, the operator Γ will produce the set of all formulas, and, again, the candidate is not a fixed point. However, if we apply one of the default instances, we obtain an extension. The two extensions are $\mathrm{Cn}(W_3 \cup \{fly(tw)\})$ and $\mathrm{Cn}(W_3 \cup \{\neg fly(tw)\})$.

Table 1 presents the simple default theories we just discussed together with their extensions.

1.3. Circumscription (CIRC)

Circumscription is a method of defining the closure of a theory by restricting its models to those that have minimal extensions of some of the predicates and functions. Since its first formulation by [McCarthy 1980], it has taken on several different forms, including domain circumscription [McCarthy 1979] (minimizing the elements in the universe of models) and the most popular and useful version, parallel predicate circumscription described in the papers [McCarthy 1980, McCarthy 1986, Lifschitz 1985]. We present this system and some of the major variants, along with a description of *abnormality theories*, a method for default reasoning using circumscription.

Although circumscription was originally presented as a schema for adding more

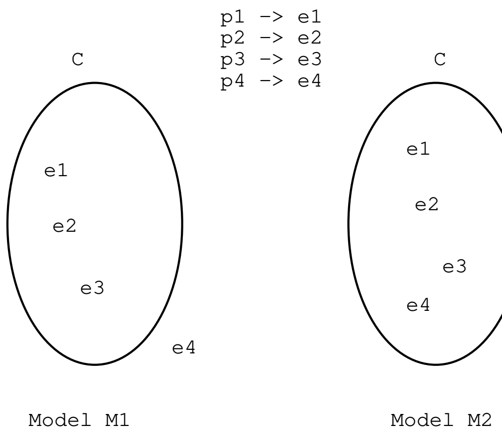


Figure 1: Two Models of a Theory with the Same Valuation.

formulas to a theory (just as Clark's completion does), here we describe it in terms of restricting the models of the theory. This view leads to the generalization of circumscription by model preference theories and is more useful analytically in relating circumscription to other nonmonotonic formalisms. More detailed references to circumscription can be found in Lifschitz's excellent survey article [Lifschitz 1994].

Parallel Predicate Circumscription

Choose a language $\mathcal{L}$ and let $\underline{p}$ be the set of predicate symbols that we are interested in minimizing and $\underline{z}$ another set of predicate symbols the interpretation of which we allow to vary across compared models. For example, if we wish to minimize the number of cannibals, we would let $\underline{p} = \{cannibal\}$ and $\underline{z}$ be all other predicate symbols (the importance of $\underline{z}$ will be indicated later). Suppose T is a theory containing the statements $cannibal(p_1)$, $cannibal(p_2)$, and $cannibal(p_3)$, but no other assertions using cannibal. Then every model of T will have at least the individuals referred to by p_1, p_2, and p_3 with property cannibal. Now consider two models with the same valuation function from terms to individuals, as in Figure 1. In model M_1, the extension of the predicate $cannibal()$ includes just the three individuals e_1, e_2, and e_3. In model M_2 there is a fourth individual, e_4, who is a cannibal. Circumscription would prefer M_1 to M_2, since the extension of $cannibal()$ in M_1 is a proper subset of its extension in M_2. Under appropriate assumptions (that these terms refer to different individuals), circumscription would yield the result $\neg cannibal(p_4)$, which is not present in the original theory. See Definition 2.10 where also *fixed* predicates are allowed.

Let $\psi(\underline{p}, \underline{z})$ be a first-order sentence containing the symbols $\underline{p}$ and $\underline{z}$. Circumscription prefers models of $\psi(\underline{p}, \underline{z})$ that are minimal in the predicates $\underline{p}$, assuming that these models have the same interpretation for all symbols not in $\underline{p}$ or $\underline{z}$: such predicates are called *fixed symbols*.

To state this more formally, let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two models of $\psi(\underline{p},\underline{z})$. $|\mathcal{M}|$ is the universe of model $\mathcal{M}$, and $\mathcal{M}[\![r]\!]$ is the interpretation of the predicate r in $\mathcal{M}$.

1.6. DEFINITION *(Minimal Models, Entailment $\models_{min}$).*

$$\mathcal{M}_1 \leq^{\underline{p};\underline{z}} \mathcal{M}_2 \quad \Leftrightarrow \quad \begin{cases} 1. & |\mathcal{M}_1| = |\mathcal{M}_2|^3. \\ 2. & \mathcal{M}_1[\![r]\!] = \mathcal{M}_2[\![r]\!] \text{ for all } r \text{ not in } \underline{p},\underline{z}. \\ 3. & \mathcal{M}_1[\![p_i]\!] \subseteq \mathcal{M}_2[\![p_i]\!] \text{ for all } p_i \in \underline{p}. \end{cases}$$

In many applications there are no fixed predicates and we are interested in minimizing all predicates while varying all predicates. In such applications we have $\underline{p} = \underline{z}$ and this is the set of all predicates. We write

$$T \models_{min} \phi$$

to denote that all minimal models of the set of formulae T also satisfy the sentence ϕ.

$\leq^{\underline{p};\underline{z}}$ is a preorder relation (reflexive and transitive) on models, but not necessarily a partial order since it is not antireflexive. We define the strict order $\mathcal{M}_1 <^{(\underline{p};\underline{z})} \mathcal{M}_2$ as $\mathcal{M}_1 \leq^{\underline{p};\underline{z}} \mathcal{M}_2$ and not $\mathcal{M}_2 \leq^{\underline{p};\underline{z}} \mathcal{M}_1$. The preferred models of $\psi(\underline{p},\underline{z})$ are those that are minimal according to the strict ordering.

Second-order form

This form of circumscription is called *parallel predicate circumscription* because a set of predicates is minimized in parallel. It can be concisely written as a second-order formula, once we make the following definitions. Let u, v be predicates of the same arity.

$$\begin{array}{lll} u \leq v & \text{stands for} & \forall x\,(u(x) \rightarrow v(x)) \\ u < v & \text{stands for} & u \leq v \wedge v \not\leq u \\ u = v & \text{stands for} & u \leq v \wedge v \leq u \end{array}$$

These sentences express facts about the extensions of u and v: $u \leq v$ means that u's extension is a subset of v's, $u < v$ that it is a proper subset, and $u = v$ that they are coextensive. If u and v are tuples of predicates, then these formulas stand for the obvious conjunction of the operator applied to each pair of predicates. Now we define:

1.7. DEFINITION *(Parallel Predicate Circumscription).*

The parallel predicate circumscription of a theory $\psi(\underline{p},\underline{z})$ is defined as follows

$$\mathrm{Circum}(\psi(\underline{p},\underline{z});\underline{p};\underline{z}) = \psi(\underline{p},\underline{z}) \ \wedge\ \neg\exists \underline{p'}\underline{z'}\,(\,\psi(\underline{p'},\underline{z'})\ \wedge\ \underline{p'} < \underline{p}\,)\ .$$

[3] We also assume that all *terms* get the same interpretation in both models.

[Lifschitz 1985] shows that this definition is equivalent to the minimal model semantics, that is, $\mathcal{M}$ is a model of Circum$(\psi(\underline{p}, \underline{z}); \underline{p}; \underline{z})$ *if and only if* it is a $\leq^{(\underline{p};\underline{z})}$-minimal model of ψ.

Note that circumscription allows comparison of models only if they have the same interpretations of terms and fixed predicates. In general, allowing more comparisons strengthens the results of circumscription since fewer minimal models are present. It is also possible to allow varying function symbols as well as predicates; this has important effects on equality results (see [Brewka et al. 1997]).

Abnormality Theories

[McCarthy 1986] originated the concept of an abnormality theory as a way to represent default reasoning in circumscription. Defaults are represented with an introduced abnormality predicate: for example, to say that normally birds fly, we would use:

$$\forall x\, (bird(x) \wedge \neg ab_1(x) \;\rightarrow\; fly(x)\,). \tag{1.4}$$

Here the meaning of $ab_1(x)$ is something like "x is abnormal with respect to flying birds." Note that there can be many different kinds of abnormality, and they are indexed according to kind. The circumscription would then minimize abnormalities, allowing relevant predicates to vary (e.g., fly). Let ψ stand for the conjunction of (1.4) and $bird(tweety)$. Using arguments about minimal models, we can show that

$$\mathrm{Circum}(\psi; ab_1; fly, bird) \models fly(tweety)\ . \tag{1.5}$$

Abnormality theories can represent two important types of *defeating* arguments. One usually distinguishes between defeaters that contradict outright a default (Type I defeaters), and those where only the justification for a default is undermined, without contradicting its conclusion (Type II defeaters). Here is some illustrating example. If it is known that *tweety* doesn't fly (Type I defeat), we can add (to Example 1.4):

$$bird(tweety) \wedge \neg fly(tweety)\ . \tag{1.6}$$

Here the conclusion $ab_1(tweety)$ follows in all models. For Type II defeat, we simply assert that Tweety is abnormal, without asserting that he doesn't fly.

Abnormality theories are widely used in applications of circumscription. Some major examples are in logic-based diagnosis [Reiter 1987], inheritance theories [Etherington and Reiter 1983], and reasoning about time and action [Baker 1991, Kartha and Lifschitz 1995].

1.4. Autoepistemic Logic (AEL)

Modal logics are an extensively studied basis for formalizing nonmonotonic reasoning. The first proposal was made by McDermott and Doyle [McDermott and Doyle 1980]. In order to avoid some deficiencies of this logic Moore [Moore 1985] introduced autoepistemic logic as a reconstruction of McDermott and Doyle's logic.

The idea is to formalize nonmonotonic reasoning in terms of an ideally rational agent reasoning about its own beliefs. Hence, a modal language $\mathcal{L}_{\mathbf{B}}$ is employed which extends a classical first-order language with a new monadic operator $\mathbf{B}$. So, e.g., $\mathbf{B}(\varphi \wedge (\neg\mathbf{B}\psi \rightarrow \varphi))$ is a formula of $\mathcal{L}_{\mathbf{B}}$ where φ and ψ are classical formulae. For example, Chellas [Chellas 1980] and Hughes and Cresswell [Hughes and Cresswell 1984] are good sources on modal logics.

The intuitive reading of a formula $\mathbf{B}\varphi$ is that the agent believes φ. Nonmonotonicity is captured by the *negative introspection* capability of the agent which amounts to a rule

$$\text{if } \varphi \text{ is not believed, then } \neg\mathbf{B}\varphi \text{ is.}$$

Such a rule is circular, i.e., beliefs are defined in terms of what is not believed, and a formalization of this principle leads to a fixed point equation. In autoepistemic reasoning it is assumed that an ideally rational agent has perfect introspection capabilities. This means that in addition to negative introspection the agent is capable of *positive introspection*:

$$\text{if } \varphi \text{ is believed, then } \mathbf{B}\varphi \text{ is believed.}$$

Moore captures this notion of beliefs by defining the potential sets of beliefs of an ideally rational agent with full introspection capabilities when given a set of premises as the basis of the beliefs. The potential sets of beliefs are called *stable expansions* of the premises and they are defined by the following fixed point equation.

1.8. Definition *(Stable Expansion,Moore [Moore 1985]).*
Let $T \subseteq \mathcal{L}_{\mathbf{B}}$. A set of formulae S is a *stable expansion* of T *if and only if*

$$S = \{\phi \mid T \cup \mathbf{B}S \cup \neg\mathbf{B}\overline{S} \vdash \phi\} \tag{1.7}$$

where $\vdash$ is the classical consequence relation which treats $\mathbf{B}\phi$ formulae like atomic formulae and the following notations are used: $\mathbf{B}S = \{\mathbf{B}\phi \mid \phi \in S\}$, $\neg S = \{\neg\phi \mid \phi \in S\}$, and $\overline{S} = \mathcal{L}_{\mathbf{B}} \setminus S$. Hence, $\neg\mathbf{B}\overline{S} = \{\neg\mathbf{B}\phi \mid \phi \in \mathcal{L}_{\mathbf{B}} \setminus S\}$.

So a stable expansion S is the logical closure of the premises and the results of positive introspection $\mathbf{B}S$ and negative introspection $\neg\mathbf{B}S$ from S.

1.9. Example. A statement of the type “a bird typically flies” can be formalized in autoepistemic logic using a formula

$$b \wedge \neg\mathbf{B}\neg f \rightarrow f.$$

which states that if a bird is not believed not to fly, it flies. The set $\{b, b \wedge \neg\mathbf{B}\neg f \rightarrow f\}$ induces exactly one potential set of beliefs (stable expansion) which contains, e.g., $b, \mathbf{B}b, \neg\mathbf{B}\neg f, f, \mathbf{B}f, \mathbf{BB}f, \mathbf{BBB}f$.

This idea has also been used in [Przymusinski 1998] to define semantics for logic programs (see also [Brass, Dix and Przymusinski 1999]).

In first-order modal logics quantifying into the scope of a modal logic is problematic. Here we consider the case where quantifying-in is not allowed, e.g., formulae of the form $\forall x(\mathbf{B}\neg p(x) \rightarrow p(x))$ are not permitted. With this assumption formulae of the form $\mathbf{B}\varphi$ can be treated as new (ground) atoms as far as the underlying consequence relation ($\vdash$) is concerned. However, we allow formulae with free variables and they are interpreted as schemata representing all of the closed instances. We refer to [Konolige 1994] for a treatment of quantifying-in in autoepistemic logic. Konolige's [Konolige 1994] survey article provides a good source on autoepistemic logic.

The problem of McDermott and Doyle's [McDermott 1982] first attempt was that it missed positive introspection, i.e. they used equation (1.7) *without* the term $\mathbf{B}S$. Another way to adding positive introspection is by using appropriate modal axioms and the necessitation rule

$$\frac{\varphi}{\mathbf{B}\varphi}.$$

This observation leads to a scheme for defining a corresponding nonmonotonic modal logic for a given monotonic modal logic $\mathcal{S}$, i.e., a notion of a potential set of beliefs ($\mathcal{S}$-expansion) for each monotonic modal logic S specifying the choice of the positive introspection capabilities of the agent.

1.10. Definition *(S-Expansion,McDermott and Doyle).*
Let $T \subseteq \mathcal{L}_{\mathbf{B}}$. A set of formulae S is an *$\mathcal{S}$-expansion* of T *if and only if*

$$S = \{\phi \mid T \cup \neg\mathbf{B}\overline{S} \vdash_{\mathcal{S}} \phi\} \tag{1.8}$$

where $\vdash_{\mathcal{S}}$ is the derivability relation of the logic $\mathcal{S}$.

The work on this approach was initiated by McDermott [McDermott 1982] and continued first by Shvarts [Shvarts 1990] and then by Marek et al. [Marek, Schwarz and Truszczyński 1993]. A good source on nonmonotonic modal logics is the book [Marek and Truszczyński 1993]. Marek et al. [Marek et al. 1993] show that the structure of the space of nonmonotonic modal logics is simpler than that of monotonic modal logics. The underlying monotonic modal logic plays a secondary role in the definition of nonmonotonic reasoning based on the fixed point equation (1.8): several monotonic modal logics induce the same nonmonotonic system, i.e. expansions coincide for a range of underlying modal logics. For example, for every modal logic $\mathcal{S}$ between **5** and **KD45**, a $\mathcal{S}$-expansion is a stable expansion. If the set of premises is taken from a restricted class of formulae, even larger ranges exist. For example, let T be strongly stratified [Marek et al. 1993]. Then for every modal logic $\mathcal{S}$ such that[4] $\mathbf{N} \subseteq \mathcal{S}$ and $\mathcal{S} \subseteq \mathbf{KD45}$ or $\mathcal{S} \subseteq \mathbf{SW5}$, [Marek et al. 1993] proved:

S is an $\mathcal{S}$-expansion *if and only if* S is the unique stable expansion of T.

[4]The modal logic $\mathbf{N}$ is an extension of the propositional calculus with no modal axioms but the rule of necessitation.

All nonmonotonic modal logics based on the fixed point equation (1.8) share a property that so called weakly grounded (or circular grounded) beliefs are not excluded from the expansions. For example, $\{\neg\mathbf{B}\neg\mathbf{B}p \rightarrow p\}$ has an $\mathbf{N}$-expansion containing a circular grounded belief p. This property of $\mathbf{N}$-expansions is immediately inherited by other McDermott and Doyle style nonmonotonic modal logics because of the following property of nonmonotonic modal logics:

- for any modal logics $\mathcal{S}_1, \mathcal{S}_2$ contained in $\mathbf{S5}$, if $\mathcal{S}_1$ is contained in $\mathcal{S}_2$, then any $\mathcal{S}_1$-expansion is an $\mathcal{S}_2$-expansion [Marek et al. 1993].

We refer to [Niemelä 1991] for an approach for defining autoepistemic reasoning which avoids weakly grounded beliefs.

Autoepistemic logic is closely related to default logic and the main difference is groundedness. All McDermott and Doyle style logics allow weakly grounded beliefs whereas extensions in default logic are strongly grounded. This needs special attention when establishing translations between the two logics. However, a default theory can be translated into a set of autoepistemic formulae such that extensions coincide with expansions of the translated theory provided that a tightly enough grounded class of expansions is employed. Truszczyński [Truszczyński 1991] proposes the following quite natural translation of a default rule into an autoepistemic formula

$$\mathrm{tr}_{\mathrm{T}}\Big(\frac{\varphi : \psi_1, \ldots, \psi_n}{\gamma}\Big) \quad = \quad \mathbf{B}\varphi \wedge \mathbf{B}\neg\mathbf{B}\neg\psi_1 \wedge \cdots \wedge \mathbf{B}\neg\mathbf{B}\neg\psi_n \rightarrow \gamma \qquad (1.9)$$

and shows that for a wide range of modal logics, extensions and expansions coincide.

1.11. Theorem ([Truszczyński 1991]).
Let $(\mathcal{D}, W)$ be a default theory. For a modal logic $\mathcal{S}$ between $\mathbf{N}$ *and* $\mathbf{S4F}$ *and a consistent set of formulae S, the set S is an extension of* $(\mathcal{D}, W)$ if and only if *S is the first-order part of an* $\mathbf{S}$*-expansion of* $W \cup \{\mathrm{tr}_{\mathrm{T}}(d) \mid d \in \mathcal{D}\}$.

The modal approach to nonmonotonic reasoning can be seen as a generalization of default logic because it can capture default extensions but also provides an expressive query language where distinctions can be made between whether a query is *true*, *believed*, *false* or *not believed.*

1.5. Rational Closure ($\mathcal{K}^{\mathrm{rat}}$)

In the previous sections we have introduced some nonmonotonic reasoning systems. A natural question to ask is what the differences and similarities are and how much *nonmonotonicity* such logics contain? The idea to *abstract* from a particular representation and to ask for the properties a nonmonotonic consequence-relation $\mid\!\sim$should satisfy dates back to [Gabbay 1984] and was investigated in detail by [Kraus, Lehmann and Magidor 1990, Lehmann and Magidor 1992]. [Makinson 1994] is a nice overview article.

Obviously, any nonmonotonic system induces an entailment relation $\mid\!\sim$:

1.12. DEFINITION *(Induced Entailment Relation $\mid\!\sim$).*
For default (resp. autoepistemic) logic, we can define a formula to be derivable if it is contained in all extensions (resp. expansions) of the theory in question. For Circumscription it is even simpler: a formula follows, if it is contained in the circumscribed theory. This approach (truth in all extensions) is often called *sceptical* as opposed to a *credulous* setting, where a formula is already declared to be entailed when it is true in at least one extension (resp. expansion) of the theory under consideration.

It turns out that there are abstract properties of an entailment relation $\mid\!\sim$, that clearly distinguish between default logic and circumscription. While any circumscription logic is *cumulative* (a term to be explained below), default logic is not.

But we can use the $\mid\!\sim$-setting also the other way around, by representing our theories using $\mid\!\sim$. We consider the following set-up:

Default-base $\mathcal{K}$: A given situation is modeled by a (finite) set of *conditional assertions* of the form "$\alpha\mid\!\sim\beta$", where α, β are expressions in propositional logic. This set forms the "default-knowledge-base" $\mathcal{K}$. An assertion "$\alpha\mid\!\sim\beta$" may be read as "*normally (typically, in most cases, if there is no evidence to the contrary) α* implies β". We also allow *hard* knowledge, which is true in any case and not subject to later revision. We code such knowledge with a propositional theory T_{hard}.

Closure of $\mathcal{K}$: Starting from the default-base $\mathcal{K}$ one attempts to derive additional conditional assertions. The set $\mathcal{K}$ of elements of the form "$\alpha\mid\!\sim\beta$" thus leads to a, usually bigger, set again made up of elements of the form "$\alpha\mid\!\sim\beta$". The set of all conditional assertions derived from $\mathcal{K}$ is called the closure of $\mathcal{K}$. We will consider several different closure operators.

Query ?-$c\mid\!\sim d$: Finally we can ask queries about $\mathcal{K}$. These are again of the form "$\gamma\mid\!\sim\delta$" and will be answered by "yes" or "no", depending on whether "$\gamma\mid\!\sim\delta$" is in the closure of $\mathcal{K}$ or not.

1.13. DEFINITION *(Cumulative Closure $\mathcal{K}^{\mathrm{cum}}$).*
The cumulative closure $\mathcal{K}^{\mathrm{cum}}$ of a knowledge base is defined by simply closing the given knowledge base by the following rules: Reflexivity, Left Logical Equivalence, Right Weakening, Cut and Cautious Monotonicity.

Let us define and discuss these inference rules.

$$\frac{}{\alpha\mid\!\sim\alpha} \qquad (\textit{Reflexivity})$$

$$\frac{T_{hard} \vdash \alpha \Leftrightarrow \beta,\ \alpha\mid\!\sim\gamma}{\beta\mid\!\sim\gamma} \qquad (\textit{Left Logical Equivalence})$$

Left Logical Equivalence expresses the requirement that logically equivalent formulas should have the same nonmonotonic consequences. Here $\vdash$ denotes the classical logical consequence relation. Using the well-known completeness theorem for classical logic, we can also write "$T_{hard} \models \phi$". For ease of notation the reference to T_{hard} will be suppressed, unless it is absolutely crucial.

$$\frac{\vdash \alpha \rightarrow \beta,\ \gamma {\mid\!\sim} \alpha}{\gamma {\mid\!\sim} \beta} \qquad \textit{(Right Weakening)}$$

Right Weakening may be paraphrased to say that any logical consequence of a nonmonotonic consequence is again a nonmonotonic consequence. *Reflexivity* and *Right Weakening* suffice to derive $\alpha {\mid\!\sim} \beta$ from $\alpha \vdash \beta$, i.e. the nonmonotonic consequence relation is an extension of the classical consequence relation.

$$\frac{\alpha \wedge \beta {\mid\!\sim} \gamma,\ \alpha {\mid\!\sim} \beta}{\alpha {\mid\!\sim} \gamma} \qquad \textit{(Cut)}$$

Cut expresses the method of added hypothesis. When we try to prove γ from α, we adopt an additional hypothesis β, try to derive γ from $\alpha \wedge \beta$ and then try to get rid of the hypothesis again by proving that β follows from α.

$$\frac{\alpha {\mid\!\sim} \beta,\ \alpha {\mid\!\sim} \gamma}{\alpha \wedge \beta {\mid\!\sim} \gamma} \qquad \textit{(Cautious Monotonicity)}$$

For a monotone consequence relation ${\mid\!\sim}$ the assertion $\alpha \wedge \beta {\mid\!\sim} \gamma$ could always be derived from $\alpha {\mid\!\sim} \gamma$. *Cautious Monotonicity* is a relaxation of this requirement in that monotony only applies, when we add assumptions that have already been proved. Thus cautious monotonicity may be thought of as sanctioning the use of lemmas in proofs.

1.14. Definition *(Preferential Closure $\mathcal{K}^{\mathrm{pref}}$).*
The preferential closure $\mathcal{K}^{\mathrm{pref}}$ of a knowledge base is defined by closing the given knowledge base by the rules: Reflexivity, Left Logical Equivalence, Right Weakening, Cut, Cautious Monotonicity and the following rule

$$\frac{\alpha {\mid\!\sim} \gamma,\ \beta {\mid\!\sim} \gamma}{\alpha \vee \beta {\mid\!\sim} \gamma} \qquad \textit{(Or).}$$

Obviously, $\mathcal{K}^{\mathrm{cum}} \subseteq \mathcal{K}^{\mathrm{pref}}$. But it turns out, that even the preferential closure is fairly weak [Dix and Schmitt 1993]. We illustrate this with the Nixon diamond:

1.15. Example *(Weakness of $\mathcal{K}^{\mathrm{pref}}$).*
We consider the knowledge base

$$\mathcal{K} := \{\, \mathit{quaker} {\mid\!\sim} \mathit{pacifist},\ \mathit{republican} {\mid\!\sim} \neg \mathit{pacifist} \,\}.$$

Although "$\mathit{quaker} \wedge \neg \mathit{republican} {\mid\!\sim} \mathit{pacifist}$" and "$\mathit{republican} \wedge \neg \mathit{quaker} {\mid\!\sim} \neg \mathit{pacifist}$" are both expected to be contained in the preferential closure, they are not.

Often, one would like to derive even stronger statements. For example adding statements that are totally irrelevant with the situation at hand, should not affect it. For example, if Peter normally goes to a party, then learning that (say) Socrates was a Greek philosopher should not change that conclusion. The principle of *Rationality* states that any proposition that is not explicitly negated by a premise set can be added to that set without changing its conclusions:

$$\frac{\alpha \not\mid\!\sim \neg\beta,\ \ \alpha \mid\!\sim \gamma}{\alpha \wedge \beta \mid\!\sim \gamma} \qquad \textit{(Rationality)}$$

Consequently, Lehmann and Magidor defined a rational closure $\mathcal{K}^{\mathrm{rat}}$, which is quite different from both $\mathcal{K}^{\mathrm{cum}}$ and $\mathcal{K}^{\mathrm{pref}}$:

$\underline{\mathcal{K} \rightsquigarrow \mathcal{K}^{\mathrm{cum}}}$**:** This closure operation, as well as $\mathcal{K} \rightsquigarrow \mathcal{K}^{\mathrm{pref}}$ is *monotone*, i.e.

$$\mathcal{K} \subseteq \mathcal{K}' \quad \Longrightarrow \quad \mathcal{K}^{\mathrm{cum}} \subseteq (\mathcal{K}')^{\mathrm{cum}}.$$

The reason for this is that both operations are defined by *inference rules* of the form:

"if $\alpha \mid\!\sim \beta$ and $\gamma \mid\!\sim \delta$ then $\zeta \mid\!\sim \eta$".

$\mathcal{K}^{\mathrm{cum}}$ and $\mathcal{K}^{\mathrm{pref}}$ are the least sets containing $\mathcal{K}$ and closed under a set of inference rules of this form.

$\underline{\mathcal{K} \rightsquigarrow \mathcal{K}^{\mathrm{rat}}}$**:** This operation is *nonmonotonic* and not so easy to describe. But Lehmann and Magidor give in [Lehmann and Magidor 1992] a very effective and intuitive algorithm to compute $\mathcal{K}^{\mathrm{rat}}$. We will restrict ourselves to the explanation of this algorithm.

Two important concepts have to be introduced before we can give the definition of the algorithm that computes $\mathcal{K}^{\mathrm{rat}}$:

1. the notion of an *exceptional formula*,
2. the notion of *rank* for formulas.

1.16. Definition *(Exceptional Formulas).*
A formula α is called *exceptional* for $\mathcal{K}$, if $\mathbf{t} \mid\!\sim \neg\alpha \in \mathcal{K}^{\mathrm{pref}}$. A conditional assertion $\alpha \mid\!\sim \beta$ is called *exceptional* for $\mathcal{K}$, if α is exceptional for $\mathcal{K}$.

Thus α is exceptional for $\mathcal{K}$ means that normally $\neg\alpha$ holds true (given $\mathcal{K}$). The set of all exceptional conditional assertions in $\mathcal{K}$ will be denoted by *E(*$\mathcal{K}$*)*. Iterating the operator E leads to a sequence $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2$, ...

$$\mathcal{C}_0 := \mathcal{K},\ \mathcal{C}_{i+1} := E(\mathcal{C}_i).$$

This is a decreasing sequence, i.e. $\mathcal{C}_{i+1} \subseteq \mathcal{C}_i$ is true for all i. Since $\mathcal{K}$ is finite there is an index n, such that

$$\mathcal{C}_1 \supset \mathcal{C}_2 \supset \ldots \supset \mathcal{C}_{n-1} \supset \mathcal{C}_n = \mathcal{C}_{n+1} = \ldots.$$

The `rank` of a formula α with respect to a knowledge base $\mathcal{K}$ is the number k such that $\alpha \in \mathcal{C}_k$ but $\alpha \notin \mathcal{C}_{k+1}$. If there is no such k, α is said to have no rank.

1.17. DEFINITION *($\mathcal{K}^{\mathrm{rat}}$).*
Let $\mathcal{K}$ be a knowledge base. $\mathcal{K}^{\mathrm{rat}}$ consists of all conditional assertions $\alpha{\mid\!\sim}\beta$, such that either

1. the rank of α is strictly smaller than the rank of $\alpha \wedge \neg\beta$ (this includes the case that α has a rank and $\alpha \wedge \neg\beta$ does not) or
2. α has no rank.

Note that the definition of $\mathcal{K}^{\mathrm{rat}}$ depends essentially on $\mathcal{K}^{\mathrm{pref}}$. In fact, as shown in [Dix and Schmitt 1993], there exists a nice relation between these two closures:

$$\alpha{\mid\!\sim}\beta \ \in \mathcal{K}^{\mathrm{pref}} \ \ \textit{if and only if} \ \ \alpha{\mid\!\sim}\mathbf{f} \ \in (\mathcal{K} \cup \{\alpha{\mid\!\sim}\neg\beta\})^{\mathrm{rat}}.$$

This means that once we have an efficient method for computing $\mathcal{K}^{\mathrm{rat}}$we can use it for the computation of $\mathcal{K}^{\mathrm{pref}}$.

In [Dix and Schmitt 1993] many of the benchmark problems of [Lifschitz 1989] are discussed and it is investigated how they are handled by the various closures introduced above.

2. Automating General Nonmonotonic Logics

In this section we address the problem of automating general nonmonotonic logics and, in particular, default logic, autoepistemic logic and circumscription. We start the section by a brief survey of the complexity and decidability results on nonmonotonic logics. The idea is to provide an overview of the computational properties of nonmonotonic logics and discuss the implications of the properties on automated reasoning in these logics. Given this background we move towards automated theorem proving methods. The first issue is the problem of developing *finitary* characterizations of correct nonmonotonic reasoning. As we have seen in Section 1, when formalizing nonmonotonic logics such as default logic or circumscription the definitions are given in terms of infinite objects: fixed point equations over infinite sets of formulae in the case of default logic and autoepistemic logic or minimality over an infinite set of models in the case of circumscription. For developing computational treatments of the logics, it is necessary to find techniques for characterizing correct nonmonotonic reasoning using finitary means. These characterizations provide the basis for the decidability and complexity results as well as for the automated theorem proving methods. We present such characterizations and demonstrate how they lead to efficient decision methods.

This section is organized as follows. After a survey on general complexity results in Subsection 2.1, we turn to finitary characterizations of our nonmonotonic logics, which form the basis of the decision methods studied in the following Subsections 2.3, 2.4, 2.5 and 2.6. We give the most attention to default logic (Subsection 2.3) because it is perhaps the most popular general nonmonotonic logic which has close relationships to logic programs. In fact the method presented here for default logic provides the basis of our treatment of the stable model semantics of logic programs in Section 5. In Subsection 2.4 we show how the approach developed for default logic can be applied to autoepistemic logic. We discuss also the fundamental issues in automating circumscription (Subsection 2.5) and provide the basis for the treatment of circumscription presented in Section 3. We end with Subsection 2.6 by considering the automation of rational closure.

2.1. Decidability and Complexity

The fundamental decision problems for default logic and nonmonotonic modal logics 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/expansion (the *extension/expansion existence problem*). Circumscription can been seen as a form of sceptical reasoning because circumscriptive consequences are the formulae that are true in every minimal model of the premises.

First-Order Case

For first-order default logic these problems are not even semi-decidable [Reiter 1980]. This holds also for first-order autoepistemic logic even when quantifying into the scope of the modal operator is not allowed [Niemelä 1990]. If quantifying-in is allowed, the problems are highly undecidable [Konolige 1991]. General first-order circumscription is highly uncomputable, too [Schlipf 1986].

Propositional Case

Default logic 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 [Gottlob 1992, Stillman 1992]. Balcázar et al. [Balcázar, Díaz and Gabarró 1988] and Garey and Johnson [Garey and Johnson 1979] provide good sources of basic definitions and results on the polynomial time hierarchy and related issues in complexity theory. Kautz and Selman [Kautz and Selman 1991], Stillman [Stillman 1990], and Ben-Eliyahu and Dechter [Ben-Eliyahu and Dechter 1991, Ben-Eliyahu and Dechter 1992] have obtained further complexity results on restricted subclasses of default theories.

Nonmonotonic modal logics Similar results as for default logic hold. For autoepistemic logic based on Moore style expansions and L-hierarchic expansions [Niemelä 1991] and for McDermott and Doyle style logics based on a number of modal logics *expansion existence problem* and *credulous reasoning* are $\mathbf{\Sigma_2^P}$-complete problems and *sceptical reasoning* is $\mathbf{\Pi_2^P}$-complete [Gottlob 1992, Niemelä 1990, Niemelä 1992, Schwarz and Truszczyński 1993]. [Marek and Truszczyński 1991*a*, Marek and Truszczyński 1991*b*] present complexity results on restricted subclasses of autoepistemic reasoning.

Circumscription The problem of deciding whether a propositional sentence is a circumscriptive consequence of a set of propositional sentences is also a $\mathbf{\Pi_2^P}$-

complete problem [Eiter and Gottlob 1993, Eiter and Gottlob 1995*a*]. Cadoli and Lenzerini [Cadoli and Lenzerini 1990] establish complexity results on limited subclasses of circumscriptive reasoning including some tractability results.

Implications of the Results

The complexity results have interesting implications on theorem proving methods for nonmonotonic logics 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 nonmonotonic logics 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, the logics are 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 [Chandra and Harel 1985, Apt, Blair and Walker 1988, Van Gelder 1988]. We refer to [Niemelä and Rintanen 1994] 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 [Van Gelder, Ross and Schlipf 1991] and the stationary semantics for default logic [Przymusinska and Przymusinski 1994, Gottlob 1995]. 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 [Gogic et al. 1995, Cadoli et al. 1997].

2.2. Finitary Characterization

In this section we elaborate the definitions of default logic, autoepistemic logic and circumscription and present characterizations of reasoning in these logics with the aim of providing the basis for the automated theorem proving methods. We start by considering default logic and then proceed with autoepistemic logic and circumscription.

Default Logic

The key objects of interest in default logic are the extensions of a default theory. [Reiter 1980] 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 formulas W and the consequents of a subset GD of the defaults $\mathcal{D}$ (the generating defaults).

2.1. Theorem (Generating Defaults).
Let E be an extension of a default theory $(\mathcal{D}, W)$. Then there is a set $\mathrm{GD} \subseteq \mathcal{D}$ *such that* $E = \mathrm{Cn}(W \cup \{\mathrm{cons}(d) \mid d \in \mathrm{GD}, \mathrm{pre}(d) \in E$ *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 $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D})}$ introduced in Section 1, Definition 1.3. $\mathrm{Cn}_{\mathrm{Mon}(\mathcal{D})}$ stands for the consequence relation induced by the monotonic versions of the default rules.

The first characterization is based on examining default rules directly. This approach has been presented in several papers [Marek and Truszczyński 1993, Risch and Schwind 1994, Baader and Hollunder 1995]. The idea is to project the conditions fulfilled by an extension to individual default rules. First, extensions are grounded (see Definition 1.4) which implies that the corresponding set of generating defaults must be grounded as well.

A set of defaults $\mathcal{D}$ is grounded in a set formulae W if and only if for all $d \in \mathcal{D}$, $\text{pre}(d) \in \text{Cn}_{\text{Mon}(\mathcal{D})}(W)$ where $\text{Mon}(\mathcal{D}) = \{\frac{\text{pre}(d)}{\text{cons}(d)} \mid d \in \mathcal{D}\}$. Second, extensions are closed under default rules, i.e. all applicable defaults are applied.

2.2. THEOREM (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 $E = \text{Cn}(W \cup \{\text{cons}(d) \mid d \in \mathcal{D}'\})$ for a set of defaults $\mathcal{D}' \subseteq \mathcal{D}$ such that

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

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

The second characterization is introduced in [Niemelä 1995*b*] and is based on focusing 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 derivation.

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

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

$$\text{for all } \psi \in \text{just}(\mathcal{D}){:}\;\; \psi \in \text{F} \iff \neg\psi \notin \text{Cn}_{\text{Mon}(\mathcal{D},\text{F})}(W)$$

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

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

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

2.5. EXAMPLE. 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 $\text{just}(\mathcal{D}) = \{\neg s, \neg r\}$. Consider $\text{F} = \{\neg r\}$. It is $(\mathcal{D}, W)$-full because $\text{Mon}(\mathcal{D}, \text{F}) = \{\frac{q}{s}\}$, $\neg\neg r \notin \text{Cn}_{\text{Mon}(\mathcal{D},\text{F})}(W)$ but $\neg\neg s \in \text{Cn}_{\text{Mon}(\mathcal{D},\text{F})}(W)$. Hence, $\text{Cn}_{\text{Mon}(\mathcal{D},\text{F})}(W) = \text{Cn}(W \cup \{s\})$ is an extension of $(\mathcal{D}, W)$. For instance, $\text{F}' = \{\neg s\}$ is not $(\mathcal{D}, W)$-full, because $\text{Mon}(\mathcal{D}, \text{F}') = \{\frac{p}{r}\}$ and, e.g., $\neg\neg r \notin \text{Cn}_{\text{Mon}(\mathcal{D},\text{F}')}(W)$ but $\neg r \notin \text{F}'$. It is easy to verify that F is the only $(\mathcal{D}, W)$-full set and, thus, $\text{Cn}_{\text{Mon}(\mathcal{D},\text{F})}(W)$ is the unique extension of $(\mathcal{D}, W)$.

Autoepistemic Logic

For autoepistemic expansions we present a characterization based on *full sets* [Niemelä 1990, Niemelä 1992]. A full set is constructed from the $\mathbf{B}\chi$ and $\neg\mathbf{B}\chi$ subformulae of the premises and it serves as the kernel of a stable expansion; it uniquely characterizes the expansion. An overview of other approaches to characterizing stable expansions can be found in [Niemelä 1992].

The characterization is based on the set of all subformulae of the form $\mathbf{B}\chi$ in a set of premises T. We denote this set by $\mathrm{Sf}_{\mathbf{B}}(T)$. Notice that in the characterization only the classical derivability relation $\vdash$ is used where $\mathbf{B}p$ formulae are treated as new atoms and no modal consequence relation is needed.

2.6. Definition *(Full Sets).*
For a set of sentences T, a set $\mathrm{F} \subseteq \mathrm{Sf}_{\mathbf{B}}(T) \cup \neg\mathrm{Sf}_{\mathbf{B}}(T)$ is T-full if and only if the following two conditions hold for every $\mathbf{B}\chi \in \mathrm{Sf}_{\mathbf{B}}(T)$:

1. $T \cup \mathrm{F} \vdash \chi$ if and only if $\mathbf{B}\chi \in \mathrm{F}$.
2. $T \cup \mathrm{F} \not\vdash \chi$ if and only if $\neg\mathbf{B}\chi \in \mathrm{F}$.

In fact, for a T-full set F, the classical consequences of $T \cup \mathrm{F}$ provide the non-modal part of an expansion. It is known that this uniquely determines the expansion. The whole expansion including the modal formulae can be constructed from the non-modal part. First such construction was proposed by [Marek 1989]. Here we present an alternative construction presented in [Niemelä 1992] which is more suitable for automation. For this we employ a restricted notion of subformulae: $\mathrm{Sf}^p_{\mathbf{B}}(\varphi)$ is the set of *primary* subformulae of φ, i.e., all subformulae of the form $\mathbf{B}\chi$ of φ which are not in the scope of another $\mathbf{B}$ operator in φ. Hence, $\mathrm{Sf}^p_{\mathbf{B}}(\mathbf{B}(\neg\mathbf{B}p \to q) \wedge \mathbf{B}\neg q) = \{\mathbf{B}(\neg\mathbf{B}p \to q), \mathbf{B}\neg q\}$. The construction uses a simple consequence relation $\models_{\mathbf{B}}$ which is given recursively on top of the classical derivability relation $\vdash$. It turns out that this consequence relation corresponds exactly to membership in an expansion when given its characterizing full set.

2.7. Definition *($\mathbf{B}$-consequence).*
Given a set of sentences T and a sentence φ,

$$T \models_{\mathbf{B}} \varphi \quad \text{if and only if } T \cup \mathrm{SB}_T(\varphi) \vdash \varphi$$

where $\mathrm{SB}_T(\varphi) = \{\mathbf{B}\chi \in \mathrm{Sf}^p_{\mathbf{B}}(\varphi) \mid T \models_{\mathbf{B}} \chi\} \cup \{\neg\mathbf{B}\chi \in \neg\mathrm{Sf}^p_{\mathbf{B}}(\varphi) \mid T \not\models_{\mathbf{B}} \chi\}$.

For an expansion S of T, there is a corresponding T-full set

$$\{\mathbf{B}\varphi \in S \mid \mathbf{B}\varphi \in \mathrm{Sf}_{\mathbf{B}}(T)\} \cup \{\neg\mathbf{B}\varphi \in S \mid \mathbf{B}\varphi \in \mathrm{Sf}_{\mathbf{B}}(T)\}$$

and for a T-full set F,

$$\{\varphi \in \mathcal{L}_{\mathbf{B}} \mid T \cup \mathrm{F} \models_{\mathbf{B}} \varphi\}$$

is a stable expansion of T. In fact it can be shown [Niemelä 1992] that there is a one-to-one correspondence between full sets and expansions.

2.8. Theorem (Expansions in Terms of Full Sets).
Let T be a set of sentences of $\mathcal{L}_{\mathbf{B}}$. Then a function SE_T *defined as*

$$\mathrm{SE}_T(\mathrm{F}) = \{\varphi \in \mathcal{L}_{\mathbf{B}} \mid T \cup \mathrm{F} \models_{\mathbf{B}} \varphi\}$$

gives a bijective mapping from the set of T-full sets to the set of stable expansions of T and for a T-full set F, $\mathrm{SE}_T(\mathrm{F})$ *is the unique stable expansion S of T such that* $\mathrm{F} \subseteq \mathbf{B}S \cup \neg\mathbf{B}\overline{S}$.

2.9. Example. The premise $T = \{\neg\mathbf{B}\neg\mathbf{B}p \rightarrow p\}$ has two T-full sets

$$\mathrm{F}_1 = \{\mathbf{B}\neg\mathbf{B}p, \neg\mathbf{B}p\} \text{ and } \mathrm{F}_2 = \{\neg\mathbf{B}\neg\mathbf{B}p, \mathbf{B}p\}.$$

For example, F_1 is T-full as $T \cup \mathrm{F}_1 \vdash \neg\mathbf{B}p$ and $T \cup \mathrm{F}_1 \not\vdash p$. This implies that T has exactly two stable expansions $\mathrm{SE}_T(\mathrm{F}_1)$ and $\mathrm{SE}_T(\mathrm{F}_2)$. As $T \cup \mathrm{F}_1 \not\models_{\mathbf{B}} p$, $p \notin \mathrm{SE}_T(\mathrm{F}_1)$. However, $\neg\mathbf{B}\mathbf{B}p \in \mathrm{SE}_T(\mathrm{F}_1)$ as $T \cup \mathrm{F}_1 \models_{\mathbf{B}} \neg\mathbf{B}\mathbf{B}p$.

Circumscription

General circumscription is highly undecidable and computationally very hard to handle and we concentrate here on parallel predicate circumscription in the clausal case and with respect to Herbrand interpretations. In parallel predicate circumscription the predicates are partitioned into three disjoint sets:

1. *minimized*, denoted by $\underline{p}$,
2. *varying*, denoted by $\underline{f}$, and
3. *fixed*, denoted usually by $\underline{z}$ (in this section we will simply omit it)

predicates. The key notion in circumscription is that of a minimal model introduced in Subsection 1.3, Definition 1.6. We present a characterization of minimal models where the minimality of a model can be determined independently of other models using a test for classical consequence. This characterization is proposed by [Niemelä 1996*a*]. A similar characterization but for the propositional case has been used in [Eiter and Gottlob 1995*a*] in the study of the computational complexity of propositional circumscription.

2.10. Definition *(Grounded Models).*
Let T be a set of clauses and let $\underline{p}$ and $\underline{f}$ be sets of predicates. A Herbrand interpretation M is said to be *grounded* in $\langle T, \underline{p}, \underline{f}\rangle$ if and only if for all ground atoms $p(\vec{t})$ such that $p \in \underline{p}$, $M \models p(\vec{t})$ implies $T \cup \mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) \models p(\vec{t})$ where

$$\begin{aligned} \mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) &= \{\neg q(\vec{t}) \mid q(\vec{t}) \text{ is a ground atom}, q \in \underline{p} \cup \underline{f}, M \not\models q(\vec{t})\} \cup \\ &\quad \{q(\vec{t}) \mid q(\vec{t}) \text{ is a ground atom}, q \in \underline{f}, M \models q(\vec{t})\} \end{aligned}$$

2.11. Theorem (Minimal Models).
Let $\underline{p}$ and $\underline{f}$ be the sets of minimized and fixed predicates, respectively. A Herbrand interpretation M is a minimal model of T if and only if M is a model of T and grounded in $\langle T, \underline{p}, \underline{f}\rangle$.

2.12. Example. Let $T = \{p(x) \vee \neg q(x)\}$ and let the underlying language have only one ground term a. Then the Herbrand base is $\{p(a), q(a)\}$. Let the set of minimized predicates $\underline{p} = \{p\}$. If the set of fixed predicates $\underline{f} = \{q\}$, then the Herbrand interpretation $M = \{p(a), q(a)\}$, which is a model of T, is grounded in $\langle T, \underline{p}, \underline{f}\rangle$ because $\mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) = \{q(a)\}$ and $T \cup \mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) \models p(a)$ holds. Hence, M is a minimal model of T. If $F = \emptyset$, then M is not grounded in $\langle T, \underline{p}, \underline{f}\rangle$ because $\mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) = \emptyset$ and $T \cup \mathrm{N}^{\langle \underline{p}, \underline{f}\rangle}(M) \models p(a)$ does not hold. Thus, with no fixed predicates M is not a minimal model of T.

2.3. Algorithms 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 2.1) can be found, e.g., in [Marek and Truszczyński 1993, Baader and Hollunder 1995, Risch and Schwind 1994]. There are approaches where default reasoning is reduced into another problem like a truth maintenance problem [Junker and Konolige 1990] or a constraint satisfaction problem [Ben-Eliyahu and Dechter 1991]. An interesting approach to provide proof theory for default reasoning based on sequent calculus is proposed by [Bonatti 1996, Bonatti and Olivetti 1997]. 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 [Niemelä 1995*b*]. 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. 2 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 2.4 it 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.

S-answer:= Yes; C-answer:= No;
For every subset S of just($\mathcal{D}$) **do**
  If S is $(\mathcal{D}, W)$-full **then**
    If $\varphi \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},S)}(W)$ **then**
      C-answer:= Yes
    else
      S-answer:= No
    end if
  end if

Figure 2: A Naive Decision Procedure for Credulous and Sceptical DL

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., [Fitting 1990]). (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. 3 presents the skeleton of the optimized decision procedure given as a function **extensions**. It employs four functions

- **expand** (for approximations),
- **conflict** (for conflict detection),
- **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 5.

As given in Definition 2.3, 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

function extensions$((\mathcal{D}, W), \mathrm{FS}, \varphi)$
$\mathrm{FS}' :=$ **expand**$((\mathcal{D}, W), \mathrm{FS})$;
if conflict$((\mathcal{D}, W), \mathrm{FS}')$ returns Yes **then** return No
else if FS' covers the negation of each justification in $\mathcal{D}$ **then**
  return **test**$((\mathcal{D}, W), \mathrm{FS}', \varphi)$
else
  $\chi :=$ **choose**$((\mathcal{D}, W), \mathrm{FS}')$;
  if extensions$((\mathcal{D}, W), \mathrm{FS}' \cup \{\chi\}, \varphi)$ returns Yes **then** return Yes
  else return **extensions**$((\mathcal{D}, W), \mathrm{FS}' \cup \{\mathrm{compl}(\chi)\}, \varphi)$
  end if
end if

Figure 3: An optimized decision procedure for default logic

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, an 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 $\{\mathrm{not}(\neg\neg p), \mathrm{not}(\neg q), \neg\neg r, \neg s\}$. 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 $\mathrm{not}(\cdot)$ operator. For instance, when we discuss the full set

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

what is actually meant is the set $\mathrm{F} = \{\neg p, q, \neg r\}$. 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 $\mathrm{Mon}(\mathcal{D}, \mathrm{FS})$ which denotes the set

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

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 $\varphi \in \mathrm{FS}$, $\varphi \in S$ and for all $\mathrm{not}(\varphi) \in \mathrm{FS}$, $\varphi \notin S$. For example, the set $\{a, b\}$ agrees with $\{a, \mathrm{not}(c)\}$ but not with $\{a, \mathrm{not}(b)\}$.
- A set of formulae and not-formulae FS *covers* a formula φ if either $\varphi \in \mathrm{FS}$ or $\mathrm{not}(\varphi) \in \mathrm{FS}$. For example, the set $\{\mathrm{not}(a), b\}$ covers a. A set of formulae S is covered by FS if each formula in S is covered by FS.

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

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**$((\mathcal{D}, W), \mathrm{FS}, \varphi)$ returns Yes if and only if there is an extension E agreeing with FS such that **test**$((\mathcal{D}, W), \mathrm{FS}', \varphi)$ 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**$((\mathcal{D}, W), \mathrm{FS}', \varphi)$ 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**$((\mathcal{D}, W), \mathrm{FS}', \varphi)$ 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 not(ψ). 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 not(ψ) is returned, determines in which order the alternatives are examined. Extensions containing ψ are covered by calling **extensions**$((\mathcal{D}, W), \mathrm{FS}' \cup \{\psi\}, \varphi)$ and extensions not containing ψ are covered by calling **extensions**$((\mathcal{D}, W), \mathrm{FS}' \cup \{\mathrm{not}(\psi)\}, \varphi)$.

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

choose: choose$((\mathcal{D}, W), \mathrm{FS})$ 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: expand$((\mathcal{D}, W), \mathrm{FS})$ 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 **conflict**$((\mathcal{D}, W), \mathrm{FS})$ returns Yes, there is no extension agreeing with FS and (ii) if FS covers negations of every justification in $(\mathcal{D}, W)$, and for some $\mathrm{not}(\chi) \in \mathrm{FS}$, $\chi \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$ or for some $\chi \in \mathrm{FS}$, $\chi \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$, then **conflict**$((\mathcal{D}, W), \mathrm{FS})$ returns Yes.

Under these assumptions we can establish the following theorem.

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

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

1. *Let* $\mathbf{test}((\mathcal{D}, W), \mathrm{FS}, \varphi)$ *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* $\mathbf{test}((\mathcal{D}, W), \mathrm{FS}, \varphi)$ *return Yes if and only if* $\varphi \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$.
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* $\mathbf{test}((\mathcal{D}, W), \mathrm{FS}, \varphi)$ *return Yes if and only if* $\varphi \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$.
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: $\mathbf{LB}((\mathcal{D}, W), \mathrm{FS})$ is characterized with the property: every extension E of $(\mathcal{D}, W)$ agreeing with FS agrees with $\mathbf{LB}((\mathcal{D}, W), \mathrm{FS})$ and

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

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 $\mathrm{not}(\neg\varphi)$ for all $\neg\varphi$ not in the upper bound. Of course, this can be iterated which leads to the following implementation:

$\mathbf{expand}((\mathcal{D}, W), \mathrm{FS})$ returns the smallest set of formulae FS' extending FS and closed under the following rules:

- if $\neg\varphi \in \mathbf{LB}((\mathcal{D}, W), \mathrm{FS}')$ for $\varphi \in \mathrm{just}(\mathcal{D})$, then include $\neg\varphi$ in FS' and
- if $\neg\varphi \notin \mathbf{UB}((\mathcal{D}, W), \mathrm{FS}')$ for $\varphi \in \mathrm{just}(\mathcal{D})$, then include $\mathrm{not}(\neg\varphi)$ in FS'.

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

$\mathbf{conflict}((\mathcal{D}, W), \mathrm{FS})$ returns Yes if and only if for some formula φ,

$$\mathrm{not}(\varphi) \in \mathrm{FS} \text{ but } \varphi \notin \mathbf{UB}((\mathcal{D}, W), \mathrm{FS}) \text{ or}$$
$$\varphi \in \mathrm{FS} \text{ but } \varphi \notin \mathbf{LB}((\mathcal{D}, W), \mathrm{FS}).$$

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**.

2.15. EXAMPLE. Consider the trivial bounds where $\mathbf{LB}((\mathcal{D}, W), \mathrm{FS})$ is the empty set and $\mathbf{UB}((\mathcal{D}, W), \mathrm{FS})$ is the whole language, a default theory $\mathcal{D} = \{:p/p\}$, $W = \{\neg p\}$ and a set $\mathrm{FS} = \{\mathrm{not}(\neg p)\}$. Clearly, the second condition for **conflict** does not hold: FS covers negations of every justification in $\mathcal{D}$, $\mathrm{not}(\neg p) \in \mathrm{FS}$ but $\neg p \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$. However, given the trivial upper bound, $\mathbf{conflict}((\mathcal{D}, W), \mathrm{FS})$ 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 negations of every justification in $(\mathcal{D}, W)$, $\mathbf{conflict}((\mathcal{D}, W), \mathrm{FS})$ returns Yes if for some $\mathrm{not}(\chi) \in \mathrm{FS}$, $\chi \in \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$ or for some $\chi \in \mathrm{FS}$, $\chi \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$.

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.

2.16. EXAMPLE. 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. 2.

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 tasks of deciding whether r is a sceptical consequence of $(\mathcal{D}, W)$. Hence, the function $\mathbf{test}((\mathcal{D}, W), \mathrm{FS}, \varphi)$ returns Yes if and only if $\varphi \notin \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$ and the decision procedure is started by a call to $\mathbf{extensions}((\mathcal{D}, W), \emptyset, r)$. First $\mathrm{FS} = \emptyset$ 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: $\{\neg p, \neg q\} \subseteq \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W)$ where $\mathrm{FS} = \{\mathrm{not}(\neg p), \mathrm{not}(\neg q)\}$.

Hence, the procedure backtracks and $\mathbf{extensions}((\mathcal{D}, W), \{\mathrm{not}(\neg p), \neg q\}, r)$ is called. Now the post-check is passed and $\{\mathrm{not}(\neg p), \neg q\}$ covers the negations of justifications in $\mathcal{D}$. 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 5 where we apply this method to logic programming semantics.

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, $\mathrm{Mon}(\mathcal{D}, \mathrm{FS}) = \{\frac{\mathrm{pre}(d)}{\mathrm{cons}(d)} \mid d \in \mathcal{D},$ for all $\psi \in \mathrm{just}(\mathcal{D}), \mathrm{not}(\neg\psi) \in \mathrm{FS}\}$ can be added without compromising the characterizing property of the lower bound. These considerations lead to quite a tight lower bound:

$$\mathbf{LB}((\mathcal{D}, W), \mathrm{FS}) = \mathrm{Cn}_{\mathrm{Mon}(\mathcal{D},\mathrm{FS})}(W \cup \mathrm{FS}^+) \tag{2.1}$$

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

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

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

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

This development leads to a rather tight upper bound:

$$\mathbf{UB}((\mathcal{D}, W), \mathrm{FS}) = \mathrm{Cn}_{\mathcal{D}^{\mathrm{FS}}}(W) \tag{2.2}$$

The bounds given in equations (2.1) and (2.2) 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 (2.1) and (2.2) can be determined by deciding membership in the closure over classical consequences and some monotonic rules. This problem is **co-NP**-complete [Gottlob 1995] 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 the 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 $\mathrm{FS} = \{\varphi\}$ instead of $\mathrm{FS} = \emptyset$ and when sceptical consequence needs to be decided, use FS $= \{\mathrm{not}(\varphi)\}$. 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. [Schaub 1995, Schaub and Brüning 1996] for work in this direction.

2.4. Algorithms for Autoepistemic Logic

Now we turn to autoepistemic logic and show how the approach introduced for default reasoning leads to a decision procedure for autoepistemic reasoning. The development is based on the work in [Niemelä 1995*a*] where an overview of decision methods for autoepistemic logic can be found. In autoepistemic reasoning the full sets are constructed from $\mathbf{B}\varphi$ subformulae of the premises and their negations. The $\mathbf{B}$ operator provides the means to represent assumptions about the membership in expansions: $\mathbf{B}\chi$ in the full set says that χ belongs to the expansion and $\neg\mathbf{B}\chi$ in the full set asserts that χ is not contained in the expansion. For discussing the procedure we need the notions of *agreeing* , *covering* and *complement*. These remain essentially same as in the case of default logic.

- We say that a set of formulae S *agrees* with a set F of $\mathbf{B}\chi$ formulae and their complements if for every $\mathbf{B}\varphi \in$ F, $\varphi \in S$ and for each $\neg\mathbf{B}\varphi \in$ F, $\varphi \notin S$.
- A set F of $\mathbf{B}\chi$ formulae and their complements *covers* a formula $\mathbf{B}\varphi$ if either $\mathbf{B}\varphi \in$ F or $\neg\mathbf{B}\varphi \in$ F.
- The classical complement of χ is denoted by: $\mathrm{compl}(\chi) = \neg\chi$ and $\mathrm{compl}(\neg\chi) = \chi$.

We can still employ the function **extensions** in Fig. 3 which now takes as input (i) a set of autoepistemic formulae T (instead of a default theory $(\mathcal{D}, W)$), (ii) a set FS of $\mathbf{B}\varphi$ formulae and their complements which determines the common part of the expansions to be considered and (iii) an autoepistemic formula φ which is just passed as an argument to the function **test**. Furthermore, the covering test is rephrased as "**else if** FS$'$ covers all $\mathbf{B}\psi$ subformulae of T **then** ..."

Soundness and completeness of the procedure can be guaranteed under similar assumptions as in the case of default logic. The only difference is that autoepistemic expansions are more weakly grounded than default extensions. Hence, the conditions for the function **conflict** have to be adjusted. We use the notation $\text{AE-Cn}(T) = \{\varphi \in \mathcal{L}_{\mathbf{B}} \mid T \vdash \varphi\}$.

choose: $\mathbf{choose}(T, \mathrm{FS})$ returns either $\mathbf{B}\psi$ or $\neg\mathbf{B}\psi$ for a $\mathbf{B}\psi$ subformula of T not covered by FS.

expand: $\mathbf{expand}(T, \mathrm{FS})$ returns a set FS' of $\mathbf{B}\psi$ formulae and their complements which extends FS and with which every expansion of T agreeing with FS agrees.

conflict: (i) If $\mathbf{conflict}(T, \mathrm{FS})$ returns Yes, there is no expansion agreeing with FS and (ii) if FS covers every $\mathbf{B}\psi$ subformula of T, and for some $\neg\mathbf{B}\chi \in \mathrm{FS}$, $\chi \in \text{AE-Cn}(T \cup \mathrm{FS})$ or for some $\mathbf{B}\chi \in \mathrm{FS}$, $\chi \notin \text{AE-Cn}(T \cup \mathrm{FS})$ then $\mathbf{conflict}(T, \mathrm{FS})$ returns Yes.

Under these assumptions we can establish the following theorem.

2.17. THEOREM (Soundness and completeness).
Let T be a set of autoepistemic formulae and FS *a set of $\mathbf{B}\chi$ subformulae of T and their complements. Then* $\mathbf{extensions}(T, \mathrm{FS}, \varphi)$ *returns Yes if and only if there exists an expansion of T agreeing with* FS *such that* $\mathbf{test}(T, \mathrm{FS}', \varphi)$ *returns Yes where the* FS' *is the corresponding full set.*

In order to develop decision procedures for the different reasoning tasks the only problem is to devise an appropriate **test** function. This can be done using the consequence relation $\models_{\mathbf{B}}$ corresponding to membership in an expansion when given its full set.

2.18. COROLLARY. *Let T be a set of autoepistemic formulae and φ an autoepistemic formula.*

1. *Let* $\mathbf{test}(T, \mathrm{FS}, \varphi)$ *return Yes for all T,* FS*, and φ. Then* $\mathbf{extensions}(T, \emptyset, \varphi)$ *returns Yes if and only if T has an expansion.*

2. *Let* $\mathbf{test}(T, \mathrm{FS}, \varphi)$ *return Yes if and only if* $T \cup \mathrm{FS} \models_{\mathbf{B}} \varphi$.
Then $\mathbf{extensions}(T, \emptyset, \varphi)$ *returns Yes if and only if there exists an expansion of T containing φ.*

3. *Let* $\mathbf{test}(T, \mathrm{FS}, \varphi)$ *return Yes if and only if* $T \cup \mathrm{FS} \not\models_{\mathbf{B}} \varphi$.
Then $\mathbf{extensions}(T, \emptyset, \varphi)$ *returns Yes if and only if there exists an expansion of T not containing φ (i.e.,* $\mathbf{extensions}(T, \emptyset, \varphi)$ *returns No if and only if φ is contained in every expansion of T).*

Now it remains to implement the lower and upper bounds. Consider first the lower bound. Similar to the case of default logic the lower bound can safely include the logical consequences of the premises and the assumptions, i.e., $\text{AE-Cn}(T \cup \mathrm{FS})$. This can be made tighter by allowing the necessitation rule

$$\frac{\varphi}{\mathbf{B}\varphi}.$$

We denote the closure of a set of autoepistemic formulae T under the classical consequence $\vdash$ and the necessitation rule by $\text{AE-Cn}_{\mathbf{N}}(T)$.

$$\mathbf{LB}(T, \mathrm{FS}) = \text{AE-Cn}_{\mathbf{N}}(T \cup \mathrm{FS}) \tag{2.3}$$

Obtaining a tight upper bound for autoepistemic expansions is not straightforward. Mainly this is because it is not easy to approximate autoepistemic derivations using simple syntactic methods when arbitrary autoepistemic formulae are allowed. This should be compared to default logic where default rules have a very regular form leading to easy ways of devising upper bounds on the extensions in terms of the potentially applicable defaults. The situation changes somewhat if the autoepistemic formulae are assumed to be in some standard form. See [Niemelä 1995*a*] for a treatment of one such case.

2.5. Implementing Circumscription

Circumscription is somewhat different from default logic and autoepistemic logic as in circumscription the main objects of interest are minimal models whereas in the other two formalisms models play a secondary role and sets of formulae (extensions/expansions) are the key objects. However, surprisingly similar ideas are applicable for automating circumscription as, e.g., default logic. Here we discuss the basic problems in automating circumscription and provide the basis for methods described in Section 3.

A number of approaches have been proposed for implementing circumscriptive reasoning. In some cases circumscription can be replaced by equivalent first-order formulae (see, e.g., [Lifschitz 1985, Doherty, Lukaszewicz and Szalas 1995]). Przymusinski develops a resolution method (MILO resolution) for circumscriptive reasoning [Przymusinski 1989]. Ginsberg presents a technique employing assumption-based truth maintenance [Ginsberg 1989]. Nerode et al. devise a method for handling circumscriptive inference using integer programming techniques [Nerode, Ng and Subrahmanian 1995] and several methods are based on tableau techniques [Olivetti 1992, Inoue, Koshimura and Hasegawa 1992, Bry and Yahya 1996, Niemelä 1996*b*, Niemelä 1996*a*].

In circumscriptive reasoning we are interested in the question whether a formula φ is true in every minimal model of premises T. A straightforward way of implementing this is through enumerating counter-models, i.e., models where the premises are true but the query φ false. The feasibility of this approach builds on the availability of suitable model generating procedures. Fortunately, the idea of doing theorem proving by trying to find a counter-model for the query underlies much of the work in automated deduction. For example, in tableau-based methods the counter-models are quite explicitly represented. However, it is not straightforward to build a circumscriptive theorem prover on top of a classical theorem prover. The complexity results on circumscription ($\mathbf{\Pi_2^P}$-completeness in the propositional case) indicate that it is highly unlikely that this can be done with only a polynomial overhead. Next we discuss the difficulties involved and present an approach to build a circumscriptive prover on top of a classical model generating procedure. In Section 3 we show how this approach can be integrated into an efficient tableau method.

In principle constructing a circumscriptive theorem prover on top of a classical model generating procedure seems straightforward: the procedure looks for counter-models, i.e., models where the premises T are true but the query φ false. All we need to do is to check for each counter-model whether it is a minimal model of T. If this is the case, then we know that φ is not a circumscriptive consequence of T. If no such model is found, φ is a circumscriptive consequence of T. The soundness and completeness of this method relies on the fact that we have a complete model generating procedure, i.e., it generates (at least) all the minimal models. This is very close to the idea introduced for sceptical default reasoning above, i.e., sceptical consequences are established by showing that no extension not containing the query exists.

A difficulty in this approach is the test for minimality. The naive approach (where minimality is established by searching through each smaller model and showing that this is no longer a model of the premises) seems unfeasible because of the potentially high number of smaller models. This is problematic specially in the presence of varying predicates. The characterization of minimal models given in Theorem 2.11 provides a solution to this problems because with it minimality of a model can be tested using a classical consequence test and without explicit comparisons to other models.

We demonstrate how this technique works in the propositional case. Now the models are sets of atomic propositions. Let $\underline{p}$ and $\underline{f}$ be sets of minimized and fixed atoms, respectively. By Theorem 2.11 a model M of a set of clauses T is a minimal model of T if and only if for each minimized atom p true in model M, $T \cup \mathrm{N}^{\langle \underline{p}, \underline{f} \rangle}(M) \models p$ holds where

$$\begin{aligned} \mathrm{N}^{\langle \underline{p}, \underline{f} \rangle}(M) \quad = \quad & \{\neg q(\vec{t}) \mid q \in \underline{p} \cup \underline{f}, M \not\models q(\vec{t})\} \cup \\ & \{q(\vec{t}) \mid q \in \underline{f}, M \models q(\vec{t})\} \end{aligned}$$

This means that for establishing minimality of a model a single classical consequence test is enough: M is a minimal model of T if and only if

$$T \cup \mathrm{N}^{\langle \underline{p}, \underline{f} \rangle}(M) \models p_1 \wedge \cdots \wedge p_n$$

where $p_1, \ldots, p_n$ are the minimized atoms true in M. Furthermore, the literals in $\mathrm{N}^{\langle \underline{p}, \underline{f} \rangle}(M)$ often reduce the complexity of the classical consequence test.

2.19. Example. Consider the case where we minimize every atom and the set T of clauses

$$\begin{gathered} a_1 \vee b_1 \\ \vdots \\ a_n \vee b_n \end{gathered}$$

The problem is to establish whether a model $M = \{a_1, \ldots, a_n\}$ of T is a minimal one. Using Theorem 2.11 this boils down to determining whether

$$T \cup \{\neg b_1, \ldots \neg b_n\} \models a_1 \wedge \cdots \wedge a_n$$

holds which is a trivial task for an efficient propositional theorem prover. This can be compared to the naive approach where the minimality can be established by testing that none of the subsets of M is a model of T in this case where there are no varying atoms.

2.6. Implementing Rational Closure

In Section 1, Definition 1.17 for $\mathcal{K}^{\mathrm{rat}}$ was quite complicated. Also the very definition of $\mathcal{K}^{\mathrm{pref}}$, closing a set of assertions under some inference rules, does not give any hints for an efficient computation. [Lehmann and Magidor 1992] give the following algorithm for computing $\mathcal{K}^{\mathrm{rat}}$:

$\mathcal{K}$: Knowledge base,
?- $\alpha \mid\!\sim \beta$: query to $\mathcal{K}$

C := $\mathcal{K}$;
while (α exceptional for C and E(C) $\neq$ C) **do**
  C := E(C) ;
if (α is exceptional for C)
  output "yes"   {α has a rank}
else
  if ($\alpha \wedge \neg\beta$ is exceptional for C)
   output "yes"
  else
   output "no";

This algorithm outputs *yes if and only if* $\alpha \mid\!\sim \beta \in \mathcal{K}^{\mathrm{rat}}$.

How can this algorithm be implemented? Fortunately, there is an efficient method which depends on a propositional theorem prover. Lehmann and Magidor defined:

2.20. Definition *($\widetilde{\mathcal{K}}$).*
Let $\widetilde{\mathcal{K}}$ be the set obtained from $\mathcal{K}$ by replacing all occurrences of $\mid\!\sim$ with the material implication $\rightarrow$. Thus $\widetilde{\mathcal{K}}$ is a set of ordinary propositional formulas.

The main notion for defining an equivalent formulation of $\mathcal{K}^{\mathrm{rat}}$ is to reformulate the notion of exceptionality of a formula (see Definition 1.16). The following theorem gives us a simple test:

2.21. Theorem (Testing Exceptionality).
Let $\mathcal{K}$ be a knowledge base. A formula α is exceptional for $\mathcal{K}$ if and only if $\widetilde{\mathcal{K}} \models \neg\alpha$.

A nice feature is that $\mathcal{K}^{\text{rat}}$ can also be used to compute $\mathcal{K}^{\text{pref}}$ quite efficiently, because the following relation holds [Dix and Schmitt 1993]:

$$\alpha{\mid\!\sim}\beta \ \in \mathcal{K}^{\text{pref}} \ \ \underline{\text{iff}} \ \ \alpha{\mid\!\sim}\mathbf{f} \ \in (\mathcal{K} \cup \{\alpha{\mid\!\sim}\neg\beta\})^{\text{rat}}$$

The above algorithm has been implemented using the *Protein Theorem Prover* developed and used in Koblenz. In fact, there are many more options available than mentioned above. For example, how should *hard knowledge*, i.e. knowledge which is certain, and not subject to any default information, be handled? We refer to [Dix and Schmitt 1993] where this is investigated in detail and where many examples are compared against. Our implementations handles all these options both for $\mathcal{K}^{\text{rat}}$ and for $\mathcal{K}^{\text{pref}}$. We refer the interested reader to `http://www.uni-koblenz.de/ag-ki/Systems/NM/`.

3. From Automated Reasoning to Disjunctive Logic Programming

In the previous sections we introduced various nonmonotonic logics and general methods for computing therein. We now focus on nonmonotonicity in logic programming. Instead of jumping directly into the issue of Horn clause-based logic programming, we will stay for this section within the general framework of first-order predicate logic, without restricting it to Horn clauses. The motivation for this is twofold:

Firstly, recent research in tableau-based deduction methods (for a comprehensive treatment see [Bibel and Schmitt 1998]) provides us with the insight, that SLD-resolution, as well as some of its nonmonotonic extensions are in fact specialised versions of proof procedures from automated deduction in full first-order predicate logic.

Secondly, there is a entire discipline, namely disjunctive logic programming which tries to directly use these unrestricted calculi for logic programming and database reasoning (the only textbook available is [Lobo, Minker and Rajasekar 1992]).

One of the important proof procedures for first-order clausal logic is model elimination which was introduced by Don Loveland already in [Loveland 1968] and in [Loveland 1969]. It was only recently that the presentation of the calculus as a tableaux calculus with special search pruning techniques became obvious. This generalised view of the calculus offers the opportunity of building in several refinements and defining variants thereof.

Nowadays there are two paradigms used as the basis of efficient general purpose theorem provers: *resolution* and *model elimination*. Indeed, the CADE System Competition (CASC) which is held every year, is won more or less alternating by a resolution and by a model elimination based theorem prover.

Disjunctive logic programming offers a programming application for those calculi. [Lobo et al. 1992] is a textbook dealing exclusively with the topic of disjunctive logic programming and its nonmonotonic extensions. The underlying language of this approach to programming and data bases is full first-order logic in clause normal form. In order to stress the procedural character of clauses they are written in this context as $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ instead of $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$. As an operational semantics of programs given by such rules, the authors of [Lobo et al. 1992] use SLI resolution, a calculus which was developed earlier in [Minker and Zanon 1982]. This tree like depicted calculus is in fact very similar to the model elimination calculus given by [Loveland 1968] and in particular to the modern presentations given in [Bibel and Schmitt 1998]. We will use below a variant of model elimination as an operational semantics for disjunctive logic programming, which is supporting the intended procedural reading of clauses. More details on the relation to SLI resolution are given in Section 3.4. The above mentioned calculi offer goal oriented procedures for the interpretation of logic programs.

We will start from theorem proving methods for first-order predicate logic and we demonstrate how to modify and extend these approaches in order to arrive at automated reasoning systems with nonmonotonic extensions. Note that we are discussing disjunctive logic programs *with variables*, while in the literature the nonmonotonic aspects are often discussed with programs which are ground, hence without variables. However, the use of nonmonotonic negation will be rather limited in this section.

In a first step we present as a goal oriented calculus the model elimination and we show how to modify it, such that it can be used as an *interpreter* for positive disjunctive logic programming. We then extend this interpreter by *negation as failure-to-explain*. These variants of model elimination are studied in detail in [Baumgartner and Furbach 1994*a*, Baumgartner, Furbach and Stolzenburg 1997*a*, Baumgartner, Furbach and Stolzenburg 1997*b*, Hähnle and Pape 1997].

In a second approach, we introduce a bottom up proof procedure, the *hyper tableau calculus*, and we discuss how minimal model reasoning can be performed in this framework.

Hyper tableaux are introduced in [Baumgartner, Furbach and Niemelä 1996] as an extension and formalisation of the SATCHMO proof procedure. In the ground case hyper tableaux are similar to the model trees from [Fernandez and Minker 91]. In [Kühn 1997, Baumgartner 1998] this calculus is even more extended towards the use of resolution-like handling of variables.

There is a similar approach in [Rajasekar 1989]: SLO-resolution. This is a goal oriented calculus for positive disjunctive programs which is presented as an extension of SLD-resolution. If all literal signs from the program clauses and from the goal clause are complemented (which preserves satisfiability) our hyper tableaux calculus corresponds to SLO-resolution. It is exactly the case for ground derivations and for the non-ground case for positive disjunctive programs. The DISLOG implementation of the SLO-resolution was discussed in [Rajasekar and Yusuf 1995] along with completeness results. A detailed investigation of this topic can be found in [Baumgartner and Furbach 1997], where such bottom up calculi are shown to be a kind of fixpoint semantics for disjunctive logic programming and a theorem which relates this T_p-operator to the goal oriented restart model elimination is given.

Since the calculi used for disjunctive logic programming have their origin directly in the field of theorem proving and automated deduction, we have to put some additional effort into some data-structure-like formalities.

Literal Trees, Tableau and Branch Sets

We first state some basic definitions, which are used in this section. A *clause* is a multiset of literals, usually written as a disjunction $L_1 \vee \ldots \vee L_n$. Clauses can be alternatively represented with an arrow. $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ is a representation of the clause $A_1 \vee \cdots \vee A_m \vee \neg B_1 \vee \cdots \vee \neg B_n$, where the As and Bs are atoms. Clauses with $m \geq 1$ are called *program clauses* with *head literals* A_i and *body literals* B_i, if present. Clauses of the form $\leftarrow B_1 \wedge \cdots \wedge B_n$ are called *negative* clauses in the sequel. We assume a clause set to be transformed into Goal normal form by changing negative clauses $\leftarrow B_1 \wedge \cdots \wedge B_n$ into Goal $\leftarrow B_1 \wedge \cdots \wedge B_n$ and by adding the additional negative clause $\neg$Goal.

We consider *literal trees*, i.e. finite, ordered trees, all nodes of which, except the root, are labeled with a literal. The labeling function is denoted by λ. Such a literal tree is also called a *tableau* and it is represented as a set of branches, where a *branch (of length n)* is a sequence $[N_0 \cdot N_1 \cdot \ldots \cdot N_n]$ ($n \geq 0$, written as indicated) of nodes such that N_0 is the root of the tree, N_i is the immediate predecessor of N_{i+1} for $0 \leq i < n$, and N_n is a leaf; the functions First and Leaf return the first *labeled*, resp. last node of a branch, i.e. First$([N_0 \cdot N_1 \cdot \ldots \cdot N_n]) = N_1$ and Leaf$([N_0 \cdot N_1 \cdot \ldots \cdot N_n]) = N_n$.

Throughout this section, the letter N is used for nodes, L, K denote literals, and the symbols $\mathbf{p}$, $\mathbf{q}$ are used for branches; like branches, branch-valued variables are also written with brackets, as in $[\mathbf{p}]$. Branch sets are typically denoted by the letters $\mathcal{P}, \mathcal{Q}, \cdots$. We write $\mathcal{P}, \mathcal{Q}$ and mean $\mathcal{P} \cup \mathcal{Q}$ (multiset union is intended here). Similarly, $[\mathbf{p}], \mathcal{Q}$ means $\{[\mathbf{p}]\}, \mathcal{Q}$. We write $N \in [\mathbf{p}]$ *if and only if* N occurs in $[\mathbf{p}]$. A substitution σ is applied to a branch set $\mathcal{P}$, written as $\mathcal{P}\sigma$, by applying σ to all labels of all nodes in $\mathcal{P}$. We say that branch set $\mathcal{P}$ is *more general* than branch set $\mathcal{P}'$ *if and only if* $\mathcal{P}\delta = \mathcal{P}'$ for some substitution δ.

Now let $[\mathbf{p}]$ be a branch $[N_0 \cdot N_1 \cdot \ldots \cdot N_n]$. Any contiguous subsequence of $[\mathbf{p}]$ (possibly $[\mathbf{p}]$ itself) is called a *partial branch* (through $[\mathbf{p}]$). The concatenation of partial branches $[\mathbf{p}]$ and $[\mathbf{q}]$ is denoted by $[\mathbf{p} \cdot \mathbf{q}]$; similarly, $[\mathbf{p} \cdot N]$ means the extension of $[\mathbf{p}]$ by the node N. We find it convenient to confuse a node with its label and write, for instance $[\mathbf{p} \cdot L]$, where L is a literal, instead of "$[\mathbf{p} \cdot N]$, where N is labeled with L"; the meaning of $L \in [\mathbf{p}]$ is obtained in the same way; also, we say "node L" instead of the "node labeled with L".

In order to memorize the fact that a branch contains a contradiction, we allow to label a branch with a "$\star$" as *closed*; we insist that if a branch is labeled as *closed* then its leaf is complementary to some of its ancestor nodes. Branches which are not labeled as closed are said to be *open*. A literal tree is *closed* if each of its branches is closed, otherwise it is *open*.

Equality on branch sets is defined wrt. the labels and the "closed" status. More precisely, suppose given branches $[\mathbf{p}]$ and $[\mathbf{p}']$ stemming from (not necessarily different) branch sets $\mathcal{P}$ and $\mathcal{P}'$ with respective labeling functions λ and λ'; define $\lambda[N_0 \cdot N_1 \cdot \ldots \cdot N_n] = \langle \lambda(N_1), \cdots, \lambda(N_n) \rangle$, $[\mathbf{p}]\star =_{\lambda,\lambda'} [\mathbf{p}']\star$ *if and only if* $[\mathbf{p}] =_{\lambda,\lambda'} [\mathbf{p}']$, where $[\mathbf{p}] =_{\lambda,\lambda'} [\mathbf{p}']$ *if and only if* $\lambda[\mathbf{p}] = \lambda'[\mathbf{p}']$. Equality for branch sets, i.e. $\mathcal{P} = \mathcal{P}'$, is defined as the usual multiset extension of "$=_{\lambda,\lambda'}$".

3.1. Restart Model Elimination

By the previous definitions literal trees are introduced as static objects. We wish to construct such literal trees in a systematic way. This is accomplished by, for instance, the *restart model elimination calculus*.

3.1. DEFINITION *(Branch Extension, Connection).*
The *extension of a branch* $[\mathbf{p}]$ *with clause* $\mathcal{C}$, written as $[\mathbf{p}] \circ \mathcal{C}$, is the branch set $\{[\mathbf{p} \cdot L] \mid L \in \mathcal{C}\}$. Equivalently, when viewed as a tree, this operation extends the branch $[\mathbf{p}]$ by $|\mathcal{C}|$ new nodes which are labeled with the literals from $\mathcal{C}$. A pair of literals (K, L) is a *connection with mgU* σ *if and only if* σ is a most general unifier for K and $\overline{L}$ (here $\overline{L}$ denotes the complement of L).

For the purpose of this section, we introduce restart model elimination in its strongest variant. For this we need the additional notion of a *head selection function*.

3.2. DEFINITION *(Head Selection Function).*
A *head selection function* ν is a function that maps a clause $A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m$ with $n \geq 1$ to an atom $L \in \{A_1, \cdots, A_n\}$. L is called the *selected literal* of that clause by ν. The head selection function f is required to be *stable under lifting* which means that if ν selects $L\gamma$ in the instance of the clause $(A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m)\gamma$ (for some substitution γ) then ν selects L in $A_1 \vee \cdots \vee A_n \leftarrow B_1 \wedge \cdots \wedge B_m$.

A head selection function will be used to distinguish one single head literal to be used as the only entry point during the whole derivation.

3.3. DEFINITION *(Restart Model Elimination (RME)).*
Given a clause set S in Goal normal form and a head selection function. The inference rules *extension step*, *reduction step* and *restart step* on branch sets are defined as in Figure 4. The branch $[\mathbf{p}]$ is called *selected branch* in all three inference rules. A restart step followed immediately by an extension step is also called a *restart extension step*.

Note that like in the usual tableau model elimination calculus (cf. [Letz, Mayr and Goller 1994]), an extension or a reduction step can be applicable to a negative leaf. To positive leaves, only restart step can be applied.

We need one more definition before turning towards derivations:

Restart Model Elimination (RME)

Extension Step:

$$\frac{[\mathbf{p}],\,\mathcal{P} \qquad A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n}{([\mathbf{p} \cdot A_i]\star, [\mathbf{p}] \circ (A_1 \vee \cdots \vee A_{i-1} \vee A_{i+1} \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n), \mathcal{P})\sigma}$$

if

1. $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ (with $m \geq 1$, $n \geq 0$ and $i \in \{1, \cdots, m\}$) is a new variant (called *extending clause*) of a clause in S, and
2. A_i is the selected literal, and
3. $(\mathrm{Leaf}([\mathbf{p}]), A_i)$ is a connection with mgU σ. In this context A_i is called the *extension literal*.

Reduction Step:

$$\frac{[\mathbf{p}],\,\mathcal{P}}{([\mathbf{p}]\star,\,\mathcal{P})\sigma}$$

if

1. $\mathrm{Leaf}([\mathbf{p}])$ is a positive literal, and
2. $(L, \mathrm{Leaf}([\mathbf{p}]))$ is a connection with mgU σ, for some node $L \in [\mathbf{p}]$.

Restart Step:

$$\frac{[\mathbf{p}],\,\mathcal{P}}{[\mathbf{p}] \circ \mathrm{First}([\mathbf{p}]),\,\mathcal{P}}$$

if $\mathrm{Leaf}([\mathbf{p}])$ is a positive literal.

Figure 4: Inference rules for RME.

3.4. DEFINITION *(Computation Rule).*
A *computation rule* is a total function κ which maps a tableau to one of its open branches. It is required that a computation rule is *stable under lifting*, which means that for any substitution σ, whenever $\kappa(\mathcal{Q}\sigma) = [\mathbf{q}]\sigma$ then $\kappa(\mathcal{Q}) = [\mathbf{q}]$.

The role of a computation rule is to determine in a derivation the selected branch for the next inference step:

3.5. DEFINITION *(Derivation).*
Let S be a clause set in Goal normal form and κ be a computation rule. A *restart model elimination derivation (RME derivation) of branch set $\mathcal{P}_n$ with substitution $\sigma_1 \cdots \sigma_n$ via κ from S* consists of a sequence $(([\neg\text{Goal}] = \mathcal{P}_0), \mathcal{P}_1, \cdots, \mathcal{P}_n)$ of branch sets, where for $i = 1, \cdots, n$:

1. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of an extension step with an appropriate variant $\mathcal{C}$ of some clause from S and mgU σ_i, or
2. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of a reduction step and mgU σ_i, or
3. $\mathcal{P}_i$ is obtained from $\mathcal{P}_{i-1}$ by means of a restart step.

Any branch set which is derivable by some RME derivation is also called a *RME tableau*.

In each case the selected branch of the inference is determined by κ. Quite often we will omit the term "via κ" and mean that κ is some arbitrary, given computation rule.

Finally, a *RME refutation* is an RME derivation such that $\mathcal{P}_n$ is closed. The term "RME" is dropped if context allows.

Notice that due to the construction of the inference rules, $\mathcal{P}_1$ is obtained from $\mathcal{P}_0$ by an extension step with some clause Goal $\leftarrow B_1 \wedge \cdots \wedge B_n \in S$ and with empty substitution. This clause is called the *goal clause* of the derivation.

It is worth noting that in extension steps we can connect only with the head literals of input clauses. Since in general this restriction is too strong, because it destroys completeness, we have to "restart" the computation with a fresh copy of a negative clause. This is achieved by the restart rule, because refutations of clause sets in Goal normal form always start with $\text{First}([\mathbf{p}]) \equiv \neg\text{Goal}$, and thus only extension steps are possible to $\neg$Goal, which in turn introduce a new copy of a negative clause.

Figure 5 depicts the above introduced concepts with the following clause set $\{\neg p \vee \neg q, p \vee \neg q, \neg p \vee q, p \vee q\}$; the same clause set in rule notation is $\{\leftarrow p \wedge q, p \leftarrow q, q \leftarrow p, p \vee q \leftarrow\}$ The left hand side of the figure shows a model elimination derivation, where no restriction with respect to the use of extension literals or reduction steps is imposed. The inference rules for this calculus are not introduced formally in this paper, they simply consist of an extension rule as defined before, but without a selection rule and a reduction rule (indicated by dashed arrows in the tree). Note that all leaves in the tree are marked with a "$\star$" as *closed* and hence the tree corresponds to a refutation.

The right hand side of the figure shows a restart model elimination derivation as it is defined above. For this the clause set is transformed into the corresponding

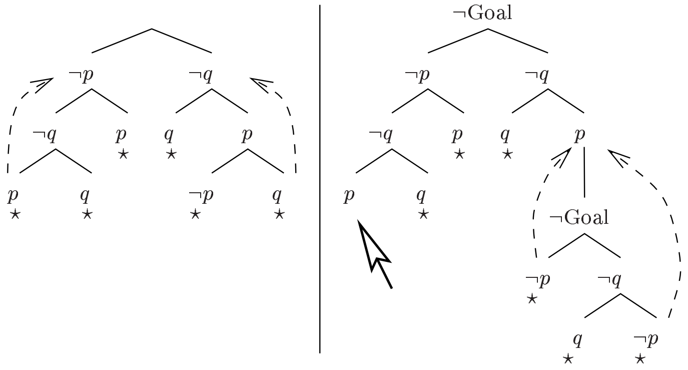


Figure 5: Model Elimination (left side) vs. Restart Model Elimination (right side).

goal normal form $\{\leftarrow \text{Goal}, \text{Goal} \leftarrow p \wedge q, p \leftarrow q, q \leftarrow p, p \vee q \leftarrow\}$. Note the open positive leaf, which is indicated by an arrow. In the present version of restart model elimination it is no more allowed to close the branch by application of a reduction step. This branch can only be extended by a further restart step as the rightmost branch of this tree. This would finally lead to a closed tree, hence a refutation.

3.6. Theorem (Correctness and Completeness).
Restart model elimination is correct and refutational complete

Correctness of the calculus can be derived very simply by writing the restart step as the introduction of an *atomic cut*, i.e. an extension of a branch with the tautology $\neg\text{Goal} \vee \text{Goal}$. In a subsequent step the branch containing Goal can be closed by a reduction step with the root node $\neg$Goal and as a result we get the open branch with Goal, which would be the result of a restart step as well. A proof of the theorem can be found in [Baumgartner and Furbach 1994*a*]. A completeness result with respect to computed answers is given in [Baumgartner, Furbach and Stolzenburg 1997*a*].

3.2. Procedural Interpretation

We are claiming that RME, as it is introduced in the previous subsection can be used in a very natural way as an interpreter for positive disjunctive logic programs.

For the ease of notation we consider the non-Horn clause $A_1 \vee A_2 \leftarrow B$, for which we discuss a procedural interpretation:

- From the definition of the extension step we know that only one head literal, namely the one selected by the head selection function, can be used as the extension literal. Let this be A_1 in our example; i.e. there is exactly one *entry point* for the procedure $A_1 \vee A_2 \leftarrow B$, namely A_1. The clause never can be used for an extension step via A_2.

- Once the clause is used to extend a branch of the tableau, we can interpret the extension of the branch, as establishing the new subgoals from the body, in this case it is the *new goal* B.
- In addition, a new proof of the original goal has to be established with the original program, which, however, is augmented with a new fact $A_2 \leftarrow$. This additional proof is started via a subsequent restart step, which is the only applicable inference rule at the positive leaf A_2 of the tableau. Instead of adding the fact A_2 to the clause set for this additional proof, restart model elimination allows the closing of branch by reduction steps to the branch literal A_2 which obviously has the same effect.

It should be obvious, that restart model elimination is an extension of SLD-resolution[5]. When restricted to definite clauses, no restart step can occur and hence all inner nodes of a branch are negative. Since reduction steps can only occur at an open branch with a negative leaf, it is impossible to find a connection from such a literal and consequently reduction steps can never be applied.

This close relation to SLD-resolution can be used very elegantly for efficient implementation by using the PTTP-technique (Prolog Technology Theorem Proving) as introduced in [Stickel 1988]. This is a compilation technique which takes advantage of efficient commercial Prolog systems. The given set of non-Horn clauses is transformed in a preprocessing step into a Prolog-program, which contains code for the reduction steps and in the case of restart model elimination for the restart steps. The extension steps are performed by the SLD-resolution from the Prolog-system. The PROTEIN-system is a high performance prover which is based on this technique [Baumgartner and Furbach 1994*b*].

3.3. Negation as Failure-to-Explain

Having established a close relationship between SLD-resolution for definite programs and restart model elimination for positive disjunctive programs, we introduce a negation-as-failure rule, which has a close similarity to the negation-as-failure from Prolog.

[Aravindan 1996] introduces an abductive framework for positive disjunctive logic programs that captures minimal model reasoning. From now on we deal with disjunctive logic programs, which are clause sets consisting only of non-negative clauses.

Given a program P and an atom A, an abductive explanation $\mathcal{E}$ for A consists only of negative literals, s.t. $P \cup \mathcal{E} \models A$ and $P \cup \mathcal{E}$ is consistent. A set $\mathcal{E}$ of negative literals is referred to as a potential candidate for A if $P \cup \mathcal{E} \models A$.

3.7. DEFINITION *(Negation by failure-to-explain).*
Let P be a disjunctive logic program and A a be an atom. Then, the *negation by*

[5] This becomes very obvious, if one takes into account that SL-resolution [Kowalski and Kuehner 1971] is a variant of model elimination. Since this calculus is commonly accepted as an extension of SLD-resolution, the same holds for model elimination.

failure-to-explain inference rule is given as:

$$\frac{\text{there is no abductive explanation for } A \text{ wrt } P}{\neg A}$$

It is now natural to ask whether this negation by failure to explain is different from known and accepted semantics such as (E)GCWA.

For this the following theorem from [Aravindan 1996] is important. Note that this is very closely related to circumscription, where every predicate is minimized. We use the minimal entailment relation $\models_{min}$, as defined in Definition 1.6.

3.8. THEOREM (Abductive Explanation).
Let P be a disjunctive logic program and A an atom. Then, there exists no abductive explanation for A wrt P if and only if $P \models_{min} \neg A$. *Also, the empty set $\emptyset$ is an abductive explanation for A wrt P* if and only if $P \models_{min} A$.

In other words, this theorem states, that negation-as-failure to explain captures exactly the minimal model reasoning as defined by the extended generalized world assumption [Lobo et al. 1992]. The negation by failure-to-explain can be seen as well as a special case of circumscription, by minimizing every atom (see section 1.3).

Let us investigate this rule with an example, where the program P consists of the following clauses:

$$\begin{aligned} P: \quad & p \vee s \leftarrow t \\ & p \vee r \leftarrow q \\ & t \vee q \leftarrow \end{aligned}$$

Given the atom p, the task is to generate potential candidates and to check if they are consistent with the given program. These candidates can be generated by obtaining the logical consequences K of $P \cup \{\neg p\}$, where K is a simple disjunction of ground atoms. Note that $\mathcal{E} = \{\neg A \mid A \in K\}$ is a potential candidate for p.

The key observation for the generation of potential candidates with the RME-calculus, is that those tableaux, which contain only open paths with positive literals represent such K. The literals from K are exactly the leaves from the open paths. If we start in our example with $\{\neg p\}$, we get with one extension the two paths $\{\neg t\}$ and $\{s\}$. A further extension step yields $\{q\}$ and $\{s\}$, and hence we have one potential candidate $K = \neg q \vee \neg s$.

At this point it is obvious that we have to weaken our calculus: If the head selection function would force us to use the first clause for an extension step only via the literal s, we can not derive this logical consequence. Hence for the purpose of negation as failure-to-explain, we have to use RME without head selection function. It is obvious, that this calculus is sound and complete as well.

In order to get a precise description, how RME computes negation as failure to explain, some further concepts are in order. First observe that one needs to generate only the minimal potential candidates: A potential candidate $\mathcal{E}$ for A wrt

P is *minimal* iff no proper subset of it is a potential candidate for A wrt P. The set of all minimal potential candidates for A wrt P is denoted by $\mathrm{MPOT}(A, P)$. In the same way we get the notion of an *minimal abductive explanation* and the set $\mathrm{ME}(A, P)$ and the following property.

3.9. PROPOSITION. *Let P be a disjunctive logic program and A an atom. Then, $\mathrm{ME}(A, P) \subseteq \mathrm{MPOT}(A, P)$.*

Thus generating MPOT and testing every member of it for consistency with the given program ensures that all minimal abductive explanations can be generated. Further, if all members of MPOT fail the consistency check then negation of the given sentence can be inferred according to the negation by failure to explain inference rule.

Generation of all minimal potential candidates may be costly and not necessary. It is possible to weaken this set, and for this we need the following notions.

3.10. DEFINITION *(Relevant Extension, $\mathcal{E}$ covers $\mathcal{E}'$).*
Let $\mathcal{E}$ and $\mathcal{E}'$ two potential candidates (not necessarily minimal) for A wrt P. Then $\mathcal{E}'$ is said to be a *relevant extension* of $\mathcal{E}$ iff $\mathcal{E} \subseteq \mathcal{E}'$ and for every literal $L \in \mathcal{E}' \backslash \mathcal{E}$, we have that $P \cup \mathcal{E} \models L$. $\mathcal{E}$ is said to *cover* $\mathcal{E}'$ if $\mathcal{E} \subseteq \mathcal{E}''$, where $\mathcal{E}''$ is a relevant extension of $\mathcal{E}'$.

Let us consider an example to clarify these notions: a program P with two clauses $p \vee q \vee r$ and $s \leftarrow r$. $\mathcal{E}_1 = \{\neg q, \neg r\}$ is a minimal potential candidate for p. It covers the other minimal potential candidate $\mathcal{E}_2 = \{\neg q, \neg s\}$ because it is contained in $\{\neg s, \neg q, \neg r\}$ which is a relevant extension of $\mathcal{E}_2$.

The essence of this discussion is that, given P and A, instead of generating $\mathrm{MPOT}(A, P)$ it is enough to generate a set of potential candidates so that every member of $\mathrm{MPOT}(A, P)$ is covered. We may not generate all minimal abductive explanations by this method, but we are assured that all of them are covered. Failing to find an abductive explanation from this set will ensure that $\neg A$ can be inferred.

Before we now turn towards a decision procedure for minimal model reasoning wrt negation as failure-to-explain, we introduce some notions for RME derivations.

In order to compute potential candidates we need the concept of a *saturated branch set* $\mathcal{P}$, which is a set of branches, i.e. a literal tree, where every leaf is a positive literal and the only inference rule, which can be applied is the restart rule. Suppose there exists a branch $[\mathbf{p}] \in \mathcal{P}$ s.t. $\mathrm{Leaf}([\mathbf{p}])$ is not ground, then $\mathcal{P}$ is said to have *floundered*.

Another important concept is similar to loop checking in programming languages: it is called *blockwise regularity*. A branch in an RME derivation is said to be *blockwise regular* if every pair of identical negative leafs are separated by at least one restart and all positive literals in the branch are pairwise distinct. A literal tree is blockwise regular *if and only if* all branches in it are blockwise regular.

The following two theorems relate RME to abductive explanations. The first one states that every saturated branch set in an RME derivation constitutes a potential candidate; and the second one ensures that all minimal potential candidates are covered.

3.11. THEOREM (Computing Abductive Explanations).
Let P be a disjunctive logic program and A be a ground atom. Let $(\mathcal{P}_0, \cdots, \mathcal{P}_k, \cdots)$ be an RME derivation for $P \cup \{\neg A\}$, where $\mathcal{P}_k$ is saturated and non-floundered. Then, $\mathcal{E} = \{\neg \mathrm{Leaf}(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}_k\}$ is a potential candidate for explaining G. Further, if $P \cup \mathcal{E}$ is consistent, then $\mathcal{E}$ is an abductive explanation for A wrt P.

3.12. THEOREM (Covering Potential Candidates).
Let P be a disjunctive logic program and A an atom. Let $\mathcal{E}$ be a minimal potential candidate for explaining A wrt P. Then, there exists a saturated branch set in an RME derivation for A that constitutes a potential candidate $\mathcal{E}'$ that covers $\mathcal{E}$.

In Figure 3.3 the use of RME for computing negation-as-failure-to-explain in our small example from above is taken from [Aravindan 1996].

We start with the branch set $\{[\neg p]\}$. After two extension steps, we obtain a saturated the branch set with leafs q and s. This corresponds to a potential candidate $\{\neg q, \neg s\}$. Now a restart step can be applied to either of these paths, and let us say it is applied to q first. After an extension step and a reduction step, we get another saturated multiset and potential candidate $\{\neg r, \neg s\}$. Note that applying an extension step with the other clause $p \vee s \leftarrow t$ after restart, fails ultimately because of regularity check. By restarting at leaf s and continuing our derivation we arrive at two more candidates corresponding to $q \vee t \vee r$ and $r \vee t \vee r$. By employing some optimization techniques, this can be cut down to a single candidate $\{\neg t, \neg r\}$. As shown in the figure, further attempts to restart and continue the derivation are failed by regularity checks.

Thus, three potential candidates are generated: $\{\neg q, \neg s\}$, $\{\neg r, \neg s\}$, $\{\neg t, \neg r\}$. Consistency checks can be carried at any point, i.e. immediately after generating a candidate or delayed till further candidates are generated. In this case, all these candidates are consistent and correspond to three minimal abductive explanations. Note that p has explanations and $\emptyset$ is not an abductive explanation. Hence, if the query is "check if p is false", then the derivation can stop immediately after generating $\{\neg q, \neg s\}$ to answer negatively.

3.4. Relation to Other Work

In the textbook on disjunctive logic programming [Lobo et al. 1992] SLI resolution is used as a base for all proof theoretic aspects (several of the systems described in that book have been implemented by D. Seipel in his *DisLoG* system, see `<URL:http://www-info1.informatik.uni-wuerzburg.de/database/DisLog>`). SLI resolution was introduced originally in [Minker and Zanon 1982] as LUST resolution, *Linear Resolution with Unrestricted Selection function based on Trees.* This calculus is a variant of model elimination, as it was introduced before in

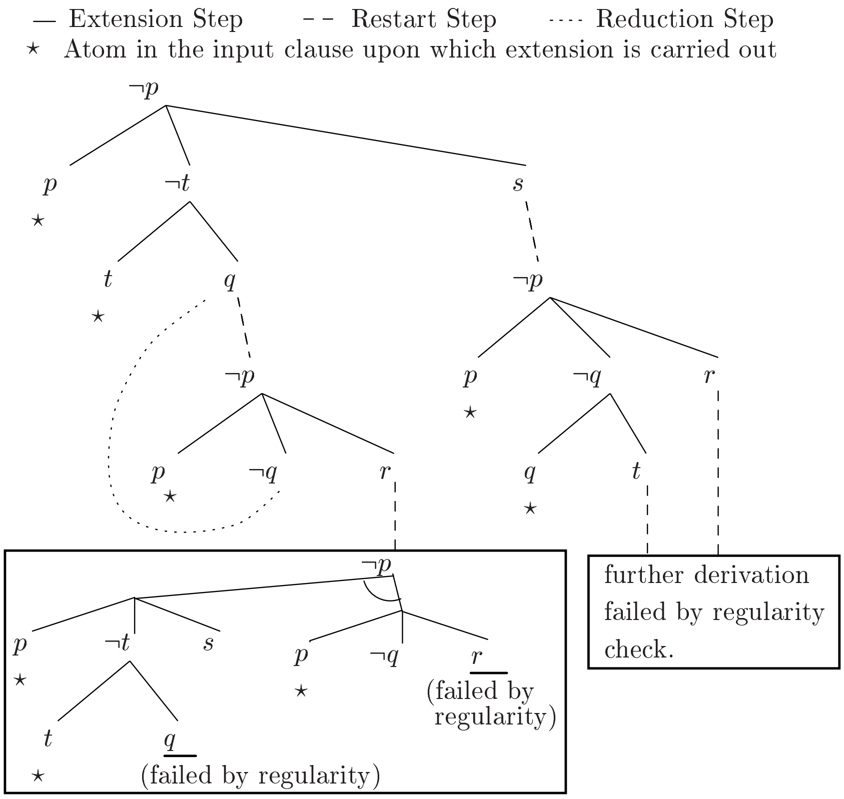


Figure 6: Abductive reasoning with modified restart model elimination calculus.

[Loveland 1968]. In this early reference, model elimination was even introduced in a tree-like manner. In so far SLI resolution is very well comparable with the presented approach via restart model elimination. The main difference, however, is that SLI resolution, like model elimination without the restart rule, needs all contrapositives of a program clause for being complete. As a consequence it does not allow for a procedural interpretation as it is mandatory for logic programming purposes. This disadvantage is removed in restart model elimination; clauses are used as expected in a programming system. Other calculi aiming at this important property are described by the family of near-Horn Prologs [Loveland 1991] or the Problem Reduction Format [Plaisted 1988].

The Failure-to-Explain rule which was introduced in the previous section established minimal model reasoning in an abductive framework. The main idea of this approach was to compute logical consequences to get explanations for a positive atom. A very similar idea is used by [Lobo et al. 1992], where the abductive explanations are given by the "support for negation", as they are called there. For the computation of these consequences SLI resolution is used. Both approaches, the one using abduction, as well as the SLI-based one, are very closely related to MILO resolution [Przymusinski 1989], where this idea was introduced for computing circumscription.

Model elimination can be seen as a natural extension of SLD resolution. This view can be used for efficient implementations of proof procedures based on model elimination by means of the PTTP-technique due to [Stickel 1988]. In a compilation phase a set of disjunctive program clauses is transformed into a Prolog-program, where those parts of the calculus, which are responsible for the non-Horn parts of the clauses are contained in the result of the compilation. The resulting Prolog program can be processed by efficient commercial Prolog systems. In [Baumgartner and Furbach 1994*b*] this technique is applied to implement PROTEIN, a theorem prover based on restart model elimination. In [Schaub and Brüning 1996] model elimination together with the PTTP-technique is used for query answering in default logics. In contrast to the algorithms, which are given in this article before, this approach incorporates default reasoning within an existing theorem prover. For this the compilation of the default rules is done in a way, that admissibility checks for the integration of default rules are handled in the resulting Prolog code. However, this works only in the semi-monotonic case, i.e. for normal default theories and not in the general case.

3.5. Hyper Tableau

In this section we discuss a bottom up calculus for positive disjunctive programs. This calculus can be seen as a model generation procedure and, hence, is very well suited for modification to enable minimal model reasoning.

In [Baumgartner et al. 1996] we introduced hyper tableaux as a variant of clausal normal form tableaux. They keep many desirable features of analytic tableaux (structure of proofs, reading off models in special cases) while taking advantage of central ideas from (positive) hyper resolution. In the ground case, hyper tableaux coincide with the well-known SATCHMO procedure of [Manthey and Bry 1988]; for the first-order case, hyper tableaux have significant advantages (see [Baumgartner et al. 1996]). For the purposes of this section, however, we use hyper tableaux in the ground case only.

As in the previous sections on model elimination, we introduce hyper tableaux, as a special kind of literal trees. In restart model elimination the generation of a tree started with a goal clause by extending always negative literals. In the hyper tableau calculus we go the other way round by starting with (possibly disjunctive) facts, which of course yields a tree which corresponds to partial models of the program. During a derivation the tree is extended such that in any step an open branch is a candidate for a partial model.

We assume a set of clauses S and a computation rule as defined in Section 3.1. In order to define hyper tableau as literal trees, we need the notion of an *initial tree*, which is a one-node literal tree, consisting of a root only, for which there is no label. Given a branch $[\mathbf{p}] = [N_0 \cdot N_1 \cdot \ldots \cdot N_n]$ the *literal set of* $\mathbf{p}$ is given as $lit(\mathbf{p}) = \{\lambda(N_1), \cdots, \lambda(N_n)\}$.

3.13. DEFINITION *(Hyper Tableau).*
Let S be a finite set of ground clauses and κ be a computation rule. *Hyper tableaux* are inductively defined as follows:

Initialisation Step: A tree, consisting of the root node only, is a hyper tableau for S. Its single branch is labeled as "open" (i.e. it is not labeled as "closed").

Hyper Extension Step: If

1. $[\mathbf{p}], \mathcal{P}$ is an open hyper tableau for S, i.e. $\mathbf{p}$ is the open branch selected by κ and
2. $\mathcal{C} = A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ is a clause from S ($m \geq 0$, $n \geq 0$), called *extending clause* in this context, and
3. $\{B_1, \cdots, B_n\} \subseteq \mathbf{p}$ (referred to as *hyper condition*)

then the tree $\{[\mathbf{p} \cdot \neg B_1]\star, \cdots, [\mathbf{p} \cdot \neg B_n]\star\}, [\mathbf{p}] \circ (A_1 \vee \cdots \vee A_m), \mathcal{P}$ is a hyper tableau for S.

We will write the fact that a tree $\mathcal{T}'$ can be obtained from a tree $\mathcal{T}$ by a hyper extension step in the way defined as $\mathcal{T} \vdash_{\mathbf{p},\mathcal{C}} \mathcal{T}'$, and say that $\mathcal{C}$ is *applicable* to $\mathbf{p}$ (or $\mathcal{T}$). Note that the head selection function κ does not appear explicitly in this relation; instead we prefer to let κ be given implicitly by the context.

The hyper condition of an extension expresses that *all* (possibly zero) body literals have to be satisfied by the branch to be extended. This similarity to hyper *resolution* coined the name "hyper tableau".

3.14. DEFINITION *(Hyper Tableau Derivation).*
Let κ be a selection function. A (possible infinite) sequence $\mathcal{T}_1, \cdots, \mathcal{T}_n, \cdots$ of hyper tableaux for S is called a *(hyper tableau) derivation from S if and only if* $\mathcal{T}_1$ is obtained by an initialisation step, and for $i > 1$, $\mathcal{T}_{i-1} \vdash_{\mathbf{p}_{i-1},\mathcal{C}_{i-1}} \mathcal{T}_i$ for some clause $\mathcal{C}_{i-1} \in S$. This is also written as

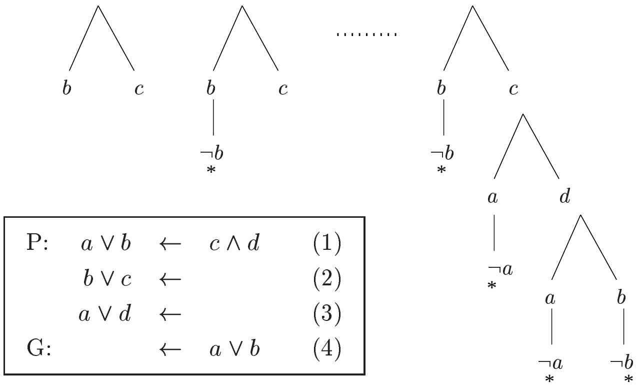
$$\mathcal{T}_1 \vdash_{\mathbf{p}_1,\mathcal{C}_1} \mathcal{T}_2 \cdots \mathcal{T}_n \vdash_{\mathbf{p}_n,\mathcal{C}_n} \mathcal{T}_{n+1} \cdots$$

A derivation is called a *(hyper tableau) refutation* if it contains a closed tableau.

Note that extension steps are no longer applicable to a closed hyper tableau.

Figure 7 shows an example refutation. This example also demonstrates that hyper tableaux handle more than one negative clause. By this it is possible to have integrity constraints in the input clause set, and not just program clauses.

The ground version of the calculus as it is presented above resembles very much the proof procedure SATCHMO [Manthey and Bry 1988]. The reader is invited to study the first-order version of the hyper tableau calculus in [Baumgartner et al. 1996]. The main difference to SATCHMO only then becomes apparent: In principle SATCHMO works for ground clauses. This is necessary in order to avoid checking of substitutions for "splitted variables". Consider a clause $p(x) \vee q(x) \leftarrow b(a)$;



P:	$a \vee b$	$\leftarrow$	$c \wedge d$	(1)
	$b \vee c$	$\leftarrow$		(2)
	$a \vee d$	$\leftarrow$		(3)
G:		$\leftarrow$	$a \vee b$	(4)

Figure 7: A hyper tableau for P and G given in the figure. Notice that the formula G stands for the two clauses $\leftarrow a$ and $\leftarrow b$. For simplicity of presentation, negative leaf nodes stemming from program clauses are not drawn.

whenever this clause is used for an hyper extension step, the information that $p(x)$ in one branch and the $q(x)$ in the other branch stem from the same quantification $\forall x(p(x) \vee q(x) \leftarrow b(a))$ has to be memorized. This can be avoided by generating ground instances of clauses during the derivation, such that this splitting of the variable x cannot occur, because is instantiated by a ground term. SATCHMO achieves this by ground instantiating all variable occurrences dynamically during the proof. Hyper tableau maintains variables, only in those case where a splitting of variables can occur a ground instantiation of the corresponding occurrences is done. (In [Kühn 1997, Baumgartner 1998] this is even more liberalized.)

The central property of an open branch b is that it can be mapped to an interpretation in the usual way, i.e. by taking the positive literals of b as *true* and all others as *false*; for infinite derivations we take the chain limit of the increasing branches. Together with an appropriate *fairness* notion for derivations (roughly: at least one open branch has to be expanded as long as possible without violating regularity) a completeness result for the hyper tableau calculus can be proven (see again [Baumgartner et al. 1996]).

3.6. Minimal Model Reasoning

In this section we will introduce two methods for minimal model reasoning, which are closely related to model generation with hyper tableaux.

Tableaux with Groundedness Test

[Niemelä 1996*b*] presents a calculus for propositional minimal model reasoning. The basic idea is to generate models with a hyper tableau proof procedure and to include an additional test for ruling out those branches in the tableau, which do not represent minimal models. This groundedness test is done *locally*, i.e. there is no need to compare a branch with other branches computed previously; hence there is no need to store models. The following definition of tableaux differs in two aspects from the hyper tableaux definition above. Firstly, the initialisation step introduces a kind of goal orientedness by introducing the atom, whose negation is to be proven and secondly, the extension rule performs the branching by the use of a cut rule.

3.15. DEFINITION *(Constructing Tableaux).*
Let S be a finite set of ground clauses, together with the problem of determining whether $S \vdash_{min} \neg A$ holds; and κ be a computation rule. Tableaux for $S \cup \{A\}$ are inductively defined as follows:

Initialisation Step: The tree, consisting of the root node labeled with A, is a tableau for $S \cup \{A\}$.

Extension Step 1: If
1. $[\mathbf{p}], \mathcal{P}$ is an open tableau for $S \cup \{A\}$, i.e. $\mathbf{p}$ is the open branch selected by κ and
2. $\mathcal{C} = A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ is a clause from S ($m \geq 0$, $n \geq 0$), and
3. $\{B_1, \cdots, B_n\} \subseteq \mathbf{p}$, and
4. $\{\neg A_1, \cdots, \neg A_{j-1}, \neg A_{j+1}, \cdots, \neg A_m\} \subseteq \mathbf{p}$

then the tree $([\mathbf{p}] \circ A_j, \mathcal{P})$ is a tableau for $S \cup \{A\}$.

Extension Step 2(Cut):
If
1. $[\mathbf{p}], \mathcal{P}$ is an open tableau for $S \cup \{A\}$, i.e. $\mathbf{p}$ is the open branch selected by κ and
2. $\mathcal{C} = A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ is a clause from S ($m \geq 0$, $n \geq 0$), and
3. $\{B_1, \cdots, B_n\} \subseteq \mathbf{p}$,

then the tree $(\{[\mathbf{p}] \circ A_j, [\mathbf{p}] \circ \neg A_j\}, \mathcal{P})$ is a tableau for $S \cup \{A\}$.

Note that in this definition there is no explicit labeling of branches with "open" or "closed", as it was in the definition of hyper tableaux. Here we assume a branch to be closed, whenever it contains two complementary literals.

In [Niemelä 1996*b*] this calculus is also given for deciding unsatisfiability of a clause set S and correctness and completeness is proven to this end as well. We introduced the calculus only for minimal model reasoning, i.e. together with a problem $S \vdash_{min} \neg A$.

Beside the closing due to complementary literals within a branch this calculus introduces a groundedness test.

3.16. DEFINITION *(Groundedness).*
A branch $[\mathbf{p}]$ is said to be ungrounded in S, when for some atom A in $[\mathbf{p}]$, $S \cup N_S([\mathbf{p}]) \models A$ does not hold, where

$$N_S([\mathbf{p}]) = \{\neg A \mid A \text{ occurs in the head of a } C \in S, \text{ but is not in } [\mathbf{p}]\}.$$

A branch is MM-closed w.r.t. S if it is closed or ungrounded in S.

Note that for checking $S \vdash_{min} \neg A$ we build the tableau for $S \cup \{A\}$, but for groundedness test, only S is used.

Consider as an example the set of clauses $S = \{a \vee b \vee c \leftarrow, a \vee c \leftarrow b\}$ and the problem of determining whether $S \vdash_{min} \neg b$ holds. We start with the initial tableau $[b]$. One extension with the cut-rule yields $\{[b \cdot \neg a], [b \cdot a]\}$. Further extension of the first branch with extension rule 1 gives $\{[b \cdot \neg a \cdot c], [b \cdot a]\}$. The first branch is ungrounded, because $N_S([b \cdot \neg a \cdot c]) = \{\neg a\}$ and b appears on the branch but $S \cup N_S([b \cdot \neg a \cdot c]) \models b$ does not hold. The second branch is ungrounded, because $N_S([b \cdot a]) = \{\neg c\}$ and b appears on the branch but $S \cup N_S([b \cdot a] \models b$ does not hold. Consequently, all branches are MM-closed w.r.t. S . Correctness of the calculus tells us that $S \vdash_{min} \neg b$ holds.

As in the case of hyper tableau, a fairness condition is needed in order to get a decision procedure (for the propositional case). This condition is called *finished* in [Niemelä 1996*b*] and it is the regularity condition introduced before. Together with this condition soundness and completeness of this calculus w.r.t. minimal model reasoning is proven. In [Niemelä 1996*a*] this approach is generalized for first-order circumscriptive reasoning. The aim is to handle parallel circumscription with fixed and varying predicates with respect to Herbrand models. This approach is proven to be correct and in case of absence of function symbols to be complete.

It is important to note, that in this approach the closing condition for branches is more complex than in the classical case; it involves not only a test on complementary literals, moreover it involves a test for classical consequence. This test can be realized by an additional call to a propositional theorem prover, which can be performed at any stage during the extension of a tableau. This seems to be in harmony with the complexity results, which tell us that propositional minimal model reasoning is a complete problem on the second level of the polynomial hierarchy. Hence minimal model reasoning is strictly harder than classical reasoning and it cannot be reduced to classical reasoning using a mapping computable in polynomial time.

MM-SATCHMO

MM-SATCHMO is described in [Bry and Yahya 1996] as an extension of the SATCHMO-proof procedure. As discussed before , the latter is similar to our ground version of hyper tableau. In order to compute minimal models, Bry and Yahya modify the generation of models in a way, that the first generated model is guaranteed to be a minimal one; while generating further models one always have to compare the current one with previously generated models. This comparison rules out the generation of non-minimal models.

In order to generate a first model, which is minimal, the extension step of a branch $\mathbf{p}$ with a clause $A_1 \vee \cdots \vee A_m \leftarrow B_1 \wedge \cdots \wedge B_n$ is modified, such that the leftmost open branch contains A_1 plus the additional literal $\neg A_2, \cdots, \neg A_m$. This is called complemented splitting, which is described elsewhere as factorization or folding up. Thus the first generated model is minimal and all other generation steps have to be constrained by the negation of the previously generated models.

In our small example $S = \{a \vee b \vee c \leftarrow, a \vee c \leftarrow b\}$ we can construct in a first step the $\{[a \cdot \neg b \cdot \neg c], [b], [c]\}$. The first branch cannot be expanded further with the second clause, because the hyper-condition is not fulfilled, hence we get a first minimal model $[a \cdot \neg b \cdot \neg c]$. Extension of the second branch $[b]$ gives us $[b \cdot a \cdot \neg c]$, which has to be identified as non-minimal by comparison with the previously generated model. The other branch $[b \cdot c]$ turns out to give non-minimal models in the next step.

Comparing MM-SATCHMO with the previous method by groundedness test, we (again) can study the tradeoff between time and space complexity. Whereas the groundedness method has to perform an extra call to a theorem prover to test groundedness of branches, the MM-SATCHMO procedure has to store generated models for further comparisons. The previously mentioned goal-orientedness of the groundedness-method is not available in MM-SATCHMO, here all minimal models have to be generated.

4. Nonmonotonic Semantics of Logic Programs

This section is devoted to logic programs. We assume some familiarity with the classical SLD-resolution for definite programs and will therefore just give a quick and compact description of the classical results in Subsection 4.1. A more detailed investigation is contained in [Brewka et al. 1997, Brewka and Dix 2001, Dix 1995*c*]. We also point to the LPNMR-series of conferences organized every other year ([Pereira and Nerode 1993, Marek, Nerode and Truszczyński 1995, Dix, Furbach and Nerode 1997]) and to the NMELP workshop series organized in conjunction with the international Logic Programming conferences ([Dix, Pereira and Przymusinski 1995, Dix, Pereira and Przymusinski 1997, Dix, Pereira and Przymusinski 1998]).

Instead, we will motivate (in Subsection 4.2) and investigate how positive programs can be extended by allowing *negation* in the bodies of rules (Subsection 4.3). This leads to the main competing approaches: the well-founded (WFS) and the stable (STABLE) semantics. In a second step, we will also allow disjunctions in the heads of rules. This leads for positive disjunctive programs to the generalized closed world assumption (GCWA) due to Minker (Subsection 4.4). In Subsection 4.5 we extend GCWA to programs with negation. This leads, like in the case of non-disjunctive programs, to an extension of WFS and one of STABLE.

Most of this section is written in a declarative style: the procedural aspect (already considered for positive programs in Section 3) will be considered in Section 5.

Let us fix some notation. A language $\mathcal{L}$ consists of a set of relation symbols and a set of function symbols (each symbol has an associated arity). Nullary functions are called constants. Terms and atoms are built from $\mathcal{L}$ in the usual way starting with variables, applying function symbols and relation-symbols.

Instead of considering arbitrary $\mathcal{L}$-formulae, our main object of interest is a program and we recall the definition introduced in the previous section:

4.1. DEFINITION *(Definite Logic Program).*
A *definite* logic program consists of a finite number of *rules* of the form

$$A \leftarrow B_1, \ldots, B_m,$$

where $A, B_1, \ldots, B_m$ are positive atoms (containing possibly free variables). We call A the *head* of the rule and $B_1, \ldots, B_m$ its *body*. The comma represents conjunction $\wedge$.

We can think of a program as formalizing our knowledge about the world and how the world behaves. Of course, we also want to derive new information, i.e. we want to ask queries:

4.2. DEFINITION *(Query).*
Given a definite program we usually have a definite query in mind that we want to be solved. A definite query Q is a conjunction of positive atoms $C_1 \wedge \ldots \wedge C_l$ which we denote by

$$\text{?- } C_1, \ldots, C_l.$$

These C_i may also contain variables. Asking a query Q to a program P means asking for all possible substitutions Θ of the variables in Q such that $Q\Theta$ follows from P. Often, Θ is also called an answer to Q. Note that $Q\Theta$ may still contain free variables.

4.1. Classical Results

Note that definite programs already contain some nonmonotonic component, which can be expressed in the following

4.3. PRINCIPLE *(Orientation).*
If a ground atom A does not unify with some head of a program rule of P, then this atom is considered to be false. In this case we say that "*not* A" is derivable from P to distinguish it from classical $\neg A$.

The orientation principle is nothing but a weak form of *negation-by-failure*. Given an intermediate goal *not* A, we first try to prove A. But if A does not unify with any head, A fails and this is the reason to derive *not* A.

SLD-Resolution
SLD-Resolution[6] is a special form of Robinson's general Resolution rule. While Robinson's rule is complete for full first order logic, SLD is complete for definite

[6]**SL**-resolution for **D**efinite clauses. **SL**-resolution stands for **L**inear resolution with **S**election function.

logic programs (see Theorem 4.4). We do not give a complete definition of SLD-Resolution (see [Lloyd 1987]) but remind the reader the overall behaviour of SLD-resolution. We start with a query in the form $\leftarrow Q$. Sometimes the notation $\Box \leftarrow Q$ is also used, where $\Box$ denotes the falsum. In any round we have to select an atom where we have to resolve a clause against. Three different sorts of branches can occur, namely

1. *infinite* branches,
2. branches that *end up with the empty clause*, and
3. branches that *end in a deadlock ("Failure")*: no applicable rule is left.

Definite programs have the nice feature that the intersection of all Herbrand models exists and is again a Herbrand model of P. It is denoted by M_P and called the *least Herbrand model* of P. Note that our original aim was to find substitutions Θ such that $Q\Theta$ is derivable from the program P. This task as well as M_P is closely related to SLD:

4.4. THEOREM (Soundness and Completeness of SLD).
The following properties are equivalent:

- $P \models \forall\, Q\Theta$, *i.e.* $\forall\, Q\Theta$ *is true in all models of* P,
- $M_P \models \forall\, Q\Theta$,
- *SLD computes an answer* τ *that subsumes* Θ *wrt* Q *(i.e.* $\exists\sigma : Q\tau\sigma = Q\Theta$*).*

Note that not any correct answer is computed, only the most general one is (which of course subsumes all the correct ones).

The main feature of SLD-Resolution is its *Goal-Orientedness*. SLD automatically ensures (because it starts with the Query) that we consider only those rules that are relevant for the query to be answered. Rules that are not at all related are simply not considered in the course of the proof.

There is also an extension of SLD which can treat negative subgoals: SLDNF. Whenever a negative subgoal *not* A is reached, the procedure starts a negation-as-failure mode and tries to prove A. When this fails finitely, then the original subgoal *not* A succeeds. If, however, A succeeds, then *not* A fails and we have to backtrack. Complications arise when *not* A is called and it is not ground: the *floundering* problem. We refer to [Dix 1995*c*, Apt and Bol 1994, Apt and Doets 1994] for a more detailed discussion.

4.2. What Do we Want?

We now motivate the need for negative literals occurring in bodies of rules. When representing knowledge, we do not only want to formulate negative queries, we also want to express *default-statements* of the form

Normally, unless something abnormal holds, then ψ implies ϕ.

Such statements were the main motivation for nonmonotonic logics, like Default Logic or Circumscription (see Section 1, in particular Subsection 1.3). How can we formulate such a statement as a logic program? The most natural way is to use negation "*not*"

$$\phi \ \leftarrow \ \psi, \ not\ ab$$

where *ab* stands for *abnormality*. Obviously, this forces us to extend definite programs by negative atoms. Also the

Law of Inertia: Things normally tend to stay the same.

can be nicely captured with negation and abnormality predicates: abnormalities should be the exception rather than the rule.

4.5. Definition *(Normal Logic Program).*
A *normal* logic program consists of a finite number of *rules* of the form

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

where $A, B_1, \ldots, B_m, C_1, \ldots, C_n$ are positive atoms (containing possibly free variables). We call A the *head* of the rule, $B_1, \ldots, B_m$ its *positive body* and $C1, \ldots, C_n$ its *negative body*. The comma represents conjunction $\wedge$.

We now have two problems:

1. to define a suitable semantics for programs with negation,
2. to design efficient query-answering methods.

Is there only one possibility or do there exists various semantics, corresponding to different grades of nonmonotonicity? In fact, research in the last 10 years clarified this important question. Two main semantics evolved, the well-founded (WFS) and the stable semantics (STABLE) and led to various refinements and extensions. For detailed investigations and overview articles we refer the reader to [Apt and Bol 1994, Minker 1993, Minker 1996, Dix 1995*c*, Brewka et al. 1997, Brewka and Dix 2001].

We note that there is one nice feature of the classical bottom-up computation that is essential for our nonmonotonic semantics. In this framework of databases, rules of the form

$$A \leftarrow A$$

do not make any problems! They simply can not be applied or do not produce anything new. Whereas in the Top-Down approach, such rules give rise to infinite branches! Elimination of such rules will turn out to be an interesting property. We therefore formulate it as a principle:

4.6. Principle *(Elimination of Tautologies).*
Suppose a program P has a rule which contains the same atom in its body as well as in its head (i.e. the head consists of exactly this atom). Then we can eliminate this rule without changing the semantics.

When we consider programs with negative literals, we need a slight extension of Principle 4.3:

4.7. PRINCIPLE *(Reduction).*
Suppose we are given a program P with possibly default-atoms in its body. If a ground atom A does not unify with any head of the rules of P, then we can delete in every rule any occurrence of "*not* A" without changing the semantics.

Dually, if there is an instance of a rule of the form "$B \leftarrow$ " then we can delete all rules that contain "*not* B" in their bodies.

So far we have collected two principles, *Elimination of Tautologies* and *Reduction*. Are there more principles? How many? Let us consider a famous example:

4.8. EXAMPLE *(The Transitive Closure).*
Assume we are given a graph consisting of nodes and edges between some of them. We want to know which nodes are reachable from a given one. A natural formalization of the property *reachable* would be

$$reachable(x) \leftarrow edge(x, y), reachable(y).$$

What happens if we are given the following facts

$$edge(a, b),\ edge(b, a),\ edge(c, d)$$

and $reachable(c)$? Of course, we expect that neither a nor b are reachable because there is no path from c to either a or b.

Also classical SLDNF-Resolution does not derive "*not* $reachable(a)$"!

Principle 4.6 does not help, because it simply does not apply. It turns out that we can augment our two principles by a third one, that constitutes together with them a very nice calculus handling the above example in the right way. This principle is related to *Partial Evaluation*, hence its name GPPE[7]. Here is some motivation. The query "*not* $reachable(a)$" leads to "$reachable(a) \leftarrow edge(a, b), reachable(b)$" and "reachable(b)" leads to the rule "$reachable(b) \leftarrow edge(b, a), reachable(a)$". Both rules can be seen as definitions for reachable(a) and reachable(b) respectively. So it should be possible to replace in these rules the body atoms of "reachable" by their definitions. Thus we obtain the two rules

$$reachable(a) \leftarrow edge(a, b), edge(b, a), reachable(a)$$
$$reachable(b) \leftarrow edge(b, a), edge(a, b), reachable(b)$$

that can both be eliminated by applying Principle 4.6. So we end up with a program that does neither contain $reachable(a)$ nor $reachable(b)$ in one of the heads. Therefore, according to Principle 4.3 both atoms should be considered false. The precise formulation of this principle is due to [Brass and Dix 1994] and, independently, to [Sakama and Seki 1994] for disjunctive programs and reads as follows:

[7]Generalized Principle of Partial Evaluation

4.9. Principle *(Generalized Partial Evaluation, GPPE).*
We say that a semantics SEM satisfies GPPE, if the following transformation does not change the semantics. *Replace a rule* $A \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ *where* $\mathcal{B}^+$ *contains a distinguished atom* B *by the rules*

$$A \leftarrow \left(\mathcal{B}^+ \setminus \{B\}\right) \cup \mathcal{B}_i^+ \wedge not\ \left(\mathcal{B}^- \cup \mathcal{B}_i^-\right)\ (i = 1, \dots, n)$$

where $B \leftarrow \mathcal{B}_i^+ \wedge not\ \mathcal{B}_i^-$ $(i = 1, \dots, n)$ are all rules with head B.

Note that any semantics SEM satisfying GPPE and Elimination of Tautologies can be seen as extending SLD by doing some *Loop-checking*. We will call such semantics *NMR-semantics* in order to distinguish them from the classical *LP-semantics* which are based on SLDNF or variants of Clark's completion $comp(P)$:

- *NMR-Semantics = SLDNF + Loop-check.*

The last principle we need is the observation that a clause $A \leftarrow B, C, not\ D, not\ E$ is certainly subsumed by $A \leftarrow B, not\ E$. We can get rid of non-minimal rules by simply deleting them.

4.10. Principle *(Subsumption).*
In a program P we can delete a rule $A \leftarrow \mathcal{B}^+ \wedge not\ \mathcal{B}^-$ whenever there is another rule $A \leftarrow \mathcal{B}'^+ \wedge not\ \mathcal{B}'^-$ with

$$\mathcal{B}'^+ \subseteq \mathcal{B}^+ \text{ and } \mathcal{B}'^- \subseteq \mathcal{B}^-.$$

As a simple example, the rule $A \leftarrow B, C, not\ D, not\ E$ is subsumed by the 3 rules $A \leftarrow C, not\ D, not\ E$, $A \leftarrow B, C, not\ E$ and by $A \leftarrow C, not\ E$.

What is a Semantics?
Up to now, we have introduced four principles and we used the term *semantics of a program* in a loose, imprecise way. We end this section with a precise notion of what we understand by a semantics.

As a first attempt, we can view a semantics as a mapping that associates to any program a set of positive atoms and a set of default atoms. In the case of SLD-Resolution the positive atoms are the ground instances of all derivable atoms. But sometimes we also want to derive negative atoms (like in our two examples above). Our *Orientation*-Principle formalizes a minimal requirement for deriving such default-atoms.

Of course, we also want that a semantics SEM should *respect* the rules of P, i.e. whenever SEM makes the body of a rule true, then SEM should also make the head of the rule true. But it can (and will) happen that a semantics SEM does not always decide *all* atoms. Some atoms A are not derivable nor are their default-counterparts *not* A. This means that a semantics SEM can view the body of a rule as being *undefined*.

Suppose a semantics SEM treats the body of a program rule as undefined. What should we conclude about the head of this rule? We will only require that this head is not treated as false by SEM — it could be true or undefined as well. This means that we require a semantics to be compatible with the program *viewed as a 3-valued theory* — the three values being **t** ("true"), **f** ("false") and **u** ("undefined"). For the understanding it is not necessary to go deeper into 3-valued logic. We simply note that we interpret "$\leftarrow$" as the Kleene-connective which is true for "$\mathbf{u} \leftarrow \mathbf{u}$" and false for "$\mathbf{f} \leftarrow \mathbf{u}$".

Our discussion shows that we can view a semantics SEM as a 3-valued model of a program. In classical logic, there is a different viewpoint. For a given theory T we consider there the set of all classical models $\mathrm{MOD}(T)$ as the semantics. The intersection of all these models is of course a 3-valued model of T, but $\mathrm{MOD}(T)$ contains more information. In order to formalize the notion of semantics as general as possible we define

4.11. DEFINITION *(SEM).*
A semantics SEM is a mapping from the class of all programs into the powerset of the set of all 3-valued structures. SEM assigns to every program P a set of 3-valued models of P:

$$\mathrm{SEM}(P) \ \subseteq \ \mathrm{MOD}^{\mathcal{L}_P}_{3-\mathrm{val}}(P).$$

This definition covers both the classical viewpoint (classical models are 2-valued and therefore special 3-valued models) as well as our first attempt in the beginning of this subsection. Later on, we will be really interested only in Herbrand models. It is also convenient to slightly abuse notation and view SEM(P) as a set consisting of atoms and default literals, namely those that are true in all intended models.

4.3. WFS and STABLE Semantics

The well-founded semantics, originally introduced in [Van Gelder, Ross and Schlipf 1988], is the *weakest* semantics satisfying our 4 principles (see [Brass and Dix 1999, Brass and Dix 1998, Dix 1995*a*]). We call a semantics

$$\mathrm{SEM}_1 \text{ weaker than } \mathrm{SEM}_2, \text{ written } \mathrm{SEM}_1 \leq_k \mathrm{SEM}_2,$$

if for all programs P and all atoms or default-atoms l the following holds: $\mathrm{SEM}_1(P) \models l$ implies $\mathrm{SEM}_2(P) \models l$. I.e. all atoms derivable from SEM_1 with respect to P are also derivable from SEM_2. The notion $\leq_k$ refers to the knowledge ordering in three-valued logic. Hence we have the following (due to [Brass and Dix 1999]):

4.12. THEOREM (Well-founded Semantics WFS).
There exists the weakest semantics satisfying our four principles Elimination of Tautologies, Reduction, Subsumption and GPPE. This semantics is called well-founded semantics WFS. WFS associates to every program P a unique 3-valued model of P.

This theorem gives a rather indirect, yet very interesting, explanation of WFS. But it can be shown, that all our transformations can be *applied* to really compute WFS.

4.13. THEOREM (Confluent Calculus for WFS,[Brass and Dix 1998]).
The calculus consisting of our four transformations is confluent, i.e. whenever we arrive at an irreducible program, it is uniquely determined. The order of the transformations does not matter.

For finite propositional programs, it is also strongly terminating *for* fair[8] *sequences of transformations: any program P is therefore associated a unique normal form $res(P)$. The well-founded semantics of P can be read off from $res(P)$ as follows*

$$\mathrm{WFS}(P) = \{A : \; A \leftarrow \quad \in res(P)\} \cup \{not\ A : \; A \textit{ is in no head of } res(P)\}$$

As an example let us consider the program

$$\begin{array}{rrcl} P: & p & \leftarrow & q, not\ r \\ & q & \leftarrow & p, not\ r \\ & r & \leftarrow & not\ r \\ & s & \leftarrow & not\ p \\ & t & \leftarrow & q \end{array}$$

We can first apply GPPE to q in the first and the fifth rule to obtain P'. Then we can remove the first rule (tautology) and also the second and fifth, by 2 applications of GPPE to p (since there is no rule with p in the head, GPPE removes the whole rule). We get P''. Finally we can apply reduction ($not\ p$ is true) and get the residual program $res(P)$.

$$\begin{array}{rrcl} P': & p & \leftarrow & p, not\ r \\ & q & \leftarrow & p, not\ r \\ & r & \leftarrow & not\ r \\ & s & \leftarrow & not\ p \\ & t & \leftarrow & p, not\ r \end{array} \qquad \begin{array}{rrcl} P'': & & & \\ & & & \\ & r & \leftarrow & not\ r \\ & s & \leftarrow & not\ p \\ & & & \end{array} \qquad \begin{array}{rrcl} res(P): & & & \\ & & & \\ & r & \leftarrow & not\ r \\ & s & \leftarrow & \\ & & & \end{array}$$

Consequently, WFS(P) $= \{not\ p, not\ q, s, not\ t\}$.

We defined WFS as the weakest semantics satisfying our four principles. This already indicates that there are stronger semantics. One of the main competing approaches is the stable semantics STABLE. The stable semantics associates to any program P a set of 2-valued models, like classical predicate logic. STABLE satisfies the following property, in addition to those that have been already introduced:

4.14. PRINCIPLE *(Elimination of Contradictions).*
Suppose a program P has a rule which contains the same atom A and *not* A in its body. Then we can eliminate this rule without changing the semantics.

[8]This condition will be discussed in more detail in Section 5, in particular in Subsection 5.1.

This principle can be used, in conjunction with the others to define the stable semantics as shown by [Brass and Dix 1997].

4.15. THEOREM (STABLE).
There exists the weakest semantics satisfying our five principles Elimination of Tautologies, Reduction, Subsumption, GPPE and Elimination of Contradictions.

If a semantics SEM satisfies *Elimination of Contradictions* it is based on 2-valued models [Brass and Dix 1997]. The underlying idea of STABLE is that any atom in an intended model should have a definite reason to be true or false. This idea was made explicit by [Bidoit and Froidevaux 1991*a*, Bidoit and Froidevaux 1991*b*] and, independently, by [Gelfond and Lifschitz 1988]. We use the latter terminology and introduce the Gelfond-Lifschitz transformation: for a program P and a model $N \subseteq B_P$ we define

$$P^N := \{\text{rule}^N : \text{ rule} \in P\}$$

where rule $:= A \leftarrow B_1, \ldots, B_n, not\ C_1, \ldots, not\ C_m$ is transformed as follows

$$(\text{rule})^N := \begin{cases} A \leftarrow B_1, \ldots, B_n, & \text{if } \forall j : C_j \notin N, \\ \mathbf{t}, & \text{otherwise.} \end{cases}$$

Note that P^N is always a *definite* program. We can therefore compute its least Herbrand model M_{P^N} and check whether it coincides with the model N with which we started:

4.16. DEFINITION *(STABLE).*
N is called a *stable* model[9] of P *if and only if* $M_{P^N} = N$. The stable semantics associates to every program P the set of stable models of P.

What is the relationship between STABLE and WFS? We have seen that they are based on rather identical principles.

- Stable models N extend WFS: $l \in$ WFS(P) implies $N \models l$.
- If WFS(P) is two-valued, then WFS(P) is the unique stable model.

But there are also differences. We consider the following example:

4.17. EXAMPLE *(Van Gelder's Example).*
Assume we are describing a two-players game like checkers. The two players alternately move a stone on a board. The moving player wins when his opponent has no more move to make. We can formalize that by

- $wins(x) \leftarrow move\_from\_to(x, y), not\ wins(y)$

meaning that

- the situation x is won (for the moving player A), if he can lead over[10] to a situation y that can never be won for B.

[9] Note that we only consider Herbrand models.
[10] With the help of a regular move, given by the relation $move\_from\_to/2$.

Assume the facts $move\_from\_to(a,b)$, $move\_from\_to(b,a)$ and $move\_from\_to(b,c)$. Our query to this program P_{game} is *?-* $wins(b)$. Here we have no problems with floundering, but using SLDNF we get an infinite sequence of oscillating SLD-trees (none of which finitely fails).

WFS derives the right results

$$\text{WFS}(P_{\text{game}}) = \{\mathit{not}\ wins(c), wins(b), \mathit{not}\ wins(a)\}$$

which matches completely with our intuitions. We get a different result when we consider the above clause $wins(x) \leftarrow move\_from\_to(x,y),\ \mathit{not}\ wins(y)$ together with the following facts: $move\_from\_to(a,b)$, $move\_from\_to(b,a)$, as well as $move\_from\_to(b,c)$, and $move\_from\_to(c,d)$. In this particular case we have two stable models: $\{wins(a),\ wins(c)\}$ and $\{wins(b),\ wins(c)\}$ and therefore

$$\text{WFS}(P) = \{wins(c),\ \mathit{not}\ wins(d)\} = \bigcap_{\text{N a stable model of P}} N.$$

This means that the 3-valued well-founded model is exactly the set of all atoms or default-atoms true in all stable models. Again, there are examples where this is not always the case [Brewka and Dix 2001, Brewka et al. 1997].

The main differences between STABLE and WFS are

- STABLE is not always consistent,
- STABLE does not allow for a goal-oriented implementation.

The inconsistency comes from odd, negative cycles

$$\text{STABLE}(p \leftarrow \mathit{not}\ p) = \emptyset.$$

The idea to consider 2-valued models for a semantics necessarily implies its inconsistency [Brass and Dix 1997]. Note that the well-founded model exists: it is the empty 3-valued model. Sufficient criteria for the existence of stable models are contained in [Dung 1992, Fages 1993].

That STABLE does not allow for a Top-Down evaluation is a more serious drawback and has nothing to do with inconsistency.

4.18. Example *(STABLE is not Goal-Oriented.).*

$$\begin{array}{lrcl@{\qquad}lrcl}
P_{rel(a)}: & a & \leftarrow & \mathit{not}\ b & P: & a & \leftarrow & \mathit{not}\ b \\
 & b & \leftarrow & \mathit{not}\ a & & b & \leftarrow & \mathit{not}\ a \\
 & & & & & p & \leftarrow & \mathit{not}\ p \\
 & & & & & p & \leftarrow & a
\end{array}$$

$P_{rel(a)}$ is the subprogram of P that consists of all rules that are relevant to answer the query *?-* a. It has two stable models $\{a\}$ and $\{b\}$ — a is not true in all of them. But the program P has the unique stable model $\{p, a\}$, so a is true in all stable models of P.

The last example shows that the truth value of an atom a also depends on atoms that are totally unrelated with a! STABLE is not *relevant* or *modular* (see [Dix 1992a, Dix 1992b, Dix 1995a]). This is considered a drawback of STABLE by many people. Note that a straightforward modification of STABLE is not possible [Dix and Müller 1994a, Dix and Müller 1994b]. Other extensions of WFS based on confluent calculi have been considered in [Dix and Osorio 1999, Dix, Osorio and Zepeda 2001].

4.4. GCWA

What happens if we also allow disjunction in the head? It turns out that even for positive programs with disjunctions, we can express relations which belong to the second level of the polynomial hierarchy. Also, the complexity of deriving negative literals is at the second level. Therefore an extension of WFS to disjunctive programs can not have the same low complexity.

Concerning the right semantics for such programs, we are in the same situation as in the previous subsection — for positive programs there is general agreement while for disjunctive programs with default-negation there exist several competing approaches.

GCWA is defined for positive disjunctive programs consisting of rules of the form

$$A_1 \vee \ldots \vee A_n \ \leftarrow \ B_1, \ldots, B_m$$

by declaring all the minimal models to be the intended ones:

4.19. DEFINITION *(GCWA).*
The generalized closed world assumption GCWA of P is the semantics given by the set of all classical (i.e. two-valued) minimal Herbrand models of P:

$$\mathrm{GCWA}(P) := \mathrm{MinMod}(P)$$

Originally, Minker denoted by GCWA(P) a set of negative atoms with the property that $P \cup \mathrm{GCWA}(P) \models$ *not* A *if and only if* $\mathrm{MinMod}(P) \models$ *not* A but we prefer here to denote by GCWA a semantics in the sense of Definition 4.11.

In Subsections 4.2 and 4.2 we have introduced the general notion of a semantics and various principles. Do they carry over to the disjunctive case? Fortunately, the answer is yes. In addition, GCWA not only satisfies all these properties, it is also uniquely characterized by them as the next theorem shows.

4.20. THEOREM (Characterization of GCWA, [Brass and Dix 1997]).
Let SEM be a semantics in the sense of Definition 4.11 satisfying GPPE *and* Elimination of Tautologies*.*

a) *Then: SEM*$(P) \subseteq \mathrm{MinMod}_{2-\mathrm{val}}(P)$ *for positive disj. programs* P*.*
I.e. any such semantics is already based on 2-valued (as opposed to 3-valued) minimal models. In particular, GCWA is the weakest semantics with these properties.

b) *If SEM is non-trivial and satisfies in addition*[11] Isomorphy *and* Relevance*, then it coincides with GCWA on positive disjunctive programs.*

4.5. D-WFS and D-STABLE

Before we can state the definition of D-WFS we have to extend our principles to disjunctive programs with default-negation.

4.21. DEFINITION *(Disjunctive Logic Program).*
A *disjunctive* logic program consists of a finite number of *rules* of the form

$$A_1 \vee \ldots \vee A_k \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$$

where $A_1, \ldots, A_k, B_1, \ldots, B_m, C_1, \ldots, C_n$ are positive atoms (containing possibly free variables). We call $A_1 \vee \ldots A_k$ the *head* of the rule, $B_1, \ldots, B_m$ its *positive body* and $C1, \ldots, C_n$ its *negative body*. The comma represents conjunction $\wedge$.

We abbreviate these rules by

$$\mathcal{A} \leftarrow \mathcal{B}^+, \mathit{not}\ \mathcal{B}^-$$

where $\mathcal{A} := \{A_1, \ldots, A_k\}$, $\mathcal{B}^+ := \{B_1, \ldots, B_m\}$, $\mathcal{B}^- := \{C_1, \ldots, C_n\}$.

We also generalize our notion of a semantics slightly:

4.22. DEFINITION *(Operator $\mid\!\sim$, Semantics $\mathcal{S}_{\mid\sim}$).*
By a semantic operator $\mid\!\sim$ we mean a binary relation between logic programs and pure disjunctions which satisfies the following three arguably obvious conditions:
1. *Right Weakening:* If $P \mid\!\sim \psi$ and $\psi \subseteq \psi'$[12], then $P \mid\!\sim \psi'$.
2. *Necessarily True:* If $\mathcal{A} \leftarrow \mathbf{t} \in P$ for a disjunction $\mathcal{A}$, then $P \mid\!\sim \mathcal{A}$.
3. *Necessarily False:* If $A \notin \text{Head\_atoms}(P)$[13] for a $\mathcal{L}$-ground atom A, then $P \mid\!\sim \mathit{not}\ A$.

Given such an operator $\mid\!\sim$ and a logic program P, by the semantics $\mathcal{S}_{\mid\sim}(P)$ of P determined by $\mid\!\sim$ we mean the set of all pure disjunctions derivable by $\mid\!\sim$ from P, i.e., $\mathcal{S}_{\mid\sim}(P) := \{\psi \mid P \mid\!\sim \psi\}$.

In order to give a unified treatment in the sequel, we introduce the following notion:

4.23. DEFINITION *(Invariance of $\mid\!\sim$ under a Transformation).*
Suppose that a program transformation *Trans* : $P \mapsto \mathit{Trans}(P)$ mapping logic programs into logic programs is given. We say that the operator $\mid\!\sim$ is invariant under *Trans* (or that *Trans* is a $\mid\!\sim$-equivalence transformation) *if and only if*

$$P \mid\!\sim \psi \iff \mathit{Trans}(P) \mid\!\sim \psi$$

[11] See [Brass and Dix 1997] for the precise definitions of Relevance and Isomorphy.
[12] I. e. ψ is a subdisjunction of ψ'.
[13] We denote by Head_atoms(P) the set of all (instantiations of) atoms occurring in some rule-head of P.

for any pure disjunction ψ and any program P.

All our principles introduced below can now be naturally extended.

4.24. Definition *(Elimination of Tautologies, Non-Minimal Rules).*
Semantics $\mathcal{S}_{\mid\sim}$ satisfies **a)** the *Elimination of Tautologies*, resp. **b)** the *Elimination of Non-Minimal Rules if and only if* $\mid\sim$ is invariant under the following transformations:

a) Delete a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \mathcal{B}^-$ with $\mathcal{A} \cap \mathcal{B}^+ \neq \emptyset$.
b) Delete a rule $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \mathcal{B}^-$ if there is another rule $\mathcal{A}' \leftarrow \mathcal{B}^{+\prime} \wedge \mathit{not}\ \mathcal{B}^{-\prime}$ with $\mathcal{A}' \subseteq \mathcal{A}$, $\mathcal{B}^{+\prime} \subseteq \mathcal{B}^+$, and $\mathcal{B}^{-\prime} \subseteq \mathcal{B}^-$.

Our partial evaluation principle has now to take into account disjunctive heads. The following definition was introduced independently by [Brass and Dix 1994, Brass and Dix 1997, Sakama and Seki 1994]:

4.25. Definition *(GPPE for Disjunctive Programs).*
Semantics $\mathcal{S}_{\mid\sim}$ satisfies GPPE *if and only if* it is invariant under the following transformation: *Replace a rule* $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \mathcal{B}^-$ *where* $\mathcal{B}^+$ *contains a distinguished atom* B *by the rules*

$$\mathcal{A} \cup \big(\mathcal{A}_i \setminus \{B\}\big) \ \leftarrow\ \big(\mathcal{B}^+ \setminus \{B\}\big) \cup \mathcal{B}_i^+ \ \wedge\ \mathit{not}\ \big(\mathcal{B}^- \cup \mathcal{B}_i^-\big)\ (i = 1, \ldots, n)$$

where $\mathcal{A}_i \leftarrow \mathcal{B}_i^+ \wedge \mathit{not}\ \mathcal{B}_i^-$ $(i = 1, \ldots, n)$ are all the rules with $B \in \mathcal{A}_i$.

Note that we are free to select a specific positive occurrence of an atom B and then perform the transformation. The new rules are obtained by replacing B by the bodies of all rules r with head literal B and adding the remaining head atoms of r to the head of the new rule.

Here is the analogue of Principle 4.7:

4.26. Definition *(Positive and Negative Reduction).*
Semantics $\mathcal{S}_{\mid\sim}$ satisfies **a)** *Positive*, resp. **b)** *Negative* Reduction *if and only if* $\mid\sim$ is invariant under the following transformations:

a) Replace $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \mathcal{B}^-$ by $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \big(\mathcal{B}^- \cap \mathrm{Head\_atoms}(P)\big)$.
b) Delete $\mathcal{A} \leftarrow \mathcal{B}^+ \wedge \mathit{not}\ \mathcal{B}^-$ if there is a rule $\mathcal{A}' \leftarrow \mathbf{t}$ with $\mathcal{A}' \subseteq \mathcal{B}^-$.

Now the definition of a disjunctive counterpart of WFS is straightforward:

4.27. Definition *(D-WFS).*
There exists the weakest semantics satisfying positive and negative Reduction, GPPE, Elimination of Tautologies and non-minimal Rules. We call this semantics D-WFS.

As it was the case for WFS, our calculus of transformations is also confluent [Brass and Dix 1998].

4.28. Theorem (Confluent Calculus for D-WFS).
The calculus consisting of our four transformations is confluent and strongly terminating for propositional programs and fair[14] *sequences of transformations. I.e. we always arrive at an irreducible program, which is uniquely determined. The order of the transformations does not matter.*

Therefore any program P is associated a unique normal form $res(P)$. The disjunctive well-founded semantics of P can be read off from $res(P)$ as follows

$$\psi \in \textit{D-WFS}(P) \iff \begin{array}{l} \text{there is } \mathcal{A} \subseteq \psi \text{ with } \mathcal{A} \leftarrow \mathbf{t} \in res(P) \quad \text{or} \\ \text{there is not } A \in \psi \text{ and } A \notin \mathrm{Head\_atoms}(res(P)). \end{array}$$

Note that the original definition of WFS, or any of its equivalent characterizations, does not carry over to disjunctive programs in a natural way.

Two properties of D-WFS are worth noticing

- For positive disjunctive programs, D-WFS coincides with GCWA.
- For non-disjunctive programs with negation, D-WFS coincides with WFS.

Unlike the well-founded semantics, the original definition of stable models carries over to disjunctive programs quite easily:

4.29. Definition *(D-STABLE).*
N is called a *stable* model[15] of P *if and only if* $N \in \mathrm{MinMod}(P^N)$. The disjunctive stable semantics associates to any disjunctive program P the set of stable models of P.

In the last definition P^N is the positive disjunctive program obtained from P by applying the Gelfond/Lifschitz transformation (as introduced before Definition 4.16 — its generalization to disjunctive programs is obvious).

Analogously to D-WFS the following two properties of D-STABLE hold:

- For positive disjunctive programs, D-STABLE coincides with GCWA.
- For non-disjunctive programs with negation, D-STABLE coincides with STABLE.

What about our transformations introduced to define D-WFS? Do they hold for D-STABLE? Yes, they are indeed true. The most difficult proof is the one for GPPE. It was proved by [Brass and Dix 1994, Sakama and Seki 1994] independently that stable models are preserved under GPPE. Moreover, [Brass and Dix 1997] proved that STABLE can be almost uniquely determined by GPPE:

4.30. Theorem (Characterization of D-STABLE).
Let SEM be a semantics satisfying GPPE, Elimination of Tautologies, *and* Elimination of Contradictions. *Then: $SEM(P) \subseteq$ D-STABLE(P). Moreover, D-STABLE is the weakest semantics satisfying these properties.*

[14] this condition will be discussed in more detail in Section 5, in particular in Subsection 5.1.
[15] Note that we only consider Herbrand models.

5. Implementing Nonmonotonic Semantics

In this section we present and discuss methods for implementing the two most popular nonmonotonic semantics for logic programs, the well-founded semantics and the stable model semantics. We study first normal programs (Subsections 5.1, 5.2) and then disjunctive programs (Subsections 5.3, 5.4). It turns out that the well-founded semantics is of *polynomial* complexity, in contrast to the stable semantics, which is located one level higher in the polynomial hierarchy. Therefore we only sketch a method for implementing the well-founded semantics and concentrate on efficient implementations of the stable semantics (Subsection 5.2).

We refer the interested reader to [Dix, Furbach and Nerode 1997], which contains a collection of nonmonotonic system descriptions and to the following webpage which is actively maintained and contains information on various logic programming systems that concentrate on nonmonotonic aspects (different kinds of negation, disjunction, abduction etc.): `<URL:http://www.uni-koblenz.de/ag-ki/LP/>`.

Before considering complexity results, we introduce an important notion which helps us to classify and to highlight different behaviour of certain classes of programs under some semantics.

5.1. Definition *(Dependency Graph $\mathcal{G}_P$).*
For a general disjunctive program P with negation, the *dependency graph* $\mathcal{G}_P$ is a finite directed graph whose vertices are the predicate symbols from P. There is a *positive* (respectively *negative*) edge from p to p' *if and only if* there is a clause in P with

1. both p and p' occurring in the head, or
2. p in its head and p' occurring positively (respectively negative) in its body.

We also say

- *p depends on p'* if there is a path in $\mathcal{G}_P$ from p to p'.
  By definition, p depends on itself.
- *p depends positively* (resp. *negatively*) on p' if there is a path in $\mathcal{G}_P$ from p to p' containing only *positive* edges (resp. *at least one negative* edge).
  By definition p depends positively on itself.
- *p depends evenly* (resp. *oddly*) on p' if there is a path in $\mathcal{G}_P$ from p to p' containing an *even* (resp. *odd*) number of negative edges.
  By definition p depends evenly on itself.

The following properties of a program P turn out to be very important:

stratified: no predicate depends negatively on itself[16],
strict: there are no dependencies that are both even and odd,
call-consistent: no predicate depends oddly on itself[17],
allowedness: every variable occurring in a clause must occur in at least one positive atom of the body of that clause.

[16] or: there are no cycles containing at least one *negative* edge.
[17] or: there are no odd cycles.

For first-order programs an analogue of the dependency graph is easily constructed: the vertices of this infinite graph are exactly the *instantiated predicates* of the program. But note that for such programs the property of *being stratified* is not decidable [Cholak and Blair 1994].

The allowedness-condition (also called *range-restrictedness*) excludes constructs of the form $equal(x, x) \leftarrow$. This has the important effect that evaluating the truth of such a formula is to a large extent independent of the underlying domain (universe of discourse). It suffices to evaluate it on a *finite* subset of the underlying Herbrand domain. This ensures that in a bottom-up evaluation the underlying domains remain finite.

Let us consider some cases where restricting the class of programs has strong effects on the underlying semantics. For example, stratified programs always possess one unique stable model, which coincides with the well-founded model (which itself is therefore twovalued) and is denoted by $\mathrm{M}_{\mathbf{P}}$. Call-consistent programs also possess at least one stable model and for strict programs, the well-founded model is exactly the (3-valued) intersection of all stable models [Dung 1992].

We first collect some overall complexity results both for first-order as well as for propositional programs (we refer the reader to [Balcázar et al. 1988, Johnson 1990, Odifreddi 1989, Papadimitriou 1994, Eiter and Gottlob 1993, Gottlob 1992] for detailed investigations of complexity classes).

First-Order Programs

Tables 2 and 3 show that the full first-order versions are highly undecidable, if negation is allowed ($\mathbf{\Pi^1_1}$-compl.[18] over $\mathbb{N}$). Also, there is no difference between WFS, STABLE and their disjunctive versions (all are $\mathbf{\Pi^1_1}$-complete). Only for definite programs and positive disjunctive programs is the problem of deriving a positive atom recursively enumerable. But already deriving negative ground literals is $\mathbf{\Pi^0_1}$-complete (for definite programs) and even $\mathbf{\Pi^0_2}$-complete for positive disjunctive programs.

Although we did not introduce Clark's completion semantics COMP nor its 3-valued variant COMP_3 (but see Subsection 5.4), we mention in passing that both semantics are important because they constitute the *declarative* counterpart of SLDNF-resolution. This itself is too weak for our purposes (see the remark after Principle 4.9 in Section 4). More on Clark's completion can be found in [Brewka and Dix 2001, Clark 1978, Fitting 1985].

This shows that systems handling first-order programs must be necessarily incomplete. They can be only complete under additional assumptions on the underlying class of programs like

1. allowed programs without function symbols, also called *safe DATALOG*-programs, or
2. programs, where never a *negative literal with free variables* is selected (to resolve against), so called *non-floundering*-programs.

[18] To illustrate $\mathbf{\Pi^1_1}$-completeness, we simply mention that such problems are of much higher complexity than the well-known halting problem of Turing-machines.

	Complexity	
	1. ord. prog. (with functions)	*prop. prog. (no variables)*
$\mathbf{M_P}$ *(P is Horn)*	A: $\mathbf{\Sigma^0_1}$-compl. *not* A: $\mathbf{\Pi^0_1}$-compl.	**linear** in $\|P\|$
$\mathbf{M_P^{supp}}$ *(P is stratified)*	**arithm.**-compl. (M_P^{supp} is $\mathbf{\Sigma^0_n}$)	**linear** in $\|P\|$
COMP	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	**co-NP**-compl.
$\mathbf{COMP_3}$	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	**linear** in $\|P\|$
STABLE	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	**co-NP**-compl.
WFS	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	**linear** in $\#At \times \|P\|$

Table 2: Complexity of Non-Disjunctive Semantics

	Complexity	
	1. ord. prog. (with functions)	*prop. prog. (no variables)*
GCWA *(P is positive)*	A: $\mathbf{\Sigma^0_1}$-compl. *not* A: $\mathbf{\Pi^0_2}$-compl.	A: **co-NP**-compl. *not* A: $\mathbf{\Pi^P_2}$-compl.
D-WFS	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	$\mathbf{\Pi^P_2}$-compl.
D-STABLE	$\mathbf{\Pi^1_1}$-compl. over $\mathbb{N}$	$\mathbf{\Pi^P_2}$-compl.

Table 3: Complexity of Disjunctive Semantics

For propositional non-disjunctive programs we have a different situation (see Table 2). For non-disjunctive programs, WFS is at most quadratic, while STABLE is **co-NP**-complete. Thus for computing stable models, it makes sense to compute first the WFS (because this is cheap) and then use this result to reduce the program (by applying some of our transformations introduced in Section 4). Therefore an efficient computation of WFS also pays off for computing stable models.

For disjunctive propositional programs there is no difference in the complexity between D-WFS and D-STABLE: they are both $\mathbf{\Pi^P_2}$-complete. Nevertheless, since D-STABLE is stronger than D-WFS, it makes sense to use an efficient computation of D-WFS first because many queries can be already decided (i.e. shown to be true or false) on this basis.

5.1. WFS for Normal Programs

One possibility to compute WFS is to just apply the confluent calculus of transformations described in Theorem 4.13. However, there are two details worth noticing. The first is the *fairness*-assumption mentioned in Theorem 4.13 and the second the complexity of the GPPE-transformation.

Fairness: Already the simple program consisting of just the loop "$p \leftarrow p$" shows a problem with termination. Applying GPPE leads to the same program, so GPPE can be applied infinitely often without leading to the residual program. There can also occur an oscillation:

$$P_{\mathrm{loop}}: \begin{array}{ccl} p & \leftarrow & q \\ q & \leftarrow & p \\ r & \leftarrow & p, \neg r \end{array}$$

If we apply GPPE_p to the third clause, this clause is replaced by $r \leftarrow q, \neg r$. We can now apply GPPE_q again to this clause and get the original program. So we have an oscillation

$$\mathrm{GPPE}_q \circ \mathrm{GPPE}_p(P_{\mathrm{loop}}) = P_{\mathrm{loop}}.$$

To get rid of these problems, we call a sequence of program transformations *fair*, if in the corresponding sequence of programs

1. every positive body-atom is eventually removed (either by removing the whole clause using a suitable transformation or by an application of GPPE), and
2. every tautology clause is eventually removed (either by applying *Elimination of Tautologies* or another suitable transformation).

With this fairness assumption, our calculus is strongly terminating.

Complexity: The second, and a much more important point is that the size of the residual program is in general *exponential* in the size of the original program! This is because rules of the form

$$\begin{array}{ccl} p & \leftarrow & q \\ q & \leftarrow & r \\ r & \leftarrow & p \end{array}$$

have to be fully evaluated by GPPE until the tautology transformation can be applied and all atoms classified as false. Recently it was shown by [Brass, Zukowski and Freitag 1997, Brass, Dix, Freitag and Zukowski 2001] how a small modification of the residual program, which still satisfies the nice characterization of computing WFS as given in Theorem 4.13, results in a *polynomial* computation. The idea is to replace the GPPE by another transformation, called *Loop-detection* that does not produce an exponential blow up. Note that all other transformations can be done in linear time.

To be more concrete, let us define the following two instances of GPPE ([Brass, Dix, Freitag and Zukowski 2001]).

5.2. DEFINITION *(Success and Failure).*
The success-transformation can be applied to a program P whenever there is a rule $A \leftarrow$ in P. Success then removes a positive occurrence of A in the body of another rule.

Dually, the failure-transformation can be applied to a program P whenever there is an atom A which does not occur in any head of a rule. Failure then removes a rule which contains A positively in its body.

If we forget about GPPE and the tautology rule we get the following interesting result:

5.3. THEOREM (Linear Complexity).
The calculus of rules consisting of success, failure and reduction (see Definition 4.7) is also confluent. In addition, it is strongly terminating and the normal form $nf(P)$ can be computed in linear time. The semantics according to the normal form coincides with the least fixpoint of the Fitting operator Φ_P and thus coincides with $comp_3$:

$$\mathrm{lfp}(\Phi_P) = \{A : \ A \leftarrow \quad \in nf(P)\} \cup \{\textit{not } A : \ A \textit{ is in no head of nf(P)}\}$$

The last result is, from the complexity point of view quite promising. But it just computes a weak approximation of the WFS. What is missing is a loop detection rule:

5.4. DEFINITION *(Loop Detection).*
The Loop Detection transformation can be applied to a program P whenever there is a set $S_{\mathrm{unfounded}}$ of ground atoms such that

- for each rule in $A \leftarrow \mathcal{B} \in P$, if $A \in S_{\mathrm{unfounded}}$ then $\mathcal{B} \cap S_{\mathrm{unfounded}} \neq \emptyset$.

Loop Detection then removes all rules of P which contain in their body a positive occurrence of an atom in $S_{\mathrm{unfounded}}$.

Although checking for such a set $S_{\mathrm{unfounded}}$ can be done in linear time, we may need various applications of it and therefore may reach an overall quadratic behaviour. A worst case estimate is the number of atoms $\#At$ occurring in P.

All the transformations above taken together, we get a calculus that also satisfies Theorem 4.13, i.e. it is confluent and the normal form (which is computable in quadratic time) determines the well-founded semantics.

But, as recently shown by [Berman, Schlipf and Franco 1995] WFS can be computed with *linear* complexity for a large class of programs. Rather than giving a detailed investigation, we just sketch a similar approach here. The method consists in three steps:

1. We first compute the *strongly connected components* of the program (as given by the dependency graph). As is well-known, this can be done in linear time (see eg. [Tarjan 1972, Berman et al. 1995]).

2. We start with working on the lowest component and propagate the values of atoms to the higher components.
3. In every connected component we iterate the following
   (a) apply *reduction* as well as *success* and *failure* until a fixpoint is reached. As mentioned in Theorem 5.3, this fixpoint corresponds exactly to the fixpoint of the Fitting operator Φ_P.
   (b) apply *loop-detection*, which amounts to computing the maximal unfounded set $S_{\text{unfounded}}$ [Van Gelder et al. 1988].

   If there are either *only* positive dependencies or *no* predicate defined in that component depends positively on itself, then no iteration of the two steps above are needed.

5.5. THEOREM (Implementation of WFS).
The above described algorithm is provably better than the alternating fixpoint procedure of [Van Gelder 1989], which already works in $\#At \times |P|$, where $\#At$ is the number of atoms and $|P|$ the size of the program.

Moreover, the algorithm is linear for the class of all programs, where in each strongly connected component the following holds: either all the head-atoms defined in that component depend only positively from each other *or* no predicate defined in that component depends positively on itself *(also called* positive-acyclic*). Note that these two conditions can change from one connected component to the other.*

5.6. EXAMPLE *(Algorithm for WFS using SCC's).*
We consider the following program, which contains both negative and positive dependencies.

$$\begin{array}{lllllllll}
p_1 & \leftarrow & q_1 & p_2 & \leftarrow & \mathit{not}\ q_2 & q_3 & \leftarrow & \mathit{not}\ w_3 \\
q_1 & \leftarrow & r_1 & q_2 & \leftarrow & \mathit{not}\ s_2 & w_3 & \leftarrow & \mathit{not}\ z_3, x_3 \\
r_1 & \leftarrow & q_1, z_3 & s_2 & \leftarrow & t_2 & z_3 & \leftarrow & \\
r_1 & \leftarrow & p_2, \mathit{not}\ q_3 & t_2 & \leftarrow & \mathit{not}\ q_3, t_2 & x_3 & \leftarrow & \mathit{not}\ x_3
\end{array}$$

We have already split the program into strongly connected components and start with the lowest component, which is positive-acyclic. z_3 gets immediately true, while w_3 gets false and therefore q_3 becomes true and x_3 remains undefined. We propagate these values to the higher components. Note that the next component now also gets positive-acyclic, because $t_2 \leftarrow \mathit{not}\ q_3, t_2$ is removed due to propagation of the truth value of q_3. Consequently t_2, s_2 and p_2 are false while q_2 is true. The last component is purely positive, so we need the loop detection (linear) to determine that p_1, q_1, r_1 are all false.

XSB from Warren et. al.
The most advanced system (in terms of a full-fledged programming environment with various tools to support a programmer) has been implemented by David Warren and his group in Stony Brook: XSB is described by [Rao, Sagonas, Swift, Warren and Freire 1997, Sagonas, Swift and Warren 1994] and available by anonymous

ftp from `ftp.cs.sunysb.edu/pub/XSB/`. XSB is a full Prolog system which approximates the ISO standard. It can handle programs with negation and implements the well-founded semantics with respect to negation. The main contrast to a usual Prolog system is that XSB is not based on SLD-resolution, but on SLG-resolution (see [Chen and Warren 1993, Chen, Swift and Warren 1995, Chen and Warren 1996]), which allows to handle loops. Consequently, a modified WAM, the SLG-WAM, is the kernel of XSB. XSB is terminating for many programs where usual Prolog systems fall into an infinite loop.

The main feature of SLG is its use of *tabling*. For definite programs, SLG is similar to the OLDT-algorithm of [Tamaki and Sato 1986].

On an abstract level, the behaviour of SLG can be described by using the transformations *reduction*, *success*, *failure* and *loop detection* as used in Theorem 5.5. Of course, SLG works for rules containing variables, whereas our framework just treats propositional programs. In SLG, negative literals are delayed until they can be evaluated (i.e. their truth value can be computed). This is much the same as computing the modified normal form described above, where both negative literals and positive atoms are left in the body of rules. In fact our term *residual program* (but for disjunctive programs) was chosen according to the same term used in SLG.

In XSB one can use the residual program to compute stable models. Given a query and a program, the residual program with respect to that query is *almost ground*: this means that if variables are present in the residual program, they can be handled as *Skolem constants*. Then fast methods for computing stable models of propositional programs (see the next Subsection 5.2) can be used.

5.7. Theorem (Completeness and Termination of XSB).
XSB is complete and terminating for non-floundering DATALOG (wrt WFS). It also works for general programs but termination is not guaranteed. If it terminates, the computed residual program is almost ground (and finite).

5.2. STABLE for Normal Programs

The stable model semantics for normal logic programs is closely related to Moore's autoepistemic logic and Reiter's default logic whose implementations have been discussed in Section 2. Our aim is to employ the techniques introduced there and show how they can be further developed to exploit the more limited structure of logic program. This is the approach used in a system called **smodels** [Niemelä and Simons 1996] 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 implementing STABLE. 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., [Baral and Gelfond 1994] for a survey). An interesting feature of the approach presented in this section is that for ground programs it enables *polyno-*

mial space implementation which is essential when working with larger programs. This is different from some recent advanced approaches [Bell, Nerode, Ng and Subrahmanian 1994, Subrahmanian, Nau and Vago 1995] which have exponential worst-case space requirements. The SLG system developed by Chen and Warren [Chen and Warren 1995] is closest to the method presented in this section and we compare the two approaches in the end of the section.

Normal logic programs can be seen as a special subclass of default theories where a logic program rule of the form

$$A \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n$$

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 [Gelfond and Lifschitz 1990]. 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).

Our treatment of default logic in Section 2 is based on the notion of full sets (see Definition 2.3). When working with logic programs we use the representation of full sets introduced in Section 2.3 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 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.

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

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

where

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

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

The set of rules $\mathrm{Mon}(P,\mathrm{F})$ can be seen as a set of definite clauses and their deductive closure is just the unique least model of the clauses. Now

1. for every P-full F, $\mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{F})}(\emptyset)$ is a stable model of P and

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

Figure 8: A decision procedure for STABLE

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

5.9. Example. Consider the program P

$$p \leftarrow \mathit{not}\ q, r$$
$$q \leftarrow \mathit{not}\ p, r$$
$$r \leftarrow \mathit{not}\ q$$

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

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 2.3 (Fig. 3). In order to clarify the starting point we repeat the method in Fig. 8 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 $\mathit{not}\ A$ 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.

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 2.3 we

introduced a technique for developing such functions by employing the notions of a lower and upper bound:

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

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

- if $\varphi \in \mathbf{LB}(P, \mathrm{FS}')$, then include φ in FS' and
- if $A \notin \mathbf{UB}(P, \mathrm{FS}')$ 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 2.3 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 2.3. 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.

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

Consider the program P

$$\begin{aligned} & p \leftarrow q \\ & q \leftarrow p \\ & p \leftarrow s, \mathit{not}\ r \\ & s \leftarrow \mathit{not}\ t, \mathit{not}\ p \\ & r \leftarrow \mathit{not}\ t, \mathit{not}\ q \\ & t \leftarrow \mathit{not}\ r \end{aligned} \tag{5.1}$$

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

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

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

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

5.10. Example. Consider the program P defined in (5.1) 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, Rules 3–4 generalize the backward propagation of assumed truth values presented by [Chen and Warren 1995] and the upper bound rule generalizes loop detection discussed in the previous section to the case where assumptions are used. 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 (5.1) $\mathbf{expand}(P, \{p\})$ 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 [Dowling and Gallier 1984]. This leads to quadratic time worst case performance for **expand**. The performance can be improved by exploiting the strongly connected components of the dependency graph of the program for computing the upper bound using, e.g., a similar technique as that proposed in the previous section for the well-founded model computation.

conflict

For soundness and completeness it is sufficient that

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

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

choose

The function **choose** implements the search heuristics and $\mathbf{choose}(P, \mathrm{FS})$ 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 [Subrahmanian et al. 1995] 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 employed in the **smodels** system [Niemelä and Simons 1996]. 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 for 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.

5.11. EXAMPLE. We illustrate the method with two examples. Consider the following program P.

$p \leftarrow not\ q, not\ u$		$1.\ not\ u$			$1.\ not\ s$
$q \leftarrow not\ p, not\ u$		$\swarrow\searrow$			$2.\ not\ u$
$r \leftarrow p$	$2.\ not\ t$		$3.\ t$		$3.\ t$
$s \leftarrow not\ q$	$4.\ s$		$11.\ not\ s$		$\swarrow\searrow$
$s \leftarrow not\ r, q$	$\swarrow\searrow$		$12.\ q$	$4.\ not\ p$	$5.\ p$
$t \leftarrow not\ s$	$5.\ not\ r$	$6.\ r$	$13.\ not\ p$	$6.\ q$	$9.\ r$
$s \leftarrow not\ t$	$7.\ not\ p$	$9.\ p$	$14.\ not\ r$	$7.\ r$	$10.\ not\ q$
	$8.\ q$	$10.\ not\ q$	$15.\ not\ q$	$8.\ not\ r$	$11.\ s$
			$\times$	$\times$	$\times$

First we discuss the problem of determining all of 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 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 in to 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) $\mathrm{FS} = \{not\ u, t, not\ s, q, not\ p\}$ 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 $\mathrm{FS} = \{not\ u, not\ t, s, not\ r, not\ p, q\}$ and no conflicts are detected. Hence, a stable model of P has been found. This can be obtained by computing the deductive closure $\mathrm{Cn}_{\mathrm{Mon}(P,\mathrm{FS})}(\emptyset)$ 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 $= \{not\ s\}$ and let **test** to return always true. The tableau on the right hand side illustrates the execution of **extensions**$(P, \{not\ s\}, \varphi)$. Two branches are built but both are not completed because a conflict is detected. Hence, **extensions**$(P, \{not\ s\}, \varphi)$ 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 implementing STABLE is quite similar to that used in the SLG system developed by Chen and Warren [Chen and Warren 1995]. 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 [Niemelä and Simons 1996].

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 to implementing STABLE for non-ground programs: given a query, use an efficient implementation of SLG-resolution like XSB 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 on the basis of the residual program. However, the use of the residual program can led to unsound results: the stable models of the residual program are not necessarily stable models of the original program [Chen and Warren 1995]. For DATALOG programs which are allowed 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 [Niemelä and Simons 1996].

5.3. D-WFS

Implementation of D-WFS is one of the main goals of the *DisLoP project*, undertaken by the Artificial Intelligence Research Group at the University of Koblenz and headed by J. Dix and U. Furbach [Aravindan, Dix and Niemelä 1997*a*, Aravindan, Dix and Niemelä 1997*b*], see also `<http://www.uni-koblenz.de/ag-ki/DLP/>`. It aims at extending the *restart model elimination* and *hyper tableau calculi*, for disjunctive logic programming (see also Section 3).

We have already seen in Section 4 that the very definition of D-WFS gives a method to compute it. The problem is the inherent complexity of applying the GPPE. We already mentioned in Subsection 5.1 the exponential behaviour of GPPE even for normal programs. So we need other techniques for efficient implementations. Currently, an extension of D-WFS first-order programs is on its way [Dix and Stolzenburg 1998, Sakama and Seki 1997]. In this framework constraint techniques are heavily used. As this approach is beyond the scope of this chapter, we consider in the following an implementation of D-WFS using an efficient *tableau-technique* as described in Section 3. The underlying theory of this implementation is studied by [Brass, Dix, Niemelä and Przymusinski 1998, Brass, Dix, Niemelä and Przymusinski 2001].

The main idea is to find an equivalent characterization of D-WFS which is related to circumscription. When considering disjunctive logic programs, it turns out that it makes sense to look at *minimal entailment*meaning that *not* (ϕ) atoms are fixed and all positive atoms are minimized.

But we have to strengthen the notion of *models* for a disjunctive logic program P by requiring that a model I (a set of atoms and formulae *not* (ϕ)) of P has to be NAF-*consistent* for P, i.e., it must satisfy the following condition:

5.12. Definition (NAF-*Consistency).*
A model I for a disjunctive logic program P (viewed as a set of atoms and formulae *not* (ϕ)) is called NAF-*consistent* if the following holds

$$\text{If } P \models A_1 \vee \cdots \vee A_n, \text{ then for some } i \in \{1, \ldots, n\}, \mathit{not}\ (A_i) \notin I.$$

Here $\models$ denotes the standard propositional consequence relation where the program is seen as a set of clauses and the *not* (ϕ) atoms as new atoms. Hence, e.g., $\{p \leftarrow \mathit{not}\ q, p \leftarrow q\} \models p$ does not hold.

For example, $\{p, r, \mathit{not}\ p, \mathit{not}\ q\}$ is not a NAF-consistent model of

$$P = \{p \vee q \leftarrow, r \leftarrow \mathit{not}\ p \wedge \mathit{not}\ q\}$$

but $\{p, r, \mathit{not}\ p\}$ is. However, the latter is not a minimal model of P.

In the following we will be interested in the consequence relation given by *truth in all* NAF-*consistent minimal models.* We denote this relation by $\models_{\min(\mathrm{N})}$:

$$P \models_{\min(\mathrm{N})} \neg A \textit{ if and only if } \text{all NAF-consistent minimal models of } P \text{ satisfy } \neg A.$$

It turns out, that the following semantics $\mathrm{SEM}_{\mathrm{NAF}}$, defined by evaluating the truth of negative literals in NAF-*consistent* models coincides with D-WFS:

5.13. DEFINITION *(SEM_{NAF})*.
Let P be a disjunctive logic program over $\mathcal{L}$. We define a set $\mathrm{Dis}_{\min}(P)$ of pure disjunctions as follows

$$\mathrm{Dis}_{\min}(P) := \mathrm{Dis}^{+}(P) \cup \mathrm{Dis}^{-}(P) \text{ where}$$

$$\begin{array}{llllll} \mathrm{Dis}^{+}(\mathrm{P}) := & \{ & A_1 \vee \ldots \vee A_k & : P & \models & A_1 \vee \ldots \vee A_k \quad \} \\ \mathrm{Dis}^{-}(P) := & \{ & \neg A & : P & \models_{\min(\mathrm{N})} & \neg A \quad \} \end{array}$$

Here, the series D_i defined by $D_0 := \emptyset$, $D_{n+1} := \mathrm{Dis}(P/D_n)$ grows monotonically and eventually gets constant. We define $\mathrm{SEM}_{\mathrm{NAF}}(P)$ to be the limit of this series.

It remains to explain the notion P/D_n. Intuitively, we mean the program obtained from *prog* by reducing it using disjunctions in D_n. More precisely, P/Dis is the program obtained by doing the following reductions on P for all *not* C and $C_1 \vee \ldots \vee C_k$:

- if $\neg C \in \mathrm{Dis}$, then remove all occurrences of *not* C,
- if $C_1 \vee \ldots \vee C_k \in \mathrm{Dis}$ then remove all rules that contain $\{\mathit{not}\ C_1, \ldots, \mathit{not}\ C_k\}$ in their bodies.

P/Dis is obviously a slight generalization of the Gelfond-Lifschitz transformation. Our P/Dis still contains negative literals in general.

5.14. THEOREM. *D-WFS*$(P) = \mathrm{SEM}_{\mathrm{NAF}}(P)$.

Definition 5.13 can be employed to develop a novel implementation method for D-WFS which contrary to most other implementations of disjunctive semantics works in *polynomial space*. See `http://www.uni-koblenz.de/ag-ki/DLP` for an implementation of the method.

The idea is to exploit directly the new characterization of D-WFS in Theorem 5.14 and to implement D-WFS as an iterative reduction on disjunctive logic programs. The reduction starts with the original program P and leads to a reduced program P^* with the property that every query can be answered from this program with one call to a decision procedure of the minimal entailment $\models_{\min(\mathrm{N})}$. This can be seen as a compilation or (a partial evaluation) of the program leading to a smaller program from which all queries can be answered.

The reduced program is obtained as a fixed point of a reduction operator $\mathrm{R_D}(\cdot)$ mirroring the operator in $\mathrm{SEM}_{\mathrm{NAF}}(P)$.

$$\begin{aligned} \mathrm{R_D}(P) \quad = \quad & \{\mathcal{A} \leftarrow \mathcal{B}^{+} \wedge \mathit{not}\ \mathcal{C}' \mid \mathcal{A} \leftarrow \mathcal{B}^{+} \wedge \mathit{not}\ \mathcal{C} \in P, \\ & P/\mathrm{At}_{\mathcal{L}} \not\models C_1 \vee \cdots \vee C_n \text{ where} \\ & \mathcal{C} = \{C_1, \ldots, C_n\}, \\ & \mathcal{C}' = \{C \in \mathcal{C} \mid P \not\models_{\min(\mathrm{N})} \neg C\}\} \end{aligned}$$

Now let P^* be the limit of the (monotonically decreasing) series of programs $P_0, P_1, \ldots$ where $P_0 = P$ and $P_{i+1} = \mathrm{R_D}(P_i)$. Hence, the reduced program is obtained by

- removing each rule such that $C_1, \ldots, C_n$ are all the negative body literals and $C_1 \vee \cdots \vee C_n$ is a classical consequence of $P/\mathrm{At}_{\mathcal{L}}$ and
- removing all *not* (C) from the bodies of the remaining rules for which $\neg C$ is minimally entailed by the remaining program

until no reduction is possible. It can be established that $P_i = P/D_i$ for all $i = 0, 1, \ldots$ Thus we have:

5.15. THEOREM. *D-WFS*$(P) = \mathrm{SEM}_{\mathrm{NAF}}(P) = \mathrm{Dis}_{\min}(P^*)$.

Next we consider the problem of computing P^* within a polynomial space limit. Clearly, the required series of reductions can be implemented if decision procedures for classical entailment and minimal entailment $\models_{\min(\mathrm{N})}$ are available. If the decision procedures work in polynomial space, then the whole series can be implemented within a polynomial space limit as the size of the program decreases after each step. Classical entailment can be implemented in polynomial space and this is also possible (see [Brass et al. 1998]) for $\models_{\min(\mathrm{N})}$ by extending the tableau method for circumscription presented in [Niemelä 1996*b*, Niemelä 1996*a*].

5.16. THEOREM (Polynomial Space Complexity).
The entailment relation $\models_{\min(\mathrm{N})}$*, and therefore also D-WFS, can be implemented in polynomial space.*

5.4. D-STABLE

D-STABLE is closely related to Reiter's default logic and methods presented in Section 2.3 can be employed to implement D-STABLE. In fact D-STABLE can be seen as a subclass of default logic, for example, using the following mapping from logic programs to default theories proposed by Sakama and Inoue [Sakama and Inoue 1993]. 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 [Sakama and Inoue 1993], 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.

In the following we describe an implementation of D-STABLE for allowed DATALOG programs due to [Brass and Dix 1995][19]. The main idea is to generalize a result of Fages and Ben-Eliyahu to the non-disjunctive case (see [Apt and Bol 1994]): if there are no positive loops, the stable models coincide with the supported models. This is certainly the case for residual programs because they have no positive body literals at all. Furthermore, supported models correspond to Clark's completion [Clark 1978, Bidoit and Hull 1986], which is a classical first-order theory. Stable models can then be determined by

1. computing the residual program $res(P)$,
2. computing Clark's completion $comp(res(P))$ as defined in Definition 1.6,
3. computing a theory $comp^*(res(P))$ by doing hyperresolution on comp(res(P)) (see Definition 5.19),
4. computing the minimal models of $comp^*(res(P))$, which are exactly the stable models of the original theory) by model-generation (Theorem 5.20).

We note that the main complexity lies in steps 1. and 3. The computation of the residual program is in general *exponential* in the size of P. Often, however, an exponential blow up does not actually occur (it is the worst case) and so the method works for small sized programs quite well. Computing comp* in the third step is also exponential, because it corresponds to doing hyperresolution. Obviously, more optimization techniques have to be considered to improve our method.

Fortunately, steps 2. and 4. are cheap. Computing comp is linear in res(P) and computing the minimal models of comp* by model generation is also not exponential. Here we can use the structure of comp* and do not need general minimal model reasoning to get the minimal models.

In order to define Clark's completion for disjunctive rules, we simply view a rule like $p \vee q \leftarrow \mathit{not}\ r$ as a shorthand for the two rules $p \leftarrow \mathit{not}\ q \wedge \mathit{not}\ r$ and $q \leftarrow \mathit{not}\ p \wedge \mathit{not}\ r$. This idea is made precise in

5.17. Definition *(Completion of a Residual Program).*
Let res(P) be a program without positive body literals. Then we define the set $comp^+(res(P))$ to be

$$\bigl\{A_1 \vee \ldots \vee A_m \vee C_1 \vee \ldots \vee C_n \bigm| A_1 \vee \ldots \vee A_m \leftarrow \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \in res(P)\bigr\}$$

and $comp^-(res(P))$ to consist of the set

$$\bigl\{\neg A \vee \neg C_1 \vee \ldots \vee \neg C_n \bigm| \begin{array}{l} A \text{ is a ground atom}, C_i \in \mathcal{B}_i^- \cup \bigl(\mathcal{A}_i \setminus \{A\}\bigr), \\ \text{where } \mathcal{A}_i \leftarrow \mathit{not}\ \mathcal{B}_i^- \in res(P),\ i = 1, \ldots, n \\ \text{are all the conditional facts with } A \in \mathcal{A}_i \bigr\}. \end{array}$$

Finally, we set

$$comp(res(P)) := comp^+(res(P)) \cup comp^-(res(P)).$$

[19] `ftp://ftp.informatik.uni-hannover.de/software/index.html`

For instance, consider the following residual program res(P) and the completed definitions for p and q in the standard "if and only if"-notation:

$$res(P): \quad p \vee q \leftarrow \mathit{not}\ r. \qquad\qquad comp(res(P)): \quad p \leftrightarrow (\neg q \wedge \neg r) \vee (\neg s \wedge \neg t).$$
$$p \leftarrow \mathit{not}\ s \wedge \mathit{not}\ t. \qquad\qquad q \leftrightarrow (\neg p \wedge \neg r).$$

The direction $\leftarrow$ corresponds to the logical information in the given program, and is represented by the positive disjunctions above. The negative disjunctions are the clausal ("multiplied out") form of the direction $\rightarrow$.

Now this completion of the residual program characterizes the stable models:

5.18. Lemma. *A two-valued Herbrand interpretation I is a stable model of* P if and only if $I \models comp(res(P))$*, where res(P) is the residual program of P.*

So an answer θ to a query ψ is (sceptically) correct wrt the stable model semantics *if and only if* $comp(res(P)) \vdash \psi\theta$. Any theorem prover can be used to check that (or generate answers during the proof of ψ). It is also possible to use standard model generating algorithms on $comp(res(P))$ to construct the stable models of P.

However, we can also make use of the very special structure of $comp(res(P))$. By applying the "hyperresolution" [Chang and Lee 1973] technique, we can compute the set $comp^*(res(P))$ of all positive consequences of $comp(res(P))$.

5.19. Definition *(Implied Positive Disjunctions).*
Hyperresolution with rules from $comp^-(res(P))$ is defined as follows:

$$H(\mathcal{D}) := \Big\{ \bigcup_{i=1}^{m} (\mathcal{A}_i \setminus \{A_i\}) \;\Big|\; \mathcal{A}_i \in \mathcal{D},\ A_i \in \mathcal{A}_i,\ \neg A_1 \vee \ldots \vee \neg A_m \in comp^-(res(P)) \Big\}.$$

I.e. for every purely negative clause $\neg A_1 \vee \ldots \vee \neg A_m$ of $comp^-(res(P))$, we look at all positive clauses $head_i \in \mathcal{D}$ that contain one of the A_i in their heads. Then we resolve these clauses by getting rid of all the A_i and putting the remaining parts $\mathcal{A}_i \setminus \{A_i\}$ of the together as one clause: the resolvent.

We start with $\mathcal{D}_0 := comp^+(res(P))$ and iterate $\mathcal{D}_i := H(\mathcal{D}_{i-1})$ until a fixpoint $\mathcal{D}_n$ is reached. Then let

$$comp^*(res(P)) := \{\mathcal{A} \in \mathcal{D}_n \mid \text{for all } \mathcal{A}' \subset \mathcal{A}\text{: } \mathcal{A}' \notin \mathcal{D}_n\}.$$

In the above computation the negative disjunctions $\neg A_1 \vee \wedge \cdots \wedge \vee \neg A_n$ are interpreted as rules $\leftarrow A_1 \wedge \cdots \wedge A_n$. The body atoms are matched with the positive disjunctions. The result of the rule application consists of the disjunctive contexts. Note that it is possible that the empty disjunction is computed, which is interpreted as logically false. This happens if the completion is inconsistent, e.g. for the program $p \leftarrow \mathit{not}\ p$.

Here is an example. Consider the following residual program $res(P)$ and its completion $comp(res(P))$ written as facts and rules:

$$
\begin{array}{llllll}
res(P): & p \vee q. & comp^{+}(res(P)): & p \vee q. & comp^{-}(res(P)): & \leftarrow p \wedge q. \\
& r \leftarrow not\ p. & & r \vee p. & & \leftarrow q \wedge p. \\
& r \leftarrow not\ q. & & r \vee q. & & \leftarrow r \wedge p \wedge q. \\
& s \leftarrow not\ r. & & s \vee r. & & \leftarrow s \wedge r.
\end{array}
$$

The second and the third rule of $comp^{-}(res(P))$ are of course superfluous. We can derive r (with the first rule applied to the second and third disjunctive fact), and after deleting non-minimal disjunctions we get

$$comp^{*}(res(P)) = \{p \vee q,\ r\}.$$

This directly describes the two stable models $I_1 := \{p, r\}$ and $I_2 := \{q, r\}$. In general, we have the following:

5.20. THEOREM (Reduction to $\mathbf{comp}(\mathbf{res}(\mathbf{P}))$).
I is a stable model of P if and only if I is a minimal model of $comp^{*}(res(P))$.

We now consider the computation of the stable models themselves. In the special case that there is only a single stable model, $comp^{*}(res(P))$ is the set of atoms true in it, so no further computation is needed. If there are multiple models, we construct them by a case analysis. Note that any atom contained in a proper disjunction of $comp^{*}(res(P))$ can really be true or false (of course, the decision on different atoms is not independent). So we also need no further computation if we are only interested in credulous truth of ground literals. In the general case, we choose a truth value for a ground atom, and then compute the implied positive disjunctions which follow from this new information. We repeat this process until no proper disjunctions remain:

procedure $model\_generation\big(comp^{+}(res(P)),\ comp^{-}(res(P))\big)$:
  compute $comp^{*}(res(P)))$;
  if $comp^{*}(res(P)))$ contains no proper disjunction **then**
    print $comp^{*}(res(P)))$;
  else
    let A be contained in a proper disjunction;
    $model\_generation\big(comp^{+}(res(P)) \cup \{A\},\ comp^{-}(res(P))\big)$;
    $model\_generation\big(comp^{+}(res(P)),\ comp^{-}(res(P)) \cup \{\neg A\}\big)$;

Note that by the completeness of hyperresolution, we can be sure that if we choose A to be true or false, there is really a stable model in which A has this truth value. So our algorithm never runs into "dead ends". This is an important difference to the algorithm used in [Chen and Warren 1995] for propagating guessed truth values of atoms. For instance, the following residual program has only a single stable model (namely $\{p, \neg q, r\}$), but it is reduced in the sense of [Chen and Warren 1995], so their algorithm has to guess a truth value of some atom:

$$\begin{array}{rcl} p & \leftarrow & \mathit{not}\ p. \\ p & \leftarrow & \mathit{not}\ q. \\ q & \leftarrow & \mathit{not}\ r. \\ r & \leftarrow & \mathit{not}\ q. \end{array}$$

Of course, this example is quite pathological, and their reduction algorithm works well in the more usual simpler cases. In particular, their algorithm does not need to compute the residual program first. However, it is not clear how to generalize their method to the disjunctive case.

We end this section with pointing to a very recent approach relying on complexity-theoretic considerations. `dlv` [Eiter, Leone, Mateis, Pfeifer and Scarcello 1997, Eiter, Leone, Mateis, Pfeifer and Scarcello 1998] is a knowledge representation system, based on disjunctive logic programming, which offers front-ends to several advanced KR formalisms (developed since the end of 1996).

`dlv` is based on solid theoretical foundations [Leone, Rullo and Scarcello 1997], and recent comparisons [Eiter et al. 1998] are encouraging. Major emphasis has been put on advanced knowledge modelling features. The kernel language, which extends disjunctive logic programming by true negation and integrity constraints, allows for representing complex knowledge based problems in a highly declarative fashion [Eiter et al. 1998].

The computational kernel of `dlv` is an efficient engine for computing the stable models of a disjunctive logic program, based on algorithms presented in [Leone et al. 1997], together with various improvements by suitable optimization techniques.

The system runs in *polynomial space* and *single exponential time*, and is able to efficiently recognize and process syntactical subclasses of disjunctive logic programs which have lower computational complexity than the general case (like, e.g., programs with head-cycle free disjunction or stratified negation). Modular evaluation techniques are currently under development along with linguistic extensions to deal also with quantitative information [Buccafurri, Leone and Rullo 1997].

The three main modules of the `dlv` kernel are the

1. *Intelligent Grounding*,
2. the *Model Generator* and
3. the *Model Checker*.

All of these modules perform a modular evaluation of their input according to various dependency graphs as defined in [Leone et al. 1997, Eiter, Leone, Mateis, Pfeifer and Scarcello 1997] and try to detect and efficiently handle special (syntactic) subclasses, which in general yields a tremendous speedup.

The Intelligent Grounding takes an input program, whose facts can be stored also in the tables of external relational databases, and efficiently generates a subset of the program instantiation that has exactly the same stable models as the full program, but is much smaller in general. (For stratified programs, for example, the Grounding already computes the single stable model.)

Then the Model Generator is run on the (ground) output of the Intelligent Grounding. It generates one candidate for a stable model at a time and invokes the Model Checker to verify whether it is indeed a stable model.

The Model Generator first computes the least fixpoint $W_P^\infty(\emptyset)$ of a deterministic operator W_P [Leone et al. 1997], which extends to the disjunctive case the well-founded operator of Van Gelder, Ross and Schlipf. If $W_P^\infty(\emptyset)$ is a total interpretation, then it is the unique stable model [Leone et al. 1997], which can be returned to the user without any further check. Otherwise, $W_P^\infty(\emptyset)$ is contained in every stable model [Leone et al. 1997] and is therefore the deterministic basis for the computation of the stable semantics. To move beyond $W_P^\infty(\emptyset)$, a literal is then added to $W_P^\infty(\emptyset)$ and its deterministic consequences are computed; the process is repeated until (i) either a contradiction arises, or (ii) a total interpretation is reached (actually, a total interpretation is not extensionally generated; rather, a sufficient logical condition guarantees totality). A contradiction causes backtracking and removal of the last chosen literal; while reaching a total interpretation implies a call to the Model Checker which verifies whether the generated interpretation is a stable model or not. Thus, the Model Generator implements a backtracking algorithm that generates all possible "candidate" stable models. To improve the efficiency of the backtracking algorithm, `dlv` extends the concept of *possibly-true literals* [Leone et al. 1997, Leone and Rullo 1992, Leone, Romeo, Rullo and Saccà 1993, Buccafurri, Leone and Rullo 1996] to the disjunctive case.

Candidate stable models produced by the Model Generator are checked for stability by the Model Checker of `dlv`. Verifying the stability condition is a difficult task in general, as it is a well-known co-**NP**-hard problem (i.e., basically, as hard as computing the stable models of a non-disjunctive program). Instead of rewriting the program by the Gelfond-Lifschitz transformation and then checking the minimality of the candidate stable model on the rewritten program, the Model Checker verifies the stability directly, by checking whether the candidate stable model is unfounded-free (this condition is equivalent to stability checking [Leone et al. 1997]). This check is performed in polynomial time for head-cycle free programs (i.e., programs where recursion does not pass *through* disjunction), and for general programs the inefficient part of the computation is limited to only the components of the program that are not head-cycle free.

Thus, thanks to the efficient strategy implemented by the Model Checker, on a large and meaningful class of disjunctive programs `dlv` obtains basically the same performances that are obtainable on or-free programs (note that the semantics of programs with recursion through disjunction is often very hard to understand).

6. Benchmarks

In this section we address the problem of devising benchmarks for nonmonotonic reasoning techniques. We are especially interested in families of test cases with increasing size and complexity because they provide a systematic way for analyzing experimentally effects of different design decisions and for comparing the performance, scalability and limits of different implementations.

The problem of providing these kinds of families of benchmarks has been addressed only recently in the literature. In earlier papers benchmarks are usually examples illustrating important aspects of nonmonotonic reasoning (see, e.g., [Lifschitz 1989]). Typically these examples are quite small and directed to evaluating the correctness of an implementation rather than its performance. One of the first examples of a scalable collection of test cases is presented in [Subrahmanian et al. 1995] where Van Gelder's win-move example is used with randomly generated game graphs of increasing size.

In this section we discuss families of benchmarks arising from combinatorial graph problems, propositional satisfiability, planning, and diagnosis. Most of the benchmarks are given in terms of normal logic programs with the stable model semantics. However, the benchmarks can be employed as test cases for systems implementing other formalizations of nonmonotonic reasoning by representing the benchmark problems in the formalism of interest or by utilizing a direct translation from logic programs with the stable model semantics to the formalism of interest.

6.1. Graph Problems

The idea of producing benchmarks from graphs provides an interesting approach to generating families of test cases with similar structure but increasing size and complexity. Furthermore, a large variety of different types of test cases are available, and test cases can be grounded to realistic data by choosing interesting graphs representing real-life information.

This approach of building benchmarks from graphs forms the basis of the `Theory Base` system [Cholewiński, Marek, Mikitiuk and Truszczyński 1995] that generates test default theories and logic programs. The idea is to generate a default theory or a logic program from a given graph and a graph problem in such a way that the extensions of the default theory or the stable models of the program corresponds to the solutions of the original graph problem for the given graph. For example, given a graph and the 3-colorability problem a logic program is generate such that the stable models of the program provide the 3-colorings of the graph. This approach is employed also by [Niemelä and Simons 1996]. Further information about interesting classes of graphs, graph problems and mappings to nonmonotonic formalisms can be found in [Cholewiński et al. 1995, Niemelä and Simons 1996].

We illustrate the approach by presenting mappings from the 3-colorability and Hamiltonian circuit problems to normal logic programs with the stable model semantics. First, we consider the 3-colorability problem, i.e., the problem of assigning each vertex of a given graph one of three colors such that if two vertices are connected with an arc, they do not have the same color. Given a graph we construct a logic program as follows: (i) a set of atomic facts of the form $vertex(v)$ and $arc(v, u)$ corresponding to the vertices and arcs of the graph is taken and (ii) the following rules are included

$$color(V, blue) \leftarrow vertex(V), not\ color(V, green), not\ color(V, red)$$

$$color(V, green) \leftarrow vertex(V), not\ color(V, blue), not\ color(V, red)$$

$$color(V, red) \leftarrow vertex(V), not\ color(V, green), not\ color(V, blue)$$

$$noncoloring \leftarrow arc(V, U), color(V, C), color(U, C)$$

$$p \leftarrow not\ p, noncoloring$$

Now the program has a stable model if and only if there is a 3-coloring of the graph. In fact, the coloring can be read from the stable model directly by taking the facts of the form $color(v, c)$ true in the model. For systems supporting query-evaluation in the form of finding stable models where a given query holds, benchmarks are obtained by removing the last rule. Then there is a 3-coloring of the graph if and only if the resulting program has a stable model not containing *noncoloring*.

The 3-colorability problem does not seem to require the enhanced expressivity of nonmonotonic formalisms and can be mapped straightforwardly to, e.g., propositional satisfiability. As an example where the expressivity of nonmonotonic logics provides compact representation we consider the Hamiltonian circuit problem, i.e., the problem of finding a path that visits each vertex of a given graph exactly once and returns to the starting vertex. The corresponding logic program is constructed as follows: (i) a set of atomic facts of the form $vertex(v)$ and $arc(v, u)$ corresponding to the vertices and arcs is taken, (ii) the following rules are included

$$hcircuit(V_1, V_2) \leftarrow arc(V_1, V_2), not\ otherroute(V_1, V_2)$$

$$otherroute(V_1, V_2) \leftarrow arc(V_1, V_2), hcircuit(V_1, V_3), not\ (V_2 = V_3)$$

$$otherroute(V_1, V_2) \leftarrow arc(V_1, V_2), hcircuit(V_3, V_2), not\ (V_1 = V_3)$$

$$reached(V_2) \leftarrow hcircuit(V_1, V_2), reached(V_1)$$

$$reached(V_2) \leftarrow hcircuit(V_1, V_2), initialnode(V_1)$$

$$noncircuit \leftarrow vertex(V), not\ reached(V)$$

$$p \leftarrow not\ p, noncircuit$$

and (iii) one of the vertices v is taken as the starting vertex ($initialnode(v)$ is added). The resulting program has a stable model if and only if the graph has a Hamiltonian circuit. Again, a stable model of the program provides directly a Hamiltonian circuit by the facts of the form $hcircuit(v, u)$ true in the model. A benchmark for query-evaluation can be obtained by removing the last rule. Then the graph has a Hamiltonian circuit if and only if the resulting program has a stable model not containing *noncircuit*.

6.2. Propositional Satisfiability

Propositional satisfiability problems can be employed as benchmarks for nonmonotonic reasoning systems in a similar way as the graph problems, i.e., by mapping sets of propositional formulae to nonmonotonic theories. Although general nonmonotonic logics contain classical logic, propositional satisfiability problems can provide interesting benchmarks for the nonmonotonic aspects of the logics, too. The idea is that propositional satisfiability is captured using the nonmonotonic primitives of the formalism rather than directly in terms of classical logical operators that are included in the formalism. In this way very challenging test cases for evaluating the implementation of the nonmonotonic primitives are obtained. For example, hard random satisfiability problems from the phase transition region [Crawford and Auton 1996] could be employed and results compared with the best satisfiability checkers. This approach has been used in [Niemelä and Simons 1997].

Here we demonstrate the idea by presenting a mapping from propositional satisfiability to the problem of finding a stable model. The mapping is devised in a way that very little classical reasoning is required and propositional satisfiability is reduced to the problem of handling the nonmonotonic operator *not* .

For a set of clauses S, we build a logic program as follows:

1. we introduce for each atom a appearing in S two atoms a and $\hat{a}$ and include two rules

$$\hat{a} \leftarrow \mathit{not}\ a$$
$$a \leftarrow \mathit{not}\ \hat{a}$$

2. for each clause in S, we introduce a new atom c and include one rule for each literal l in the clause as follows: if l is a positive atom a, take the rule $c \leftarrow a$ and if l is the negation of an atom a, add $c \leftarrow \hat{a}$ and
3. finally we include the rules

$$\mathit{satisfied} \leftarrow c_1, \ldots, c_n$$
$$p \leftarrow \mathit{not}\ p, \mathit{not}\ \mathit{satisfied}$$

   where $c_1, \ldots, c_n$ are all the atoms introduced for the clauses in S.

Then the set of clauses S has a model if and only if the resulting program has a stable model. The propositional model is obtained from the stable model by assigning true for atoms true in the stable model and assigning false to the rest of the atoms (for atoms a for which $\hat{a}$ is true in the stable model). For query-evaluation a benchmark is obtained by removing the last rule. Then the set of clauses S has a model if and only if the resulting program has a stable model containing the atom *satisfied*.

6.3. Planning

Planning provides particularly interesting benchmarks for nonmonotonic reasoning systems because this is a domain from which some of the main motivation for developing nonmonotonic formalisms originates. In planning difficult issues related to reasoning about action and change such as the frame problem have to be addressed and the expressivity of the nonmonotonic formalisms can be utilized to overcome some of the difficulties. However, a detailed account of using nonmonotonic formalisms in planning is beyond the scope of this article. Instead, we illustrate with an example of blocks world planning how planning problems can be mapped to nonmonotonic reasoning problems. For more detailed accounts, we refer the reader, e.g., to [Gelfond and Lifschitz 1993, Dimopoulos, Nebel and Koehler 1997].

In the blocks world we are given initial conditions of blocks on a table stating how they are stacked on top of each other, similar goal conditions and a robot that is able to move blocks. The aim is to generate a plan for the robot to move blocks from the initial configuration so that the goal conditions are satisfied. Families of benchmarks are obtained by increasing the number of blocks and the complexity of the initial and goal conditions.

Consider the following example. In the initial configuration we have three blocks a, b, c such that b and c are on the table and a is on top of b. The goal conditions are that c is on a and b is on c. A solution for this planning problem is a sequence of moves where a is moved on the table, c is moved on a and finally b is moved on c.

The idea is to map a planning problem to a logic program such that stable models correspond to valid plans. For formalizing blocks world planning we use situations where facts hold. We assume that we have situations $t_0, \ldots, t_n$ where t_0 is the initial situation and a predicate *nextstate* specifying the order of the situations, i.e., for which $nextstate(t_{i+1}, t_i)$ for $i = 0, \ldots, n-1$ holds. We employ predicates $on(X, Y, T)$ (X is on Y in the situation T) and $moveop(X, Y, T)$ (X is moved on Y in the situation T) and assume that the available blocks are specified using facts of the form $block(b)$.

The initial conditions are straightforward to formalize. For instance, for the example above it is sufficient to include the facts

$$
\begin{aligned}
&on(a, b, t_0)\\
&on(b, table, t_0)\\
&on(c, table, t_0)
\end{aligned}
$$

In order to capture the goal conditions we employ a predicate *goal* which holds in any situation where the goal conditions have been reached and in all subsequent situations. For the example above, the resulting rules are

$$
\begin{aligned}
&goal(T) \leftarrow on(b, c, T), on(c, a, T), on(a, table, T)\\
&goal(T_2) \leftarrow nextstate(T_2, T_1), goal(T_1)\\
&goal \leftarrow goal(T)\\
&p \leftarrow not\ p, not\ goal
\end{aligned}
$$

In order to formalize the preconditions and effects of the move operator we use the following rules.

$$\begin{aligned} moveop(X,Y,T) \leftarrow\ & moveable(X), block(Y), not\ (X = Y), \\ & on\_something(X,T), \\ & not\ covered(X,T), not\ covered(Y,T), \\ & not\ blocked\_move(X,Y,T) \end{aligned}$$

$$on(X,Y,T_2) \leftarrow nextstate(T_2,T_1), moveop(X,Y,T_1)$$

$$on\_something(X,T) \leftarrow on(X,Z,T)$$

$$covered(X,T) \leftarrow moveable(X), on(Z,X,T)$$

$$moveable(X) \leftarrow block(X), not\ (X = table)$$

It is enough to provide a frame axiom only for the predicate *on*.

$$on(X,Y,T_2) \leftarrow nextstate(T_2,T_1), on(X,Y,T_1), not\ moving(X,T_1)$$

$$moving(X,T) \leftarrow moveop(X,Y,T)$$

What remains to be stated are the blocking conditions for the moves. The first set of conditions covers the cases where the goal has been reached or the instance of the move operator has not been chosen.

$$blocked\_move(X,Y,T) \leftarrow moveable(X), block(Y), goal(T)$$

$$blocked\_move(X,Y,T) \leftarrow on\_something(X,T), block(Y), not\ moveop(X,Y,T)$$

The second set depends on whether concurrency is allowed, i.e., whether more than one operator can be applied in a situation. We allow this and block only the operator instances whose effects are in conflict.

$$blocked\_move(X,Y,T) \leftarrow moveable(X), block(Y),$$

$$moveop(X,Z,T), not\ (Y = Z)$$

$$blocked\_move(X,Y,T) \leftarrow moveable(X), moving(Y,T)$$

$$\begin{aligned} blocked\_move(X,Y,T) \leftarrow\ & moveable(X), block(Y), not\ (Y = table), \\ & moveop(Z,Y,T), not\ (X = Z) \end{aligned}$$

For a program constructed like this it holds that the program has a stable model if and only if there is a sequence of moves from the initial configuration to a situation satisfying the goal conditions that can be executed concurrently in at most n steps. For query-evaluation a benchmark is obtained by removing the rule with p in the head. Then there is plan if and only if the resulting program has a stable model containing *goal*.

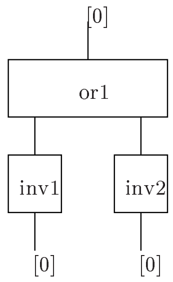


Figure 9: A Circuit

6.4. Diagnosis Problems

Model-based diagnosis of technical systems provides an interesting class of benchmark problems for two reasons. Firstly, the examples which stem from hardware design usually contain a large number of clauses, which have to be handled efficiently in order to solve realistic problems. Secondly, the diagnosis task involves minimal model reasoning by definition, and hence techniques from nonmonotonic reasoning have to be applied.

According to [Reiter 1987] a simulation model of the technical device under consideration is constructed and is used to predict its normal behavior. By comparing this prediction with the actual behavior it is possible to derive a diagnosis. This approach uses a logical description of the device, called the system description (SD), formalized by a set of first–order formulas. The system description consists of a set of axioms characterizing the behavior of system components of certain types. The topology is modeled separately by a set of facts. The diagnostic problem is described by system description SD, a set $COMP$ of components and a set OBS of observations (logical facts). With each component we associate a behavioral mode: $Mode(c, Ok)$ means that component c is behaving correctly, while $Mode(c, Ab)$ (abbreviated by $Ab(c)$) denotes that c is faulty.

A *Diagnosis* of $(SD, COMP, OBS)$ is a set $\Delta \subseteq COMP$, such that

$$SD \cup OBS \cup \{Mode(c, Ab) | c \in \Delta\} \cup \{\neg Mode(c, Ab) | c \in COMP - \Delta\}$$

is consistent. Δ is called a *Minimal Diagnosis*, *if and only if* it is the minimal set (wrt. $\subseteq$) with this property.

The set of all minimal diagnoses can be large for complex technical devices. Therefore, stronger criteria than minimality are often used to further discriminate among the minimal diagnoses.

These criteria are usually based on the probability or cardinality of diagnoses. In the remainder of this section we will use restrictions on the cardinality of diagnoses. We say that a diagnosis satisfies the n*-fault assumption if and only if* $|\Delta| \leq n$.

As an example consider the simple digital circuit in Figure 9 consisting of an or–gate ($or1$) and two inverters $inv1$ and $inv2$. The system description SD is given by the following propositional clauses:[20].

$$\begin{array}{lrl}
\text{OR1:} & ab(or1), high(or1,i1), high(or1,i2) & \leftarrow high(or1,o) \\
& ab(or1), high(or1,o) & \leftarrow high(or1,i1) \\
& ab(or1), high(or1,o) & \leftarrow high(or1,i2)
\end{array}$$

$$\begin{array}{ll}
\text{INV1:} & ab(inv1) \leftarrow high(inv1,o), high(inv1,i) \\
& ab(inv1), high(inv1,i), high(inv1,o) \leftarrow
\end{array}$$

$$\begin{array}{ll}
\text{INV2:} & ab(inv2) \leftarrow high(inv2,o), high(inv2,i) \\
& ab(inv2), high(inv2,i), high(inv2,o) \leftarrow
\end{array}$$

$$\begin{array}{rlrl}
high(or1,i1) & \leftarrow high(inv1,o) & high(or1,i2) & \leftarrow high(inv2,o) \\
high(inv1,o) & \leftarrow high(or1,i1) & high(inv2,o) & \leftarrow high(or1,i2)
\end{array}$$

We observe that both inputs of the circuit have low voltage and the output also has low voltage, i.e. the clause set of OBS is given by $\{\leftarrow high(inv1,i), \leftarrow high(inv2,i), \leftarrow high(or1,o)\}$.

The expected behavior of the circuit given that both inputs are low would be high voltage at the outputs of both inverters and consequently high voltage at the output of the or–gate. This model of the correctly functioning device, namely

$$I_0 = \{high(inv1,o), high(inv2,o), high(or1,i1), high(or1,i2), high(or1,o)\},$$

can be computed very efficiently even for large devices by domain–specific tools, e.g. circuit simulators.

[Baumgartner, Fröhlich, Furbach and Nejdl 1997*a*] show how the diagnosis task can be efficiently solved by a general first-order theorem prover, based on model generation. The proof system used therein is based on the hyper tableau calculus presented above. In this approach a minimal diagnosis is a computed model of a clause set, as given in our example. However, it turned out, that for examples from the ISCAS-85 benchmarks [Isc 1985], which is a standard from the diagnosis literature, one needs additional domain dependent pruning techniques of the search space. This can be achieved by taking into account an initial model of a correctly functioning device I_0, as it is called in the above example. This set of atoms can be used in a compilation step to transform the given clause sets for the diagnosis task, such that only deviations from this initial model are computed. It is shown in [Baumgartner, Fröhlich, Furbach and Nejdl 1997*b*], that these deviations correspond to the minimal diagnosis. In Figure 10 some experiments from the benchmark suites are reported.

[20] These formulas can be obtained by instantiating a first-order description of the gate functions with the structural information. For instance, the clauses for the OR1 gate stem from the formula $\forall OR\_gate : (\neg ab(OR\_gate) \rightarrow (high(OR\_gate,o) \leftrightarrow (high(OR\_gate,i1) \vee high(OR\_gate,i2))))$

Name	*# Gates*	*# Clauses*	*# Diagnosis*	*Time (sec.)*	*# Steps*	*All?*
C499	202	1685	2	5	3015	no
2-fault:			67	50	27323	yes
C880	383	2776	19	2	161	yes
C1355	546	3839	5	47	24699	no
2-fault:			5	2948	1284454	no
C2670	1193	8260	31	6	533	yes
C3540	1669	10658	3	10853	1473572	yes
C5315	2307	16122	5	13	3071	yes

Figure 10: ISCAS'85 Circuits and runtime results.

7. Conclusion

Reasoning in everyday life is quite different from the kind of reasoning considered in classical formal logic. While the latter has been investigated exhaustively in this century, the former has been brought to attention in the early research on artificial intelligence. In many areas (like planning or diagnosis) we are faced up with problems related to *uncertainty* about the current situation: we do not have complete knowledge, neither about the facts that are true nor about the rules describing how the world evolves. Many competing methods arose out of this problem: *fuzzy logic*, *probabilistic reasoning*, *case-based reasoning*, *qualitative reasoning*, *spatial reasoning* etc. Most, if not all of them, have an important nonmonotonic component. Recently there has been growing interest in *Multi-Agent Systems*. A particular system, based on logic programming and incorporating the above mentioned methods is described in [Subrahmanian, Bonatti, Dix, Eiter, Kraus, Özcan and Ross 2000] (see also [Dix, Subrahmanian and Pick 2000, Eiter and Subrahmanian 1999, Eiter, Subrahmanian and Pick 1999, Dix, Nanni and Subrahmanian 2000, Dix, Kraus and Subrahmanian 2001]).

We tried in this chapter to provide the reader with a concise yet understandable description of the state-of-the-art in nonmonotonic reasoning, with special attention to implementations. While we left out many of the numerous theoretical variants of classical frameworks, we hope not to have forgotten any of the implemented systems that are currently in use. We again refer to the following webpage which is and will be actively maintained and contains information on various logic programming systems that concentrate on non-monotonic aspects: `<URL:http://www.uni-koblenz.de/ag-ki/LP/>`.

The future of nonmonotonic reasoning (see [Dix 1995*b*, Dix 1998] will heavily depend on the applications such systems can successfully cope with. With applications, we mean *real-world* applications and not the well-known blocksworlds examples or the good old Tweety.

In our section on *Benchmarks* we tried to point to some interesting areas. They all seem to contain promising problems for nonmonotonic systems. Of course, much more applications are needed and future work will show if nonmonotonic reasoning can offer valuable tools to deal with them.

We end this article with giving some pointers to actively maintained web-pages providing the interested reader with more information about nonmonotonic reasoning and nonmonotonic aspects of logic programming (see also the recent [Dix and Lobo 1999]):

General Sites:

- **NMR:** `<URL:http://www.medg.lcs.mit.edu/nm/`,
- **LP:** `<URL:http://www.cwi.nl/projects/alp/index.html`,
- **LPNMR:** `<URL:http://www.cs.engr.uky.edu/~lpnmr/`,
- **CLN:** `<URL:http://www.compulog.org/`,
  `<URL:http://www.cs.nmsu.edu/~complog/`,

Conferences:

- **NMR 00:** `<URL:http://www.cs.engr.uky.edu/nmr2000>`,
- **ICLP 99:** `<URL:http://www.cs.kuleuven.ac.be/~iclp99>`,
- **LPNMR 99:** `<URL:http://www.dbai.tuwien.ac.at/lpnmr99/>`,
- **CL 00:** `<URL:http://www.doc.ic.ac.uk/cl2000/>`.

Acknowledgments

Many people contributed to this article. Most of them indirectly (via many coauthored papers or books) others more directly, by giving us suggestions on how to modify and strengthen this paper. We owe special thanks to Gerd Brewka and Mirek Truszczyński for a careful reading of drafts of this article. Nevertheless, all remaining typos and mistakes are under our sole responsibility.

Bibliography

APT K., BLAIR H. AND WALKER A. [1988], Towards a theory of declarative knowledge, *in* J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufmann Publishers, Los Altos, pp. 89–148.

APT K. R. AND BOL R. N. [1994], 'Logic Programming and Negation: A Survey', *Journal of Logic Programming* **19-20**, 9–71.

APT K. R. AND DOETS H. [1994], 'A new definition of SLDNF-resolution', *Journal of Logic Programming* **18**(2), 177–190.

ARAVINDAN C. [1996], An abductive framework for negation in disjunctive logic programming, *in* 'Proc. JELIA 96', number 1126 *in* 'LNAI', European Workshop on Logic in AI, Springer.

ARAVINDAN C., DIX J. AND NIEMELÄ I. [1997*a*], 'Dislop: A research project on disjunctive logic programming', *AI Communications* **10**(3/4), 151–165.

ARAVINDAN C., DIX J. AND NIEMELÄ I. [1997*b*], DisLoP: Towards a Disjunctive Logic Programming System, *in* J. Dix, U. Furbach and A. Nerode, eds, 'Logic Programming and Non-Monotonic Reasoning, Proceedings of the Fourth International Conference', LNAI 1265, Springer, Berlin, pp. 342–353.

BAADER F. AND HOLLUNDER B. [1995], 'Embedding defaults into terminological knowledge representation formalisms', *Journal of Automated Reasoning* **14**, 149–180.

BAKER A. B. [1991], 'Nonmonotonic reasoning in the framework of situation calculus', *Artificial Intelligence* **49**, 5–23.

BALCÁZAR J., DÍAZ I. AND GABARRÓ J. [1988], *Structural Complexity I*, Springer-Verlag, Berlin.

BARAL C. AND GELFOND M. [1994], 'Logic Programming and Knowlege Representation', *Journal of Logic Programming* **19-20**, 73–148.

BAUMGARTNER P. [1998], Hyper Tableaux — The Next Generation, *in* H. de Swaart, ed., 'Automated Reasoning with Analytic Tableaux and Related Methods', number 1397 *in* 'Lecture Notes in Artificial Intelligence', Springer.

BAUMGARTNER P., FRÖHLICH P., FURBACH U. AND NEJDL W. [1997*a*], Semantically Guided Theorem Proving for Diagnosis Applications, *in* '15th International Joint Conference on Artificial Intelligence (IJCAI 97)', pp. 460–465.

BAUMGARTNER P., FRÖHLICH P., FURBACH U. AND NEJDL W. [1997*b*], Semantically Guided Theorem Proving for Diagnosis Applications, *in* '15th International Joint Conference on Artificial Intelligence (IJCAI 97)', International Joint Conference on Artificial Intelligence, Nagoya, pp. 460–465.

BAUMGARTNER P. AND FURBACH U. [1994*a*], 'Model Elimination without Contrapositives and its Application to PTTP', *Journal of Automated Reasoning* **13**, 339–359. Short version in: Proceedings of CADE-12, Springer LNAI 814, 1994, pp 87–101.

BAUMGARTNER P. AND FURBACH U. [1994*b*], PROTEIN: A *PRO*ver with a *T*heory *E*xtension *I*nterface, *in* A. Bundy, ed., 'Automated Deduction – CADE-12', Vol. 814 of *Lecture Notes in Artificial Intelligence*, Springer, pp. 769–773.

BAUMGARTNER P. AND FURBACH U. [1997], Calculi for Disjunctive Logic Programming, *in* J. Maluszynski, ed., 'Logic Programming - Proceedings of the 1997 International Symposium', The MIT Press, Port Jefferson, New York.
URL: *http://www.uni-koblenz.de/ peter/Publications/ilps97-proceedings.ps.gz*

BAUMGARTNER P., FURBACH U. AND NIEMELÄ I. [1996], Hyper Tableaux, *in* 'Proc. JELIA 96', number 1126 *in* 'LNAI', European Workshop on Logic in AI, Springer. (Long version in: *Fachberichte Informatik*, 8–96, Universität Koblenz-Landau).

BAUMGARTNER P., FURBACH U. AND STOLZENBURG F. [1997*a*], 'Computing Answers with Model Elimination', *Artificial Intelligence* **90**(1–2), 135–176.

BAUMGARTNER P., FURBACH U. AND STOLZENBURG F. [1997*b*], Model Elimination, Logic Programming and Computing Answers, *in* '14th International Joint Conference on Artificial Intelligence (IJCAI 95)', Vol. 1, International Joint Conference on Artificial Intelligence, Montreal, pp. 335–340.

BELL C., NERODE A., NG R. AND SUBRAHMANIAN V. [1994], 'Mixed integer programming methods for computing nonmonotonic deductive databases', *Journal of the Association for Computing Machinery* **41**(6), 1178–1215.

BEN-ELIYAHU R. AND DECHTER R. [1991], Default logic, propositional logic and constraints, *in* 'Proceedings of the 9th National Conference on Artificial Intelligence', The MIT Press, pp. 370–385.

BEN-ELIYAHU R. AND DECHTER R. [1992], Propositional semantics for default logic, *in* 'Working Notes of the 4th International Workshop on Nonmonotonic Reasoning', Plymouth, Vermont, USA, pp. 13–27.

BERMAN K. A., SCHLIPF J. S. AND FRANCO J. V. [1995], Computing the Well-Founded Semantics Faster, *in* A. Nerode, W. Marek and M. Truszczyński, eds, 'Logic Programming and Non-Monotonic Reasoning, Proceedings of the Third International Conference', LNCS 928, Springer, Berlin, pp. 113–126.

BIBEL, W. AND SCHMITT, P. H., EDS [1998], *Automated Deduction — A Basis for Applications*, Kluwer Academic Publishers.

BIDOIT N. AND FROIDEVAUX C. [1991*a*], 'General logical Databases and Programs: Default Logic Semantics and Stratification', *Information and Computation* **91**, 15–54.

BIDOIT N. AND FROIDEVAUX C. [1991*b*], 'Negation by Default and unstratifiable logic Programs', *Theoretical Computer Science* **78**, 85–112.

BIDOIT N. AND HULL R. [1986], Positivism vs. minimalism in deductive databases, *in* 'Proc. of the 5th ACM Symp. on Principles of Database Systems (PODS'86)', pp. 123–132.

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

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

BRASS S. AND DIX J. [1994], A disjunctive semantics based on unfolding and bottom-up evaluation, *in* B. Wolfinger, ed., 'Innovationen bei Rechen- und Kommunikationssystemen, *(IFIP '94-Congress, Workshop FG2: Disjunctive Logic Programming and Disjunctive Databases)*', Springer, Berlin, pp. 83–91.

BRASS S. AND DIX J. [1995], A General Approach to Bottom-Up Computation of Disjunctive Semantics, *in* J. Dix, L. Pereira and T. Przymusinski, eds, 'Nonmonotonic Extensions of Logic Programming', LNAI 927, Springer, Berlin, pp. 127–155.

BRASS S. AND DIX J. [1997], 'Characterizations of the Disjunctive Stable Semantics by Partial Evaluation', *Journal of Logic Programming* **32(3)**, 207–228. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).

BRASS S. AND DIX J. [1998], 'Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA', *Journal of Automated Reasoning* **20(1)**, 143–165. (Extended abstract appeared in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).

BRASS S. AND DIX J. [1999], 'Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation', *Journal of Logic Programming* **38(2)**, 167–213. (Extended abstract appeared in: Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation, *Proceedings of the 12-th International Logic Programming Conference, Tokyo*, pages 199–213, 1995. MIT Press.).

BRASS S., DIX J., FREITAG B. AND ZUKOWSKI [2001], 'Transformation-based bottom-up computation of the well-founded model', *Theory and Practice of Logic Programming* **to appear**.

BRASS S., DIX J., NIEMELÄ I. AND PRZYMUSINSKI T. [2001], 'On the Equivalence of the Static and Disjunctive Well-Founded Semantics and its Computation', *Theoretical Computer Science* **256**.

BRASS S., DIX J., NIEMELÄ I. AND PRZYMUSINSKI T. C. [1998], A Comparison of the Static and the Disjunctive Well-founded Semantics and its Implementation, *in* A. G. Cohn, L. K. Schubert and S. C. Shapiro, eds, 'Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)', San Francisco, CA, Morgan Kaufmann, pp. 74–85. appeared also as TR 17/97, University of Koblenz.

BRASS S., DIX J. AND PRZYMUSINSKI T. [1999], 'Computation of the semantics of autoepistemic belief theories', *Artificial Intelligence* **112**(1-2), 104–123.

BRASS S., ZUKOWSKI U. AND FREITAG B. [1997], Transformation Based Bottom-Up Computation of the Well-Founded Model, *in* J. Dix, L. Pereira and T. Przymusinski, eds, 'Nonmonotonic Extensions of Logic Programming', LNAI 1216, Springer, Berlin, pp. 171–201.

Brewka G. [1996], *Principles of Knowledge Representation*, CSLI, Dikran.

Brewka G. and Dix J. [2001], Knowledge representation with extended logic programs, *in* D. Gabbay and F. Guenthner, eds, 'Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies', Reidel Publ., chapter 6.

Brewka G., Dix J. and Konolige K. [1997], *Nonmonotonic Reasoning: An Overview*, CSLI Lecture Notes 73, CSLI Publications, Stanford, CA.

Bry F. and Yahya A. [1996], Minimal Model Generation with Positive Unit Hyper-Resolution Tableaux, *in* P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Theorem Proving with Analytic Tableaux and Related Methods', number 1071 *in* 'Lecture Notes in Artificial Intelligence', Springer, pp. 143–159.

Buccafurri F., Leone N. and Rullo P. [1996], 'Stable models and their computation for logic programming with inheritance and true negation', *Journal of Logic Programming* **27**(1), 5–43.

Buccafurri F., Leone N. and Rullo P. [1997], Strong and Weak Constraints in Disjunctive Datalog, *in* 'Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)', Dagstuhl, Germany, pp. 2–17.

Cadoli M., Donini F. M., Liberatore P. and Schaerf M. [1995], The size of a revised knowledge base, *in* 'PODS '95', pp. 151–162.

Cadoli M., Donini F. M. and Schaerf M. [1996], 'Is intractability of non-monotonic reasoning a real drawback?', *Artificial Intelligence Journal* **88**, 215–251.

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

Cadoli M. and Lenzerini M. [1990], The complexity of closed world reasoning and circumscription, *in* 'AAAI-90 — Proceedings of the 9^{th} AAAI National Conference on Artificial Intelligence', The MIT Press, Boston, MA, USA, pp. 550–555.

Ceri S., Gottlob G. and Tanca L. [1990], *Logic Programming and Databases*, Surveys in Computer Science, Springer.

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

Chang C.-L. and Lee R. C.-T. [1973], *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, New York.

Chellas B. [1980], *Modal Logic: an Introduction*, Cambridge University Press, Cambridge.

Chen W., Swift T. and Warren D. S. [1995], 'Efficient Top-Down Computation of Queries under the Well-Founded Semantics', *Journal of Logic Programming* **24(3)**, 219–245.

Chen W. and Warren D. S. [1993], 'A Goal Oriented Approach to Computing The Well-founded Semantics', *Journal of Logic Programming* **17**, 279–300.

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

Chen W. and Warren D. S. [1996], 'Tabled Evaluation with Delaying for General Logic Programs', *Journal of the ACM* **43**(1), 20–74.

Cholak P. and Blair H. [1994], 'The Complexity of Local Stratification', *Fundamenta Informaticae* **XXI(4)**, 333–344.

Cholewiński P., Marek V., Mikitiuk A. and Truszczyński M. [1995], Experimenting with nonmonotonic reasoning, *in* 'Proceedings of the 12th International Conference on Logic Programming', Tokyo, pp. 267–281.

Clark K. L. [1978], Negation as Failure, *in* H. Gallaire and J. Minker, eds, 'Logic and Data-Bases', Plenum, New York, pp. 293–322.

Crawford J. and Auton L. [1996], 'Experimental results on the crossover point in random 3-SAT', *Artificial Intelligence* **81**(1), 31–57.

DIMOPOULOS Y., NEBEL B. AND KOEHLER J. [1997], Encoding planning problems in non-monotonic logic programs, *in* 'Proceedings of the Fourth European Conference on Planning', Springer-Verlag, Toulouse, France. To appear.

DIX J. [1992*a*], A Framework for Representing and Characterizing Semantics of Logic Programs, *in* B. Nebel, C. Rich and W. Swartout, eds, 'Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR '92)', San Mateo, CA, Morgan Kaufmann, pp. 591–602.

DIX J. [1992*b*], Classifying Semantics of Disjunctive Logic Programs, *in* K. R. Apt, ed., 'LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium', MIT Press, Cambridge, Mass., pp. 798–812.

DIX J. [1995*a*], 'A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties', *Fundamenta Informaticae* **XXII(3)**, 257–288.

DIX J. [1995*b*], 'Detailed Report on the First LP & NMR retreat', *AI Communications* **Vol. 8, No. 1**, 39–43. Also in Computational Logic, Vol. 2, 1995, pages 94–97.

DIX J. [1995*c*], Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview., *in* A. Fuhrmann and H. Rott, eds, 'Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence', DeGruyter, pp. 241–327.

DIX J. [1998], 'The Logic Programming Paradigm', *AI Communications* **Vol. 11, No. 3**, 39–43. Short version in Newsletter of ALP, Vol. 11(3), 1998, pages 10–14.

DIX, J., FURBACH, U. AND NERODE, A., EDS [1997], *Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, Springer, Berlin.

DIX J., KRAUS S. AND SUBRAHMANIAN V. S. [2001], 'Temporal agent reasoning', *Artificial Intelligence* .

DIX J. AND LOBO J. [1999], 'Logic Programming and non-monotonic reasoning', *Special Issue of the Annals in Mathematics and Artificial Intelligence* **25**(3–4).

DIX J. AND MÜLLER M. [1994*a*], Partial Evaluation and Relevance for Approximations of the Stable Semantics, *in* Z. Ras and M. Zemankova, eds, 'Proceedings of the 8th Int. Symp. on Methodologies for Intelligent Systems, Charlotte, NC, 1994', LNAI 869, Springer, Berlin, pp. 511–520.

DIX J. AND MÜLLER M. [1994*b*], The Stable Semantics and its Variants: A Comparison of Recent Approaches, *in* L. Dreschler-Fischer and B. Nebel, eds, 'Proceedings of the 18th German Annual Conference on Artificial Intelligence (KI '94), Saarbrücken, Germany', LNAI 861, Springer, Berlin, pp. 82–93.

DIX J., NANNI M. AND SUBRAHMANIAN V. [2000], 'Probabilistic Agent Programs', *ACM Transactions of Computational Logic* **I**(2), 207–245.

DIX J. AND OSORIO M. [1999], 'Confluent rewriting systems in non-monotonic reasoning', *Computacion y Sistemas* **Volume II, No. 2-3**, 104–123.

DIX J., OSORIO M. AND ZEPEDA C. [2001], 'A General Theory of Confluent Rewriting Systems for Logic Programming and its Applications', *Annals of Pure and Applied Logic* **to appear**.

DIX, J., PEREIRA, L. AND PRZYMUSINSKI, T., EDS [1995], *Non-Monotonic Extensions of Logic Programming*, LNAI 927, Springer, Berlin.

DIX, J., PEREIRA, L. AND PRZYMUSINSKI, T., EDS [1997], *Non-Monotonic Extensions of Logic Programming*, LNAI 1216, Springer, Berlin.

DIX, J., PEREIRA, L. AND PRZYMUSINSKI, T., EDS [1998], *Logic Programming and Knowledge Representation*, LNAI 1471, Springer, Berlin.

DIX J. AND SCHMITT P. [1993], 'Nichtmonotones Schließen: Wieviel Nichtmonotonie ist nötig?', *Kognitionswissenschaft* **3**, 53–69.

DIX J. AND STOLZENBURG F. [1998], 'A Framework to incorporate Nonmonotonic Reasoning into Constraint Logic Programming', *Journal of Logic Programming* **37(1,2,3)**, 47—76. Special Issue on *Constraint Logic Programming*, Guest Editors: Kim Marriott and Peter Stuckey.

DIX J., SUBRAHMANIAN V. AND PICK G. [2000], 'Meta Agent Programs', *Journal of Logic Programming* **46**(1–2), 1–60.

Doherty P., Lukaszewicz W. and Szalas A. [1995], Computing circumscription revisited: Preliminary report, *in* 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Montreal, Canada, pp. 1502–1508.

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

Dung P. M. [1992], 'On the relations between stable and wellfounded semantics of logic programs', *Theoretical Computer Science* **105**, 7–25.

Eiter T. and Gottlob G. [1993], 'Propositional Circumscription and Extended Closed World Reasoning are Π_2^P-complete', *Theoretical Computer Science* **144**(2), 231–245, Addendum: vol. 118, p. 315, 1993.

Eiter T. and Gottlob G. [1995*a*], 'On the Computational Cost of Disjunctive Logic Programming: Propositional Case', *Annals of Mathematics and Artificial Intelligence* **15**(3/4), 289–323.

Eiter T. and Gottlob G. [1995*b*], 'The Complexity of Logic-Based Abduction', *Journal of the Association for Computing Machinery* **42**(1), 3–42.

Eiter T. and Gottlob G. [1996], 'Mächtigkeit von Logikprogrammierung über Datenbanken', *KI* **3**, 32–39.

Eiter T., Gottlob G. and Mannila H. [1997], 'Disjunctive Datalog', *ACM Transactions on Database Systems* **22**(3), 364–417.

Eiter T., Leone N., Mateis C., Pfeifer G. and Scarcello F. [1997], A Deductive System for Nonmonotonic Reasoning, *in* J. Dix, U. Furbach and A. Nerode, eds, 'Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97)', number 1265 *in* 'Lecture Notes in AI (LNAI)', Springer, Berlin.

Eiter T., Leone N., Mateis C., Pfeifer G. and Scarcello F. [1998], The KR System `dlv`: Progress Report, Comparisons and Benchmarks., *in* 'Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)', pp. 406–417.

Eiter T., Subrahmanian V. and Pick G. [1999], 'Heterogeneous Active Agents, I: Semantics', *Artificial Intelligence* **108**(1-2), 179–255.

Eiter T. and Subrahmanian V. S. [1999], 'Heterogeneous Active Agents, II: Algorithms and Complexity', *Artificial Intelligence* **108**(1-2), 257–307.

Etherington D. W. and Reiter R. [1983], On Inheritance Hierarchies with Exceptions, *in* 'Proceedings AAAI'.

Fages F. [1993], 'Consistency of Clark's completion and existence of stable models', *Methods of Logic in Computer Science* **2**.

Fernandez J. and Minker J. [91], Bottom-up evaluation of hierarchical disjunctive deductive databases, *in* K. Furukawa, ed., 'Proc. 8th International Conference on Logic Programming', pp. 660–675.

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

Fitting M. C. [1985], 'A Kripke-Kleene Semantics of logic Programs', *Journal of Logic Programming* **4**, 295–312.

Gabbay D. [1984], Theoretical Foundations for Non-Monotonic Reasoning in Expert- Systems, *in* K. R. Apt, ed., 'Logics and Models of Concurrent Systems', Springer, Berlin, pp. 439–458.

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

Gelfond M. and Lifschitz V. [1988], The Stable Model Semantics for Logic Programming, *in* R. Kowalski and K. Bowen, eds, '5th Conference on Logic Programming', MIT Press, pp. 1070–1080.

Gelfond M. and Lifschitz V. [1990], Logic Program with Classical Negation, *in* D. H. Warren and P. Szeredi, eds, 'Proceedings of the 7th Int. Conf. on Logic Programming', MIT, pp. 579–597.

GELFOND M. AND LIFSCHITZ V. [1993], 'Representing Actions and Change by Logic Programs', *Journal of Logic Programming* **17**, 301–322.

GINSBERG M. [1989], 'A circumscriptive theorem prover', *Artificial Intelligence* **39**, 209–230.

GOGIC G., PAPADIMITRIOU C., SELMAN B. AND KAUTZ H. [1995], The Comparative Linguistics of Knowledge Representation, *in* 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Montreal, Canada, pp. 862–869.

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

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

HÄHNLE R. AND PAPE C. [1997], Ordered tableaux: Extensions and applications, *in* D. Galmiche, ed., 'Automated Reasoning with Tableaux and Related Methods', number 1227 *in* 'LNAI', Springer, pp. 173–187.

HUGHES G. AND CRESSWELL M. [1984], *A Companion to Modal Logic*, Methuen and Co., London.

INOUE K., KOSHIMURA M. AND HASEGAWA R. [1992], Embedding negation as failure into a model generation theorem prover, *in* 'The 11th International Conference on Automated Deduction', Springer-Verlag, Saratoga Springs, NY, USA, pp. 400–415.

ISC [1985], 'The ISCAS-85 Benchmarks', `http://www.cbl.ncsu.edu/www/CBL_Docs/iscas85.html`.

JOHNSON D. [1990], A catalog of complexity classes, *in* J. van Leeuwen, ed., 'Handbook of Theoretical Computer Science', Vol. A. Algorithms and Complexity, Elsevier, pp. 67–161.

JUNKER U. AND KONOLIGE K. [1990], Computing the extensions of autoepistemic and default logics with a truth maintenance system, *in* 'Proceedings of the 8th National Conference on Artificial Intelligence', MIT Press, Boston, MA, USA, pp. 278–283.

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

KARTHA G. N. AND LIFSCHITZ V. [1995], A Simple Formalization of Actions Using Circumsription, *in* C. S. Mellish, ed., '14th IJCAI', Vol. 2, Morgan Kaufmann, pp. 1970–1975.

KAUTZ H. AND SELMAN B. [1991], 'Hard problems for simple default logics', *Artificial Intelligence* **49**, 243–279.

KONOLIGE K. [1991], Quantification in autoepistemic logic, Technical Note 510, SRI International, Menlo Park, California, USA.

KONOLIGE K. [1994], Autoepistemic logic, *in* D. Gabbay, C. Hogger and J. Robinson, eds, 'Handbook of Logic and Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning', Oxford University Press, pp. 217–295.

KOWALSKI R. A. AND KUEHNER D. [1971], 'Linear resolution with selection function', *Artificial Intelligence* **2**, 227–260.

KRAUS S., LEHMANN D. AND MAGIDOR M. [1990], 'Nonmonotonic Reasoning, Preferential Models and Cumulative Logics', *Artificial Intelligence* **44**(1), 167–207.

KÜHN M. [1997], Rigid Hypertableaux, *in* G. Brewka, C. Habel and B. Nebel, eds, 'KI-97: Advances in Artificial Intelligence', number 1303 *in* 'LNAI', Springer, pp. 87–99.

LEHMANN D. AND MAGIDOR M. [1992], 'What does a conditional knowledge base entail?', *Artificial Intelligence* **55**, 1–60.

LEONE N., ROMEO M., RULLO P. AND SACCÀ D. [1993], Effective implementation of negation in database logic query languages, *in* 'LOGIDATA+: Deductive Database with Complex Objects', LNCS 701, pp. 159–175.

LEONE N. AND RULLO P. [1992], 'Safe computation of the well-founded semantics of datalog queries', *Information Systems* **17**(1), 17–31.

LEONE N., RULLO P. AND SCARCELLO F. [1997], 'Disjunctive stable models: Unfounded sets, fixpoint semantics and computation', *Information and Computation* **135**(2), 69–112.

LETZ R., MAYR K. AND GOLLER C. [1994], 'Controlled Integrations of the Cut Rule into Connection Tableau Calculi', *Journal of Automated Reasoning* **13**.

LIFSCHITZ V. [1985], Computing Circumscription, *in* 'Proceedings of the International Joint Conference on Artificial Intelligence, Los Angeles, California', pp. 121–127.

LIFSCHITZ V. [1989], Benchmark Problems for Formal NMR, Version 2.00, *in* Reinfrank, de Kleer, Ginsberg and Sandewall, eds, 'Non-Monotonic Reasoning', LNAI 346, Springer, Berlin, pp. 202–219.

LIFSCHITZ V. [1994], *Circumscription*, Clarendon, Oxford, pp. 297–353.

LLOYD J. W. [1987], *Foundations of Logic Programming*, Springer, Berlin. 2nd edition.

LOBO J., MINKER J. AND RAJASEKAR A. [1992], *Foundations of Disjunctive Logic Programming*, MIT Press.

LOVELAND D. [1968], 'Mechanical Theorem Proving by Model Elimination', *JACM* **15**(2).

LOVELAND D. [1969], 'A Simplified Version for the Model Elimination Theorem Proving Procedure', *Journal of the Association for Computing Machinery* **16**(3).

LOVELAND D. [1991], 'Near-Horn Prolog and Beyond', *Journal of Automated Reasoning* **7**, 1–26.

MAKINSON D. [1994], General Patterns in Nonmonotonic Reasoning, *in* D. Gabbay, C. Hogger and J. Robinson, eds, 'Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3, Nonmonotonic and Uncertain Reasoning', Oxford University Press, chapter 3, pp. 35–110.

MANTHEY R. AND BRY F. [1988], SATCHMO: a theorem prover implemented in Prolog, *in* E. Lusk and R. Overbeek, eds, 'Proceedings of the 9^{th} Conference on Automated Deduction, Argonne, Illinois, May 1988', Vol. 310 of *Lecture Notes in Computer Science*, Springer, pp. 415–434.

MAREK W. [1989], 'Stable theories in autoepistemic logic', *Fundamenta Informaticae* **12**, 243–254.

MAREK, W., NERODE, A. AND TRUSZCZYŃSKI, M., EDS [1995], *Logic Programming and Nonmonotonic Reasoning*, LNAI 928, Springer, Berlin.

MAREK W., SCHWARZ G. AND TRUSZCZYŃSKI M. [1993], 'Modal nonmonotonic logics: Ranges, characterization, computation', *Journal of the Association for Computing Machinery* **40**(4), 963–990.

MAREK W. AND TRUSZCZYŃSKI M. [1991*a*], 'Autoepistemic logic', *Journal of the Association for Computing Machinery* **38**, 588–619.

MAREK W. AND TRUSZCZYŃSKI M. [1991*b*], Computing intersection of autoepistemic expansions, *in* 'Proceedings of the 1st International Workshop on Logic Programming and Non-monotonic Reasoning', The MIT Press, Washington, D.C., USA, pp. 37–50.

MAREK W. AND TRUSZCZYŃSKI M. [1993], *Nonmonotonic Logics; Context-Dependent Reasoning*, 1st edn, Springer, Berlin.

MCCARTHY J. [1979], First order theories of individual concepts and propositions, *in* B. Meltzer and D. Michie, eds, 'Machine Intelligence 9', Edinburgh University Press, Edinburgh, pp. 120–147.

MCCARTHY J. [1980], 'Circumscription: A Form of Nonmonotonic Reasoning', *Artificial Intelligence* **13**, 27–39.

MCCARTHY J. [1986], 'Applications of Circumscription to formalizing Common-Sense Reasoning', *Artificial Intelligence* **28**, 89–116.

MCDERMOTT D. [1982], 'Non-monotonic logic II', *Journal of the Association for Computing Machinery* **29**, 33–57.

MCDERMOTT D. AND DOYLE J. [1980], 'Non-monotonic logic I', *Artificial Intelligence* **13**, 41–72.

MINKER J. [1993], 'An Overview of Nonmonotonic Reasoning and Logic Programming', *Journal of Logic Programming, Special Issue* **17**(2/3/4), 95–126.

MINKER J. [1996], Logic and databases: A 20 year retrospective, *in* D. Pedreschi and C. Zaniolo, eds, 'Proceedings of the International Workshop on Logic in Databases (LID)', LNCS 1154, Springer, Berlin, pp. 3–58.

MINKER J. AND ZANON G. [1982], 'An extension to linear resolution with selection function', *Information Processing Letters* **14**(3), 191–194.

MOORE R. [1985], 'Semantical considerations on nonmonotonic logic', *Artificial Intelligence* **25**, 75–94.

NERODE A., NG R. AND SUBRAHMANIAN V. [1995], 'Computing circumscriptive databases: I. theory and algorithms', *Information and Computation* **116**, 58–80.

NIEMELÄ I. [1990], Towards automatic autoepistemic reasoning, *in* 'Proceedings of the European Workshop on Logics in Artificial Intelligence—JELIA'90', Springer-Verlag, Amsterdam, The Netherlands, pp. 428–443.

NIEMELÄ I. [1991], Constructive tightly grounded autoepistemic reasoning, *in* 'Proceedings of the 12th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Sydney, Australia, pp. 399–404.

NIEMELÄ I. [1992], 'On the decidability and complexity of autoepistemic reasoning', *Fundamenta Informaticae* **17**(1,2), 117–155.

NIEMELÄ I. [1995*a*], 'A decision method for nonmonotonic reasoning based on autoepistemic reasoning', *Journal of Automated Reasoning* **14**, 3–42.

NIEMELÄ I. [1995*b*], Towards efficient default reasoning, *in* 'Proceedings of the 14th International Joint Conference on Artificial Intelligence', Morgan Kaufmann Publishers, Montreal, Canada, pp. 312–318.

NIEMELÄ I. [1996*a*], Implementing circumscription using a tableau method, *in* W. Wahlster, ed., 'Proceedings of the European Conference on Artificial Intelligence', John Wiley, Budapest, Hungary, pp. 80–84.

NIEMELÄ I. [1996*b*], A tableau calculus for minimal model reasoning, *in* P. Miglioli, U. Moscato, D. Mundici and M. Ornaghi, eds, 'Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods', LNAI 1071, Springer-Verlag, Terrasini, Italy, pp. 278–294.

NIEMELÄ I. AND RINTANEN J. [1994], 'On the impact of stratification on the complexity of nonmonotonic reasoning', *Journal of Applied Non-Classical Logics* **4**(2), 141–179.

NIEMELÄ I. AND SIMONS P. [1996], Efficient Implementation of the Well-founded and Stable Model Semantics, *in* M. Maher, ed., 'Proceedings of the Joint International Conference and Symposium on Logic Programming', The MIT Press, Bonn, Germany, pp. 289–303.

NIEMELÄ I. AND SIMONS P. [1997], 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', Springer-Verlag, Dagstuhl, Germany.

ODIFREDDI P. [1989], *Classical Recursion Theory*, North-Holland.

OLIVETTI N. [1992], 'A tableaux and sequent calculus for minimal entailment', *Journal of Automated Reasoning* **9**, 99–139.

PAPADIMITRIOU C. [1994], *Computational Complexity*, Addison-Wesley.

PEREIRA, L. M. AND NERODE, A., EDS [1993], *Logic Programming and Non–monotonic Reasoning: Proceedings of the Second Int. Ws.*, MIT Press, Lisboa, Portugal.

PLAISTED D. [1988], 'Non-Horn Clause Logic Programming Without Contrapositives', *Journal of Automated Reasoning* **4**, 287–325.

PRZYMUSINSKA H. AND PRZYMUSINSKI T. [1994], 'Stationary default extensions', *Fund. Informat.* **21 (1,2)**, 67–87.

PRZYMUSINSKI T. C. [1989], 'An Algorithm to Compute Circumscription', *Artificial Intelligence* **38**, 49–73.

PRZYMUSINSKI T. C. [1998], 'Autoepistemic logic of knowledge and beliefs', *Artificial Intelligence* **95**, 115–154.

RAJASEKAR A. [1989], Semantics for Disjunctive Logic Programs, PhD thesis, University of Maryland.

RAJASEKAR A. AND YUSUF H. [1995], 'DWAM – A WAM model extension for disjunctive logic programs', *Annals of Mathematics and Artificial Intelligence* **14**(2-4), 275–308. Presented also at the Symposium on Logic in Databases, Knowledge Representation and Reasoning, College Park, Md., USA.

RAO P., SAGONAS K., SWIFT T., WARREN D. S. AND FREIRE J. [1997], XSB: A System for Efficiently Computing Well-Founded Semantics, *in* J. Dix, U. Furbach and A. Nerode, eds, 'Logic Programming and Non-Monotonic Reasoning, Proceedings of the Fourth International Conference', LNAI 1265, Springer, Berlin, pp. 430–440.

REITER R. [1978], On closed world data bases, *in* H. Gallaire and J. Minker, eds, 'Logic and Data Bases', Plenum, New York, pp. 55–76.

REITER R. [1980], 'A Logic for Default-Reasoning', *Artificial Intelligence* **13**, 81–132.

REITER R. [1987], 'A Theory of Diagnosis from First Principles', *Artificial Intelligence* **32**, 57–97.

RISCH V. AND SCHWIND C. [1994], 'Tableau-based characterization and theorem proving for default logic', *Journal of Automated Reasoning* **13**, 223–242.

SAGONAS K., SWIFT T. AND WARREN D. S. [1994], XSB as an Efficient Deductive Database Engine, *in* 'Proc. of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94)', ACM Press, Minneapolis, Minnesota, pp. 442–453.

SAKAMA C. AND INOUE K. [1993], Relating disjunctive logic programs to default theories, *in* 'Proceedings of the 2nd International Conference on Logic Programming and Non-Monotonic Reasoning', MIT Press, pp. 266–282.

SAKAMA C. AND SEKI H. [1994], Partial Deduction of Disjunctive Logic Programs: A Declarative Approach, *in* 'Logic Program Synthesis and Transformation – Meta Programming in Logic', LNCS 883, Springer, Berlin, pp. 170–182.

SAKAMA C. AND SEKI H. [1997], 'Partial Deduction in Disjunctive Logic Programming', *Journal of Logic Programming* **32(3)**, 229–245.

SCHAUB T. [1995], 'A new methodology for query-answering in default logics via structure-oriented theorem proving', *Journal of Automated Reasoning* **15**(1), 95–165.

SCHAUB T. AND BRÜNING S. [1996], Prolog technology for default reasoning, *in* 'Proceedings of the European Conference on Artificial Intelligence', John Wiley, Budapest, Hungary, pp. 105–109.

SCHLIPF J. [1986], How uncomputable is general circumscription?, *in* 'Symposium on Logic in Computer Science', IEEE Computer Society Press, Cambridge, USA, pp. 92–95.

SCHWARZ G. AND TRUSZCZYŃSKI M. [1993], Nonmonotonic reasoning is sometimes simpler, *in* 'Proceedings of the 3rd Kurt Gödel Colloquium on Computational Logic and Proof Theory', Springer, LNCS, Brno, Czech Republic, pp. 313–324.

SHVARTS G. [1990], Autoepistemic modal logics, *in* 'Proceedings of the 3rd Conference on Theoretical Aspects of Reasoning about Knowledge', Morgan Kaufmann Publishers, Pacific Grove, USA, pp. 97–109.

STICKEL M. [1988], 'A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler', *Journal of Automated Reasoning* **4**, 353–380.

STILLMAN J. [1990], 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', The MIT Press, Boston, Massachusetts, USA, pp. 571–578.

STILLMAN J. [1992], The complexity of propositional default logics, *in* 'AAAI-92 — Proceedings of the AAAI National Conference on Artificial Intelligence', The MIT Press, San Jose, California, USA, pp. 794–800.

SUBRAHMANIAN V., BONATTI P., DIX J., EITER T., KRAUS S., ÖZCAN F. AND ROSS R. [2000], *Heterogenous Active Agents*, MIT-Press.

SUBRAHMANIAN V., NAU AND VAGO [1995], 'WFS + Branch and Bound = Stable Models', *IEEE Transactions on Knowledge and Data Engineering* **17(3)**, 362–377.

TAMAKI H. AND SATO T. [1986], OLD Resolution with Tabulation, *in* 'Proceedings of the Third International Conference on Logic Programming, London', LNAI, Springer, Berlin, pp. 84–98.

TARJAN R. [1972], 'Depth first search and linear graph algorithms', *SIAM Journal on Computing* **1**, 146–160.

TRUSZCZYŃSKI M. [1991], Embedding default logic into modal nonmonotonic logics, *in* 'Proceedings of the 1st International Workshop on Logic Programming and Non-monotonic Reasoning', The MIT Press, Washington, D.C., USA, pp. 151–165.

VAN GELDER A. [1988], Negation as failure using tight derivations for general logic programs, *in* J. Minker, ed., 'Foundations of Deductive Databases and Logic Programming', Morgan Kaufmann Publishers, Los Altos, pp. 149–176.

VAN GELDER A. [1989], The Alternating Fixpoint of Logic Programs with Negation, *in* 'Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Philadelphia, Pennsylvania', ACM Press, pp. 1–10.

VAN GELDER A., ROSS K. A. AND SCHLIPF J. S. [1988], Unfounded Sets and well-founded Semantics for general logic Programs, *in* 'Proceedings 7th Symposion on Principles of Database Systems', pp. 221–230.

VAN GELDER A., ROSS K. AND SCHLIPF J. [1991], 'The well-founded semantics for general logic programs', *Journal of the Association for Computing Machinery* **38**(3), 620–650.

Index

R

S

T

U

W

X