HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A:** Research Reports ISSN 0783–5396
No. **20**; November 1992 ISBN 951-22-1300-1

STRATIFICATION AND TRACTABILITY IN NONMONOTONIC REASONING

Jussi Rintanen

Digital Systems Laboratory
Department of Computer Science
Helsinki University of Technology
Otaniemi, FINLAND

Helsinki University of Technology
Department of Computer Science
Digital Systems Laboratory
Otaniemi, Otakaari 1
FIN–02150 ESPOO, FINLAND

HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A**: Research Reports
No. **20**; November 1992

ISSN 0783–5396
ISBN 951-22-1300-1

Stratification and Tractability in Nonmonotonic Reasoning

JUSSI RINTANEN

Abstract: Knowledge-based systems use forms of reasoning that do not satisfy the monotonicity property of classical logical reasoning. Therefore, several nonmonotonic logics have been developed for the investigation of theoretical foundations of knowledge-based systems. This work investigates efficient inference procedures for knowledge representation using the framework of nonmonotonic logics. Recent results show that nonmonotonic reasoning cannot be brought to tractable level by restricting merely the form of the formulae. In this work the impact of the structural property of *stratification* on the complexity of nonmonotonic reasoning is investigated. Autoepistemic logic is used as the framework for establishing the results, because several other nonmonotonic formalisms can be embedded in it. The main result of the work is that in stratified cases the complexity of nonmonotonic reasoning is approximately the same as that of corresponding classical monotonic reasoning. This is shown by giving a general decision procedure for stratified autoepistemic theories and analyzing its complexity with respect to the complexity of the classical theorem-proving required. It turns out that for theories for which the classical theorem-proving is polynomial time, also the decision procedure runs in polynomial time. As an example of this, two tractable classes of autoepistemic theories based on the Horn clause subset of propositional logic are presented. Also the implications of these results on the complexity of reasoning in default logic, truth-maintenance systems, and propositional logic programs are discussed.
Keywords: nonmonotonic reasoning, autoepistemic logic, tractability, stratification.

Printing: TKK Monistamo; Otaniemi 1992

Helsinki University of Technology
Department of Computer Science
Digital Systems Laboratory
Otaniemi, Otakaari 1
FIN–02150 ESPOO, FINLAND

Phone: $\frac{90}{+358\text{–}0}$ 4511
Telex: 125 161 htkk fi
Telefax: +358–0–465 077
E-mail: lab@saturn.hut.fi

Contents

1 Introduction

The aim of knowledge representation research is to develop better software systems for representing knowledge about complex real-world problem domains. Current techniques either support the representation of only limited forms of knowledge or data, or the ways the knowledge can be used are restricted. The idea that the representation of knowledge should be *declarative* is one of the basic assumptions of the research area. Declarativeness means that the meaning of the knowledge is independent of the mechanisms that utilize it. This makes the construction and understanding of knowledge bases possible without deep understanding of the procedural aspects of the knowledge-based system. Declarativeness also brings a certain level of flexibility: it is easier to use knowledge for new purposes if its representation is not tightly tied to the original mechanisms for using it, and proper notions of consistency of knowledge base and correctness of query procedures are easier to define.

A good example of a declarative paradigm of knowledge representation that is used in practice is the relational data model used in relational databases. The contents of a database can be understood independently of the query evaluation mechanisms, and the correct answers to queries are uniquely defined by relational algebra. The correctness of implementations of query evaluation and optimization procedures can be assessed rigorously against the correct semantics specified by the relational algebra. The relational data model and the expressivity of relational algebra queries are fairly restricted and hence the relational data model is not usually considered a genuine knowledge representation formalism.

Research in artificial intelligence and knowledge engineering has produced many software systems for knowledge representation. These systems are best referred to as software because they are not declarative and their existence as formal objects independent of their particular implementations is controversial. For example, production rule languages which are perhaps the paradigmatic example of knowledge representation in existing practical systems, are used for application development the same way as programming languages [Hayes-Roth, 1987] and the production rules in such systems often do not have clear declarative meaning.

1.1 Nonmonotonicity

First-order predicate logic, like most other logics, is a declarative language using which knowledge can be represented. The semantics of predicate logic

person(Pertti)
person(Emma)
...
person(Anne)

blind(Anne)
fat(Pertti)
...

Figure 1: A Database

is well understood, and a lot of research has been devoted to the development of efficient theorem-proving procedures for predicate logic. The obvious question arises: why is this logic not used as the basis of software systems for knowledge representation. It is clear that predicate logic is not an universal solution to all knowledge representation problems, e.g. because it does not naturally support the representation of degrees of uncertainty, but even for applications for which two truth values are enough, predicate logic suffers from some deficiences. For example, its applicability is hindered by its unfavourable computability and complexity properties. This of course is also the case with every other equally expressive language; a fact which suggests that less expressive languages should suffice. Another problem is that predicate logic and other classical logics do not support the representation of default and negative information in a natural way.

1.1.1 The Closed-World Assumption

In classical logics a sentence beginning with a negation sign has to be inferred exactly the same way as any other sentence, namely using the available axioms and inference rules. However, in practical applications it seems that also a different form of negation is needed, a negation that would correspond to the *unprovability* of a formula, i.e., everything which is not known to be true, can be assumed false. In the context of databases this notion is called the *closed-world assumption* [Reiter, 1978]. For example, given the database about persons and their properties in Figure 1, we would like to be able to conclude that Emma is not blind because no knowledge of such an inability is in the database. However, if we interpret the database in a logical fashion we find that $\neg$*blind(Emma)* is not provable from it. Extending the database to explicitly contain all information about normal persons not being blind, so that the respective conclusions could be logically drawn, would clearly be very uneconomical; certainly something we would like to avoid if at all possible.

The database extended with *blind(Emma)* does not admit the conclusion $\neg$*blind(Emma)* by the closed-world assumption although the original database did. Negation corresponding to unprovability thus breaks the property of monotonicity which holds for all classical logics:

$$\Delta \vdash \phi \text{ and } \Delta \subseteq \Gamma \text{ implies } \Gamma \vdash \phi \tag{1}$$

i.e., the set of theorems of a theory grows monotonically as the theory is extended. Consequently it seems that classical logics are incompatible with more economical ways of representing negative information, and that logics more suitable for knowledge representation are nonmonotonic.

1.1.2 The Frame Problem

Temporal reasoning, i.e., reasoning about events, time, and changes caused by events, can easily be embedded in sufficiently expressive logical languages, e.g. in first-order predicate logic. In practice, the specification of what things are not affected by events poses the so-called *frame problem* [McCarthy and Hayes, 1969] the solution of which seems to require logical formalisms exhibiting nonmonotonic behaviour.

In predicate logic formalizations of temporal reasoning, states of the world can be denoted by terms that describe what events have taken place since the initial situation. The effects of events can be stated as axioms, e.g. "If the switch n:o 1 is turned to *on* position, the lamp is lit". The curious thing from a common-sense point of view is that even if this axiom is the only axiom concerning switch 1, we cannot infer whether the lamp is the only thing that is affected by turning the switch to the *on* position. It certainly is logically possible that switch 1 is also connected to e.g. a microwave oven which is also turned on. The explanation why e.g. predicate logic behaves seemingly counterintuitively in this situation is that for the above axiom there are always models for which the microwave oven is also affected. Therefore, in addition to defining what effects the events do have, also the things that are not affected by events have to be specified explicitly as *frame axioms*. The number of frame axioms may be significantly larger than the number of other axioms describing actual effects, because typical events affect only a small fraction of the whole world. Hence the representation task may become unmanageable because of the frame axioms [McCarthy and Hayes, 1969].

The explication of the frame axioms could be avoided if some kind of closed-world assumption were used just like in the database example earlier: only those effects take place which are explicitly specified. Nonmonotonicity is required because e.g. the above axiom and some kind of a closed-world assumption allow the conclusion that microwave oven is not affected, but if a suitable axiom concerning the microwave oven is introduced, then the conclusion cannot be made any more. Thus the natural formalization of this is not possible in e.g. predicate logic because this logic is monotonic.

1.1.3 Logic Programs

The logic programming language Prolog has a negation operator that exhibits nonmonotonic behaviour. In pure Prolog, i.e., Prolog without the negation operator and other nonlogical properties, program execution can be seen as a goal-directed theorem-proving in a subset of first-order predicate logic. The negation operator is essentially an unprovability operator, i.e., when in the execution of the program a negated goal is encountered, the theorem-prover is invoked recursively to test the unprovability of the goal, and if it fails to prove the goal the negated goal succeeds. This is called *negation by failure*. For example, the facts of the database presented earlier can be seen as clauses of a Prolog program. Consider the database extended with the following clause.

watches_tv(X) ← not blind(X)

When the goal *watches_tv(emma)* is given to the Prolog interpreter, the subgoal *not blind(emma)* is obtained by the clause *watches_tv(X) ← not blind(X)*. The goal *blind(emma)* fails because the blindness of Emma is not provable from the database, and consequently *not blind(emma)* succeeds and so does the original goal *watches_tv(emma)*. Again, extending the program with *blind(emma)* retracts the conclusion *watches_tv(emma)*. Hence, nonmonotonic reasoning is used by Prolog as well.

The negation in Prolog is very procedural. E.g., consider the program

male(X) ← not female(X)
female(X) ← not male(X)

that says that the entities that are not female are male, and vice versa (in an awkward way, though). Given the query *male(Pertti)*, the subgoal *not female(Pertti)* is obtained, which reduces to testing *female(Pertti)* and so on until we obtain the goal *male(Pertti)*, which is our query. Hence the program loops. Giving a declarative semantics for Prolog with the *not* operator in pathological situations like this as well, requires the investigation of principles of nonmonotonic reasoning. As in the case of pure Prolog, where the semantics is the predicate logic, the semantics of pure Prolog with *not* would also be some kind of logic, in this case a nonmonotonic one, however.

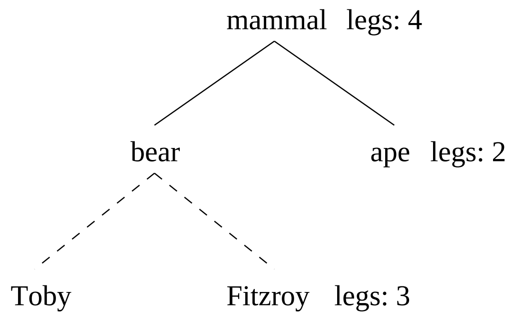

Figure 2: An Inheritance Hierarchy

1.1.4 Default Values and Inheritance

Object-oriented data representation is considered useful in representing concrete real-world entities that naturally form hierarchies of classes ordered by set inclusion. In object-oriented programming languages, frame languages, object-oriented databases and elsewhere, the hierarchical structure is utilized by sharing data between several classes and objects by inheritance. The default values for attributes of objects are defined in the parent class of the object or some superclass of the parent class, and hence the unnecessary explication of the default values in many objects is avoided. As an example of this, Figure 2 depicts three classes, *mammal*, *bear*, *ape*, and two bears, Toby and Fitzroy. By default, all mammals have four legs. For species of mammals that do not have four legs, like apes, the fact has to be specified separately. For other species the number of legs is obtained by inheritance from the parent class *mammal*. Bears, because they are mammals, usually have four legs. Fitzroy is an exceptional bear because it has somehow lost one of its legs. Inheritance reasoning is nonmonotonic because the default values in classes higher in the hierarchy may be overridden by specifying more specific values lower in the hierarchy.

1.2 Formalization of Nonmonotonic Reasoning

The examples above involving nonmonotonicity and the monotonicity of classical logics prompted researchers in artificial intelligence and knowledge representation to pursue general formalizations of nonmonotonic reasoning. The first formalizations which are also the leading ones were introduced in an issue of the Journal of Artificial Intelligence in 1980. These are McCarthy's circumscription, Reiter's default logic, McDermott and Doyle's nonmonotonic logic [McCarthy, 1980; Reiter, 1980; McDermott and Doyle, 1980]. McDermott and Doyle's logic was superseded by the autoepistemic logic of Moore [1985] which represents the same approach based on modal operators. These logics try to elucidate the principles of nonmonotonic reasoning, e.g. how it should be defined and how a rigorous semantics for it can be given. Default logic introduces nonmonotonicity in the form of nonmonotonic inference rules, autoepistemic logic and McDermott and Doyle's nonmonotonic logic have modal operators denoting provability and unprovability, and circumscription is based on model minimization which is syntactically formalized using special axiom schemes. Although the basic ideas of the above logics are distinct the relations between the logics are nevertheless close. Results of [Konolige, 1988; Marek and Truszczyński, 1989; Niemelä, 1991] indicate close connections between autoepistemic logic and default logic, e.g. default theories can be translated in a straightforward way to similar autoepistemic theories. Similarly, circumscription [Konolige, 1989], abduction [Kakas and Mancarella, 1990], truth-maintenance systems [Elkan, 1990], and logic programs [Gelfond and Lifschitz, 1988] can be embedded in the frameworks of default and autoepistemic logics.

1.2.1 Autoepistemic Logic

Because of the close relation of autoepistemic logic to other nonmonotonic formalisms we choose it as the framework of this work. The choice between default logic and autoepistemic logic is largely a matter of taste. Autoepistemic logic can be preferred e.g. because it provides a uniform syntax unlike default logic in which default rules unnecessarily are a syntactic category distinct from formulae.

The language of autoepistemic logic is the language of first-order predicate logic extended with the modal operator L which is read as "is believed" or "is provable". Autoepistemic logic can be thought of the logic of a belief of an agent that reasons about its own beliefs. Given a set of initial beliefs (a knowledge-base) of the agent the logic describes what is the final set of all beliefs the agent may obtain by inference and introspection. The set of all beliefs corresponds to theorems of a theory. Because the presence of some beliefs may depend on the lack of others the definition of valid inference of autoepistemic logic is cyclic in a specific sense, and cannot be formalized using a set of axioms and inference rules like in classical logics. The sets of theorems of a theory in autoepistemic logic are defined using fixpoint equations instead. A characteristic property of autoepistemic logic is that the solutions of the fixpoint equations are not unique, i.e., there may be several alternative sets of theorems or even none, each solution intuitively corresponding to a possible set of beliefs of an autoepistemic agent.

The forms of nonmonotonic reasoning discussed earlier in this section can often be seen as specific cases of reasoning in autoepistemic logic (or other nonmonotonic formalisms like default logic, circumscription, etc.). For example, logic programs with the negation-by-failure operator *not* can be seen as theories of autoepistemic logic by the following translation [Gelfond and Lifschitz, 1988].

$$c \leftarrow b_1, \ldots, b_m, \text{not } a_1, \ldots, \text{not } a_n \implies \neg La_1 \wedge \cdots \wedge \neg La_n \wedge b_1 \wedge \cdots \wedge b_m \rightarrow c \tag{2}$$

where the operators $\neg L$ correspond to the operator *not*. As the monotonicity of classical logics can be explained by the monotonicity of their inference rules, the introduction of nonmonotonic inference rules could lead to nonmonotonicity and the possibility of default inference. This is the approach of default logic [Reiter, 1980]. The form of the nonmonotonic inference rules in theories of default logic and their translation to formulae of autoepistemic logic are as follows [Konolige, 1988].

$$\frac{\alpha : \beta}{\gamma} \implies L\alpha \wedge \neg L \neg \beta \rightarrow \gamma \tag{3}$$

The formula α above is the *prerequisite* of the default rule, β is the *justification*, and γ is the *consequence* of the rule. For a rule to be applicable the justification α has to be provable, the negation of the justification β must not be provable. If these conditions hold conclusion γ can be inferred. Also in default logic sets of consequences are defined by fixpoint equations that may have several solutions.

1.2.2 The Research Problem

The research problem we try to contribute to in this work is the derivation of expressive and efficient knowledge representation languages based on nonmonotonic logics. The logics themselves in the general cases are very expressive languages. They for instance subsume first-order predicate logic, and consequently their complexity and computability properties suggest that they cannot be used for knowledge representation in their most general form, at least not for applications that involve large knowledge bases. A knowledge representation language should have sufficient expressivity for most applications and it should be tractable, i.e., it should have decision procedures of relatively low time-complexity, preferably of low polynomials. The distinction between tractable and intractable problems is the following one usually made in complexity theory [Garey and Johnson, 1979]: if a problem has a polynomial time decision procedure, then it is tractable, otherwise it is intractable.

The problem of identifying tractable sublanguages of nonmonotonic logics has been addressed by many researchers recently, the idea being that a sufficient restriction of the syntactic form of the formulae automatically reduces the complexity of the decision procedures to polynomial time. Kautz and Selman [1991] and Stillman [1990] show that in the framework of default logic this line of research does not yield the expected results, i.e., the decision procedures of only syntactically extremely restricted default theories are tractable. It seems that the reason for this is that in the definition of valid inference a reference to unprovability is made (e.g. in default logic and autoepistemic logic), and due to this there is certain kind of cyclicity in the definition of valid inference, the resolution of which in general requires exponential time computation in the decision procedures. This resolution task is significantly easier if the acyclicity condition of *stratification* is employed. Stratification has not been investigated very thoroughly in the context of nonmonotonic logics, and its actual impact on the complexity of reasoning has been unknown. In this work we show that under the stratification condition, tractability in several nonmonotonic formalisms can be achieved with substantially less serious syntactical restrictions than in the more general unstratified case.

1.2.3 Stratification

Stratification was first introduced by Chandra and Harel [1985] in an article discussing the expressivity of various kinds of query languages for deductive databases. Its more general importance for deductive databases and logic programming was recognized by [Apt *et al.*, 1988; Van Gelder, 1988]. The idea of stratification is that before the negation of a fact can be referred to, the fact has to be defined first. When this requirement is analyzed the following definition results. A logic program is *stratified* if its clauses can be partitioned into strata or levels so that negated subgoals in the bodies of the clauses on a certain level refer to predicates defined on lower levels only. For example the program {watches_tv(X)←not blind(X), blind(anne)} is stratified because its clauses can be partitioned into levels P_1, P_2 in the following way.

$$P_2 = \{\ \text{watches\_tv(X)} \leftarrow \text{not blind(X)}\ \}$$
$$P_1 = \{\ \text{blind(anne)}\ \}$$

The stratification condition is fulfilled, because the reference to the predicate *blind* inside the negation operator *not* is on the level P_2, and the predicate *blind* is defined on the level P_1 which is below P_2. The following program cannot be so partitioned.

male(X) ← not female(X)
female(X) ← not male(X)

For this program the stratification condition requires that the clause defining the predicate *male* is on a strictly higher level in the stratification than the clause that defines *female*, and vice versa, and hence the condition cannot be satisfied.

It is not clear what answer the above program should give to the query *male(Pertti)*. More generally, the semantics of unstratified programs is a controversial issue, i.e., there are several competing semantics for logic programs in the general case each with some intuitive force. The leading general semantics are the stable model semantics [Gelfond and Lifschitz, 1988] and the well-founded semantics [Van Gelder *et al.*, 1988]. The well-founded semantics is three-valued, i.e., for some unstratified programs some facts are assigned the truth-value *unknown*. Stable model semantics is two-valued, and for unstratified programs it may give several models or none. The well-founded semantics and stable models are incomparable, i.e., for both of them there are examples for which the other gives more plausible results.

The class of stratified logic programs is much less problematic. As in the case of logic programs without the *not* operator, the set of correct conclusions of a stratified logic program can be defined by a natural fixpoint construction [Apt *et al.*, 1988]. The perfect model semantics [Przymusinski, 1988] of stratified logic programs coincides with the fixpoint construction, as do the stable model semantics and well-founded semantics. Hence all major semantics agree on the meaning of stratified programs. A good discussion of this subject and the above semantics can be found in [Przymusinska and Przymusinski, 1990].

Stratification conditions for nonmonotonic logics based on the similar concept for logic programs have been presented e.g. in [Bidoit and Froidevaux, 1991; Gelfond, 1987; Marek and Truszczyński, 1991]. The definitions of stratification for autoepistemic logic and some of the definitions for default logic are sufficient conditions for the uniqueness of stable expansions and extensions. In this work we show that they also give the decision procedures an immediate way of resolving the conflicts caused by the nonconstructive cyclic definition of provability.

1.3 Outline of the Report

Basic definitions of autoepistemic logic that will be needed in the body of the report are presented in Section 2. In Section 3 a notion of stratification for autoepistemic logic is discussed, a decision procedure for stratified autoepistemic theories is presented and its complexity is analyzed, and finally two tractable classes of autoepistemic theories are identified on the basis of the complexity analysis. The implications of the complexity results to other nonmonotonic formalisms are discussed in Section 4. Comparison to some earlier complexity results on the tractability of nonmonotonic reasoning is made in the same section. An implementation of the decision procedure for stratified theories is described in Section 5. Finally, the results of this report are briefly discussed in Section 6.

2 Autoepistemic Logic

Autoepistemic logic is a nonmonotonic logic of belief introduced by [Moore, 1985]. The language $\mathcal{L}_{ae}$ of autoepistemic logic is the language of the classical propositional (or quantificational) logic $\mathcal{L}$ extended with the monadic operator L which is read "is believed that". The logic describes the reasoning capabilities of an ideally rational agent. The agent is logically omniscient, i.e., it believes all logical consequences of its own beliefs, and it is also capable of introspection, i.e., for each proposition ϕ either $L\phi$ or $\neg L\phi$ is its belief depending on whether ϕ is its belief or not. The description of the ideally rational agent was first presented by Stalnaker [1980] in a commentary on the nonmonotonic logic of McDermott and Doyle. Later Moore [1983] used Stalnaker's ideas to introduce an alternative to McDermott and Doyle's logic. The description of the ideally rational agent yields three *stability* conditions on the sets of beliefs of the agent.

1. If $\phi_1, \ldots, \phi_n$ are beliefs and $\phi_1, \ldots, \phi_n \models \psi$, then ψ is a belief.
2. If ϕ is a belief, then $L\phi$ is a belief.
3. If ϕ is not a belief, then $\neg L\phi$ is a belief.

The symbol $\models$ is the logical consequence relation of propositional logic extended to formulae that start with the L operator. These formulae are treated exactly like propositional variables.

An autoepistemic agent bases its beliefs on a set of formulae Σ called the *initial premises*. Now we can define that an autoepistemic theory T is *grounded* in the initial premises Σ if T is contained in

$$\{\phi|\Sigma \cup \{L\phi|\phi \in T\} \cup \{\neg L\phi|\phi \notin T \models \phi\}. \tag{4}$$

A stable set T grounded in premises Σ is a *stable expansion* of Σ if it does not contain other premises than required by the stability and groundedness condition, i.e., it satisfies the following equation.

$$T = \{\phi|\Sigma \cup \{L\phi|\phi \in T\} \cup \{\neg L\phi|\phi \notin T\} \models \phi\} \tag{5}$$

Stable expansions are the primary objects of study in autoepistemic logic. Moore [1985] gives also a semantical characterization of stable expansions. *An autoepistemic interpretation* I of an autoepistemic theory $T \subseteq \mathcal{L}_{ae}$ is a pair $\langle v, T\rangle$, where $v : P \rightarrow \{\top, \bot\}$ is a valuation of the propositional variables P, and $T \subseteq \mathcal{L}_{ae}$. A truth definition for formulae of $\mathcal{L}_{ae}$ is given in the following way.

$$
\begin{array}{lll}
\langle I,T\rangle \models p & \text{iff} & v(p)=\top \\
\langle I,T\rangle \models L\phi & \text{iff} & \phi \in T \\
\langle I,T\rangle \models \neg\phi & \text{iff} & \langle I,T\rangle \not\models \phi \\
\langle I,T\rangle \models \phi\wedge\psi & \text{iff} & \langle I,T\rangle \models \phi \text{ and } \langle I,T\rangle \models \psi
\end{array}
$$

Autoepistemic interpretations are the model construction that we will use in later sections where they will be called simply *models*.

Definition 2.1 ([Moore, 1985]) *An* autoepistemic model *of T is an autoepistemic interpretation of T in which all formulae in T are true.*

Definition 2.2 ([Moore, 1985]) *An autoepistemic theory T is* semantically complete *iff T contains every formula that is true in every autoepistemic model of T.*

Theorem 2.3 ([Moore, 1985]) *An autoepistemic theory T is semantically complete iff it is stable.*

Definition 2.4 ([Moore, 1985]) *An autoepistemic theory T is* sound *with respect to a set of premises A iff every autoepistemic interpretation of T that is model of A is also a model of T.*

Theorem 2.5 ([Moore, 1985]) *An autoepistemic theory T is sound with respect to a set of premises A iff T is grounded in A.*

By Theorems 2.3 and 2.5 we see that a theory is a stable expansion exactly when it is both semantically complete and sound with respect to the initial premises Σ.

A set of initial premises Σ does not in general have a unique stable expansion. One could think that the fixpoint operator related to Equation 5 would conveniently have a least solution that could be found by iteration (by the famous fixpoint theorem of Tarski). The operator, however, is not monotonic because of its appeal to unprovability. In general, the equation can have none, one, or several solutions depending on the set Σ.

Example 2.1 The set of initial premises $\Sigma = \{\neg L\phi \rightarrow \phi\}$ has no stable expansions. Suppose there is a stable expansion Δ_1 of Σ and $\phi \notin \Delta_1$. Now by the stability conditions $\neg L\phi \in \Delta$ and further with $\neg L\phi \rightarrow \phi$ by logical consequence we obtain ϕ, a contradiction. Suppose then that there is a stable expansion Δ_2 of Σ and $\phi \in \Delta_2$. This can only be if ϕ is a logical consequence of Δ, but this requires that $\neg L\phi \in \Delta$ which by stability conditions contradicts $\phi \in \Delta$. ∎

Example 2.2 Sets of initial premises Σ that do not contain the L operator have exactly one stable expansion [Moore, 1985]. ∎

Example 2.3 The set $\{L\phi \rightarrow \phi\}$ has two stable expansions. If ϕ is in the stable expansion then also $L\phi$ is and ϕ follows. If ϕ is not in the stable expansion then neither is $L\phi$, and ϕ does not follow. ■

Because of the possibility of several stable expansions, the question of what is meant by correct reasoning in autoepistemic logic arises. The notions of *brave* and *cautious* reasoning have been introduced in this context, denoted by the relation symbols $\models_b$ and $\models_c$, respectively. $\Sigma \models_b \phi$ if ϕ is contained in every stable expansion of Σ. $\Sigma \models_c \phi$ if ϕ is contained in at least one stable expansion of Σ.

2.1 Finite Characterizations of Stable Expansions

For computational purposes like theorem-proving stable expansions cannot directly be used because they are infinite sets. Niemelä [1990] presents a useful characterization of stable expansions which is of finite nature. Stable expansions contain large amounts of redundant information that does not have to be represented explicitly. The relevant subset of a stable expansion of a set Σ is the set $Sf^L(\Sigma)$ of subformulae of Σ that begin with the L operator. $Sf^{qL}(\Sigma)$ is the subset of $Sf^L(\Sigma)$ of those subformulae of Σ beginning with L that are not in the scope of an L operator.

Definition 2.6 ([Niemelä, 1990]) *A set of formulae Λ is Σ-full if it satisfies the following conditions.*

$$\Lambda \subseteq Sf^L(\Sigma) \cup \{\neg L\chi | L\chi \in Sf^L(\Sigma)\} \tag{6}$$

$$\text{for all } L\chi \in Sf^L(\Sigma),\ L\chi \in \Lambda \text{ iff } \Sigma \cup \Lambda \models \chi \tag{7}$$

$$\text{for all } L\chi \in Sf^L(\Sigma),\ \neg L\chi \in \Lambda \text{ iff } \Sigma \cup \Lambda \not\models \chi \tag{8}$$

Theorem 2.7 ([Niemelä, 1990]) *For a set of sentences Σ there is a bijective mapping from the Σ-full sets to the stable expansions of Σ.*

If in Σ there are only finitely many different subformulae beginning with an L, then the respective Σ-full sets are finite. This is the case e.g. when Σ is finite. The following definition and theorem describe how the membership in a stable expansion of a set of premises Σ can be determined in terms of a Σ-full set Λ.

Definition 2.8 ([Niemelä, 1990]) *Given a set of sentences Σ and a sentence ϕ*

$$\Sigma \models_L \phi \text{ iff } \Sigma \cup SB_\Sigma(\phi) \models \phi$$

where $SB_\Sigma(\phi) = \{L\chi \in Sf^{qL}(\phi) | \Sigma \models_L \chi\} \cup \{\neg L\chi | L\chi \in Sf^{qL}(\phi)$ *and* $\Sigma \not\models_L \chi\}$.

Theorem 2.9 ([Niemelä, 1990]) *Let* Λ *be a* Σ*-full set. Then* $SE_\Sigma(\Lambda) = \{\phi | \Sigma \cup \Lambda \models_L \phi\}$ *is the unique stable expansion of* Σ *for which* $\Lambda \subseteq \{L\phi | \phi \in SE_\Sigma(\Lambda)\} \cup \{\neg L\phi | \phi \notin SE_\Sigma(\Lambda)\}$.

The above definitions and theorems are the basis of the results developed in Section 3.

Example 2.4 Consider the set of initial premises $\Sigma = \{\neg L\phi \rightarrow \psi, \neg L\psi \rightarrow \phi\}$. It has four candidate Σ-full sets, two of which are in fact Σ-full. The set $\Lambda_1 = \{\neg L\phi, L\psi\}$ is Σ-full because $\Sigma \cup \Lambda_1 \not\models_L \phi$ and $\Sigma \cup \Lambda_1 \models_L \psi$. By symmetry also $\Lambda_2 = \{\neg L\psi, L\phi\}$ is Σ-full. ■

2.2 Computability and Complexity

In this section we present some of the most important decidability and complexity results related to autoepistemic logic. An introduction to the concepts of computational complexity can be found in [Balcázar *et al.*, 1988; Garey and Johnson, 1979; Johnson, 1990].

Tractability of decision problems of languages is usually identified with polynomial time solvability. Languages the decision problems of which are solvable by a deterministic Turing machine in polynomial time belong to the class P and those for which decision problems are solvable by a nondeterministic Turing machine in polynomial time belong to the class NP. The hardest languages in NP to which all other languages in NP reduce in polynomial time are the NP-complete languages, of which the language of satisfiable propositional formulae is an example. It is not known for certain that NP strictly extends P, i.e. there is the possibility that all languages in NP can be solved in polynomial time. This is however considered very unlikely.

The polynomial hierarchy is an infinite hierarchy of complexity classes Σ_i^P, Π_i^P, and $\Delta_i^P, i \geq 0$. It is defined in the following way.

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = P \tag{9}$$

$$\Sigma_{n+1}^P = \mathrm{NP}^{\Sigma_n^P},\ \Pi_{n+1}^P = \text{co-}\Sigma_{n+1}^P,\ \Delta_{n+1}^P = P^{\Sigma_n^P} \tag{10}$$

$C_1^{C_2}$ denotes the class of languages that are in C_1 if given an oracle for the problems in C_2, and co-C denotes the set of languages whose complements are in the class C. Note that $\Sigma_1^P = NP$. Whether each class in the polynomial hierarchy contains languages not contained in the classes below, is not known, but the collapse of the polynomial hierarchy is even less probable than the equivalence of P and NP. Therefore, complete languages for a class high in the polynomial hierarchy which are at least as complex as all other languages in that class, are considered strictly more complex than e.g. NP-complete languages.

The following results presented in [Gottlob, 1991] suggest that in the general case decision problems of propositional autoepistemic logic are highly intractable.

Theorem 2.10 ([Gottlob, 1991]) *The problem of existence of stable expansions for a set of premises Σ is Σ_2^P-complete.*

The problem of brave reasoning, i.e., whether a given formula belongs to at least one stable expansion of Σ, is Σ_2^P-complete.

The problem of cautious reasoning, i.e., whether a given formula belongs to all stable expansions of Σ, is Π_2^P-complete.

For propositional default logic similar results are presented in [Gottlob, 1991; Stillman, 1992]: existence of extensions, brave reasoning, and cautious reasoning are Σ_2^P-complete, Σ_2^P-complete, and Π_2^P-complete, respectively. These results are quite expected as the connection between these two logics is very close.

Autoepistemic logics based on undecidable logics and even the ones based on semidecidable logics are not decidable and not even semidecidable [Niemelä, 1990]. For example, the quantificational version of autoepistemic logic which is based on first-order predicate logic which is semidecidable, is undecidable.

3 Reasoning in Stratified Autoepistemic Logic

In this section we present the main results of the report. The results are about the tractability of reasoning in stratified autoepistemic theories. First the stratification condition we use is discussed and an algorithm for computing stratifications is presented. Then general algorithms for solving the decision problems of stratified autoepistemic theories are given. Stratification is a structural restriction on whole theories, and if also further syntactical restrictions on the formulae in the theories are made it is possible to achieve polynomial time decision procedures. As an example of this we present two tractable classes of stratified autoepistemic theories which are based on the Horn clause subset of classical propositional logic. The satisfiability of sets of propositional Horn clauses can be tested in linear time.

In Section 4 we present how the results of this section can be used in achieving tractable reasoning in other nonmonotonic formalisms, e.g. in default logic, propositional logic programs, and truth maintenance systems.

3.1 Stratification

Marek and Truszczyński define a notion of a stratified set for propositional autoepistemic logic that is a generalization of an earlier definition of [Gelfond, 1987].

Definition 3.1 ([Marek and Truszczyński, 1991]) *A set of formulae Σ is stratified if*

1. *The formulae $\phi \in \Sigma$ are of the form $a(\phi) \wedge o(\phi) \rightarrow c(\phi)$, where subformulae $o(\phi)$ and $c(\phi)$ do not contain the L operator and $o(\phi)$ may be missing, and $a(\phi)$ is a formula of the form $L\phi_1 \wedge \cdots \wedge L\phi_r \wedge \neg L\psi_1 \wedge \cdots \wedge \neg L\psi_s$ where $r, s \geq 0$.*
2. *The set $\{c(\phi) | \phi \in \Sigma\}$ is satisfiable.*
3. *There exists a set of indices $I = \{1, \ldots, n\}$ or $I = \{1, \ldots\}$ and a partition of $\Sigma = \bigcup_{i \in I} \Sigma_i$ such that for all $j \in I$ the propositional variables occurring in $\{c(\phi) | \phi \in \Sigma_j\}$ do not occur in Σ_j in the scope of an L operator or in Σ_1^{j-1} (where $\Sigma_a^b = \bigcup_{i=a}^{b} \Sigma_i$).*

The reasoning methods developed in this section use the stratifications, i.e., the partitions of the formulae in the stratified sets. In the general case the computation of a stratification or testing whether a set of autoepistemic formulae is stratified is exponential time because of the satisfiability testing of

$\{c(\phi)|\phi \in \Sigma\}$. In the polynomial time classes developed in this section Condition 2 of stratification can be tested in linear time. For testing Condition 3 efficiently Marek and Truszczyński define the notion of *a-stratifiedness* which coincides with stratifiedness. A finite set Σ is a-stratified if it satisfies conditions 1 and 2 of stratification and there is a partition $P_1, \ldots, P_n$ of the propositional variables in Σ that fulfills the following condition. For each pair of variables $p \in P_i, q \in P_j$ such that p occurs in $a(\phi)$ and q in $c(\phi)$ for some $\phi \in \Sigma$, $i < j$, and for each pair of variables $p \in P_i, q \in P_j$ such that p occurs in ϕ and q in $c(\phi)$ for some $\phi \in \Sigma$, $i \leq j$.

Theorem 3.2 ([Marek and Truszczyński, 1991]) *A finite set of formulae Σ is stratified if and only if Σ is a-stratified.*

The test for the existence of the partition of variables can easily be reduced to the computation of the strong components of a graph. The *characteristic graph* of a set of formulae is a graph representing the constraints imposed on the partition of the propositional variables by the definition of a-stratifiedness. In the characteristic graph there is an edge from the variable p to the variable q if for some $\phi \in \Sigma$ p occurs in $c(\phi)$ and q anywhere in ϕ. The edge is an L-edge if there is an occurrence of q in $a(\phi)$.

Theorem 3.3 ([Marek and Truszczyński, 1991]) *A finite set of formulae Σ is a-stratified if and only if Σ satisfies Conditions 1 and 2, and no strong component of its characteristic graph contains an L-edge.*

Example 3.1 The set

$$r \wedge \neg Lp \rightarrow h$$
$$q \wedge \neg Lh \rightarrow p$$

is not a-stratified and consequently it is not stratified. The characteristic graph has three strong components. The variables r and p occupy singleton strong components, and since there are edges both from h to p and from p to h, p and h are in the same strong component. The set is not a-stratified – and consequently not stratified – because the edges between p and h are L-edges. ■

Example 3.2 The set

$$L\neg Lr \wedge p \rightarrow q$$
$$q \rightarrow p$$
$$L(p \leftrightarrow q) \rightarrow s$$

is a-stratified. Variables p and q belong to the same component and there are no L-edges between them, s and r both occupy a singleton component. ■

For the tractable classes of autoepistemic logic investigated in this section, we present a linear time algorithm that computes a stratification if the set is stratified, and for sets that are not, detects this fact. The algorithm is based on Theorem 3.3. Marek and Truszczyński [1991] sketch a similar algorithm for arbitrary sets of autoepistemic formulae that fulfill Condition 1. Another algorithm for computing a stratification which is based on strong components is presented in [Lassez *et al.*, 1987].

In the general case the size of the characteristic graph is quadratic on the size of Σ, and consequently the traversal of the graph for finding the strong components takes quadratic time. However, if the form of the subformulae $c(\phi), \phi \in \Sigma$ is restricted the computation becomes linear time.

Linear time complexity results in this section, e.g. the linearity of our stratification algorithm, rest on the following assumption.

Assumption 3.4 *Each propositional variable is assigned a unique number so that data structures with constant access time (arrays) can be used for storing various data related to them.*

Taking a set of formulae Σ as input and assigning a number for each propositional variable can be done in $\mathcal{O}(n \log v)$ time where n is the size of Σ and v is the number of distinct propositional variables in Σ. LL(1) grammars can be parsed in $\mathcal{O}(n)$ time, but construction of symbol tables of v symbols for inputs of length n is $\mathcal{O}(n \log v)$.

Proposition 3.5 *Let Σ be a set of formulae that fulfills Condition 1 of stratification, and for each $\phi \in \Sigma$ there are occurrences of at most one propositional variable in $c(\phi)$. Under Assumption 3.4 the computation of a stratification for Σ or detecting that Σ is not stratified is $\mathcal{O}(|\Sigma|)$ time.*

In this restricted case Condition 2 of the definition of a stratified set can be tested in $\mathcal{O}(|\Sigma|)$ time. Because each $c(\phi)$, $\phi \in \Sigma$ contains occurrences of only one propositional variable p, each $c(\phi)$ is equivalent to either p, $\neg p$, $\top$, or $\bot$ as there are exactly four truth functions of one variable. The reduction of all $c(\phi), \phi \in \Sigma$ to one of these formulae is $\mathcal{O}(|\Sigma|)$ time because for each ϕ it can be done in one traversal of the formula tree of $c(\phi)$. The satisfiability of a set of formulae of the form p, $\neg p$, $\bot$, and $\top$ can be tested in linear time on the size of the set. By Theorem 3.3 testing Condition 3 of stratification can be implemented as a computation of the strong components of the characteristic graph together with detection of L-edges inside the strong components. For this purpose we give a variant of Tarjan's [1972] well-known algorithm for the strong components of a graph as presented in [Aho *et al.*, 1974]. Tarjan's algorithm runs in $\mathcal{O}(n + e)$ time where n is the number of nodes and e is the number of edges, and it has the useful property that the strong components are produced in an order that qualifies as a stratification, i.e., the first component the algorithm emits consists of the variables in $c(\phi)$ for formulae $\phi \in \Sigma_1$ that can be taken as the lowest stratum of a stratification Σ_1^n, and so on.

The size of the characteristic graph is linear on the size of Σ because for each $\phi \in \Sigma$ there are occurrences of at most one propositional variable in $c(\phi)$ and consequently there is at most one edge for each variable occurrence in $\{a(\phi) \wedge o(\phi) | \phi \in \Sigma\}$.

For the computation of the strata the following arrays and variables are needed.

formulae[p] the list of formulae ϕ in which the variable p appears in $c(\phi)$. This array can be initialized in linear time by traversing the set of formulae once.

edges[p] the list of variables q that appear in $a(\phi) \wedge o(\phi)$ for a formula ϕ in which the variable p appears in $c(\phi)$. The initialization can be done in linear time. First initialize the elements to empty lists. Then for each p the list **formulae**[p] is traversed and for each occurrence of a variable q an auxiliary array of flags is tested whether q already is in **edges**[p]. If not, it is added in the head and the auxiliary array is updated.

l-edges[p] the list of variables q that appear in $a(\phi)$ for a formula ϕ in which the variable p appears in $c(\phi)$. This array is initialized the same way as **edges**.

ccount a counter for the strong components. Initialized to zero.

component[i] the list of formulae in stratum i.

stratum[p] the number of the stratum to which formulae ϕ having p in $c(\phi)$ belong.

The following variables are part of the original strong components algorithm (see [Aho *et al.*, 1974] for details):

count a counter for numbering the nodes of the graph in the order of depth-first traversal.

dfnumber[p] the number assigned to the node p during depth-first traversal.

lowlink[p] an array which is used for recognizing strong components during the traversal.

SEARCHC is called repeatedly for unmarked variables until all variables are marked *old*. If the error NOT_STRATIFIED is signalled, then Condition 3 cannot be fulfilled and the set is not stratified.

The first half of the procedure is responsible for the traversal of the characteristic graph depth-first and maintenance of the data structures for detection of the strong components. All our modifications are in the *if* statement that forms the second half of the procedure. First, a new element is reserved in the array *components* for the formulae in the newly found strong component. The *repeat* loop assigns each variable in the strong component the number of the component. Finally the *while* loop tests the component for the containment of an L-edge using the numbers assigned to variables, and concatenates to the array *components* the list of formulae belonging to the stratum.

We show that the running time of the algorithm is within bounds proportional to the size of the set of formulae Σ. First note that everything that can be done within $\mathcal{O}(n+e)$ is within the bounds because the number of nodes, i.e. propositional variables, is bounded by the set of formulae and the so is the number of edges which is the number of variable occurrences in the antecedents of formulae.

SEARCHC is called exactly once for each node in the graph and each call (the recursive call to SEARCHC excluded) takes constant time plus time proportional to the number of edges leaving from the node (the second half of the procedure is analyzed below). For the whole run of the algorithm this is $\mathcal{O}(n+e)$. All computation in the second half of the procedure during the *whole* computation of the algorithm is bounded by the size of the set of formulae. The update of *ccount* and initialization of the *components*[*ccount*] and the stack are constant time operations and they are performed for each strong component found. In one run of the algorithm exactly one iteration of the *repeat* loop is executed for each node of the graph, and this is $\mathcal{O}(|\Sigma|)$. The tests for containment of an edge through an L operator are also within the $\mathcal{O}(|\Sigma|)$ bound because for each propositional variable p the presence of L-edges inside the stratum of p is tested exactly once, and the number of elements in the lists of the array *l-edges* is linearly bounded by the size of the set of formulae. Constructing the lists of the array *components* is $\mathcal{O}(|\Sigma|)$ since each formula belongs to exactly one stratum and we restrict $c(\phi)$ for each ϕ to contain occurrences of at most one propositional variable.

Example 3.3 Our algorithm computes for the set in Example 3.2 the stratification shown below. The first column contains the numbers of the strata, the second contains the variables in the corresponding strong components of the associated characteristic graph, and the third the strata, i.e., the sets of formulae ϕ in which the variables of the respective strong component occur in $c(\phi)$.

```
procedure SEARCHC(v);
begin
  mark v "old"
  dfnumber[v] := count;
  count := count + 1;
  lowlink[v] := dfnumber[v];
  push v on stack;
  for each vertex w on edges[v] do
    if w is marked "new" then
      SEARCHC(w);
      lowlink[v] := min(lowlink[v],lowlink[w]);
    else
      if dfnumber[w] < dfnumber[v] and w is on stack
        then lowlink[v] := min(dfnumber[w],lowlink[v]); end if
    end if
  end for
  if lowlink[v] = dfnumber[v] then
    ccount := ccount + 1;
    components[ccount] := empty_list;
    initialize stack2 to empty;
    repeat
      pop x from top of stack;
      push x to stack2;
      stratum[x] := ccount;
    until x=v;
    while stack2 not empty do
      pop x from stack2;
      foreach y in l-edges[x] do
        if stratum[x] = stratum[y]
          then signal NOT_STRATIFIED;
        end if
      end for
      concatenate formulae[x] to components[ccount];
    end while
  end if
end
```

Figure 3: The Main Procedure of an Algorithm for Computing a Stratification

3	s	$L(p \leftrightarrow q) \rightarrow s$
2	p, q	$q \rightarrow p, L\neg Lr \wedge p \rightarrow q$
1	r	$\emptyset$

Because there are no formulae ϕ with r in $c(\phi)$ the lowest stratum is empty, and can be ignored. ■

3.2 A Decision Procedure for Stratified Theories

In this section is we present algorithms using which the membership of arbitrary formulae in the unique stable expansions of stratified sets can be decided. First we give a general algorithm for computing Σ-full sets of stratified theories. It turns out that the explicit construction of the Σ-full sets is unnecessary and the membership in the unique stable expansion can be decided from a set of objective formulae instead, i.e., formulae with no occurrences of the L operator.

The next theorem gives an iterative algorithm for computing the Σ-full set of an arbitrary stratified set Σ. This algorithm can be seen as a special case of the backtracking algorithm for computing Σ-full sets of autoepistemic theories suggested by the finitary characterization of stable expansions of [Niemelä, 1990] presented in Section 2.1. The iterativeness is obviously due to the partition of the set of formulae to a number of strata. Because of the results of Marek and Truszczyński and on the other hand of Niemelä, the Σ-full set characterizes the unique stable expansion of Σ.

Theorem 3.6 *Let $\Sigma = \Sigma_1^n$ be a stratified set. Then $\Lambda = \Lambda_n$ defined by*

$$\begin{aligned}
\Lambda_0 &= \emptyset \\
\Lambda_{i+1} &= \Lambda_i \cup \{L\chi | L\chi \in Sf^L(\Sigma_{i+1}) - Sf^L(\Sigma_1^i), \Sigma_1^i \cup \Lambda_i \models_L \chi\} \\
&\quad \cup \{\neg L\chi | L\chi \in Sf^L(\Sigma_{i+1}) - Sf^L(\Sigma_1^i), \Sigma_1^i \cup \Lambda_i \not\models_L \chi\}, 0 \leq i < n
\end{aligned}$$

is Σ-full and $SE_\Sigma(\Lambda)$ is the unique stable expansion of Σ.

For proving Theorem 3.6 the following lemma is essential. It is the most important piece for the correctness of the algorithm in the theorem, since it accounts for the monotonicity property that allows the conclusion of underivability of formulae, and hence an iterative instead of a backtracking algorithm.

Lemma 3.7 *Let $\Sigma = \Sigma_1^n$ be stratified and Λ_i as in Theorem 3.6. Then for all $i, 0 \leq i < n$ and for all $L\chi \in Sf^L(\Sigma_{i+1})$, $\Sigma_1^i \cup \Lambda_i \models_L \chi$ iff $\Sigma_1^j \cup \Lambda_j \models_x \chi$ where $\models_x$ is $\models$ or $\models_L$, and $i < j \leq n$.*

Proof: By induction on i.

$(i = 0)$. The proof that for all $L\chi \in Sf^L(\Sigma_1)$, $\Sigma_1^0 \cup \Lambda_i \models_L \chi$ iff $\Sigma_1^j \cup \Lambda_j \models_x \chi$ is by induction on the L-depth s of χ. The L-depth of a formula is the maximum nesting of L operators in it.

$(s = 0)$. For all $L\chi \in Sf^L(\Sigma_1)$ with L-depth 0,

$$\begin{aligned} \Sigma_1^0 \cup \Lambda_0 \models_L \chi \quad & \text{iff} \quad \Sigma_1^0 \cup \Lambda_0 \models \chi && (11)\\ & \text{implies} \quad \Sigma_1^j \cup \Lambda_j \models \chi && (12)\\ & \text{iff} \quad \Sigma_1^j \cup \Lambda_j \models_L \chi. && (13) \end{aligned}$$

Equivalence 11 is because $SB_\Delta(\chi) = \emptyset$ for χ with no L operators, and consequently $\models$ coincides with $\models_L$ for χ. Implication 12 is because $\models$ is monotonic and Σ_1^0 and Λ_0 are empty by definition. Equivalence 13 is again because $\models$ coincides with $\models_L$.

To establish the equivalence between $\Sigma_1^0 \cup \Lambda_0 \models_L \chi$ and $\Sigma_1^j \cup \Lambda_j \models_x \chi$ we have to prove the converse of implication 12 above. This is proved by showing that $\emptyset \not\models \chi$ implies $\Sigma_1^j \cup \Lambda_j \not\models \chi$. Assume $\emptyset \not\models \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \not\models \chi$. We show that then also $\Sigma_1^j \cup \Lambda_j \not\models \chi$ holds by giving a model $\mathcal{M}'$ such that $\mathcal{M}' \models \Sigma_1^j \cup \Lambda_j$ and $\mathcal{M}' \not\models \chi$. The model $\mathcal{M}'$ can be obtained by modifying $\mathcal{M}$ to satisfy $\Sigma_1^j \cup \Lambda_j$. We show that this modification is possible and that it does not make χ true in $\mathcal{M}'$. The set Σ_1^j can be made true in $\mathcal{M}'$ by making the formulae $\{c(\phi)|\phi \in \Sigma_1^j\}$ true. This set of formulae is satisfiable by the second condition of stratification. The truth-value of χ is not affected because by the third condition of stratification the set of propositional variables in $L\chi \in Sf^L(\Sigma_1)$ is disjoint from the variables in $\{c(\phi)|\phi \in \Sigma_1^j\}$. Making the formulae Λ_j true does not affect the valuation of χ or $\{c(\phi)|\phi \in \Sigma_1^j\}$ because the valuation of formulae beginning with L is disjoint from propositional variables. Λ_j is satisfiable because by construction it contains no formula and its negation. The exact modification is $T' = (T \cup \{\chi|L\chi \in \Lambda_j\}) - \{\chi|\neg L\chi \in \Lambda_j\}$ where $\mathcal{M}' = \langle I', T'\rangle$ and $\mathcal{M} = \langle I, T\rangle$.

$(s \geq 1)$. For all $L\chi \in Sf^L(\Sigma_1)$,

$$\begin{aligned} \Sigma_1^0 \cup \Lambda_0 \models_L \chi \quad & \text{iff} \quad \Sigma_1^0 \cup \Lambda_0 \cup SB_{\Sigma_1^0 \cup \Lambda_0}(\chi) \models \chi && (14)\\ & \text{iff} \quad SB_{\Sigma_1^0 \cup \Lambda_0}(\chi) \models \chi && (15)\\ & \text{iff} \quad SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \models \chi && (16)\\ & \text{implies} \quad \Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \models \chi && (17)\\ & \text{iff} \quad \Sigma_1^j \cup \Lambda_j \models \chi && (18)\\ & \text{iff} \quad \Sigma_1^j \cup \Lambda_j \models_L \chi. && (19) \end{aligned}$$

Equivalence 14 is by the definition of $\models_L$. Equivalence 15 is because by definition $\Sigma_1^0 \cup \Lambda_0$ is empty. Equivalence 16 is by Lemma **A** below. Implication 17 is by the monotonicity of $\models$. Equivalence 18 is because $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) = SB_{\Sigma_1^0 \cup \Lambda_0}(\chi)$ is contained in Λ_j as shown by Lemma **B** below. Equivalence 19 is by Lemma **B** and the definition of $\models_L$.

Lemma **A**. For all $j, 1 \leq j < n$ $SB_{\Sigma_1^i \cup \Lambda_i}(\chi) = SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$.
Formulae ϕ such that $L\phi \in Sf^{qL}(\chi)$ are in $Sf^L(\Sigma_1)$ as $L\chi \in Sf^L(\Sigma_1)$, thus by the induction hypothesis on s we get $\Sigma_1^0 \cup \Lambda_0 \models_L \phi$ iff $\Sigma_1^j \cup \Lambda_j \models_L \phi$, for formulae $L\phi \in Sf^{qL}(\chi)$. By the definition of SB the claim is immediate. □

Lemma **B**. $SB_{\Sigma_1^0 \cup \Lambda_0}(\chi) \subseteq \Lambda_j$.
Suppose $L\phi \in SB_{\Sigma_1^0 \cup \Lambda_0}(\chi)$, i.e. $L\phi \in Sf^{qL}(\chi)$ and $\Sigma_1^0 \cup \Lambda_0 \models_L \phi$. As $L\chi \in Sf^L(\Sigma_1)$ also $L\phi \in Sf^L(\Sigma_1)$. Now $L\phi \in \Lambda_1 \subseteq \Lambda_j, j \geq 1$ by the equations of Theorem 3.10. Similarly with $\neg L\phi$. □

To establish the equivalence between $\Sigma_1^i \cup \Lambda_i \models_L \chi$ and $\Sigma_1^j \cup \Lambda_j \models_x \chi$ we have to prove the converse of implication 17 above. This is proved by showing that $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$ implies $\Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$. Assume $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \models SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$ and $\mathcal{M} \not\models \chi$. We show that then also $\Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$ holds by giving a model $\mathcal{M}'$ such that $\mathcal{M}' \models \Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$ and $\mathcal{M}' \not\models \chi$. The model $\mathcal{M}'$ can be obtained by modifying $\mathcal{M}$ to satisfy $\Sigma_1^j \cup \Lambda_j$. We show that this modification is possible and that it does not make χ true in $\mathcal{M}'$. The set Σ_1^j can be made true in $\mathcal{M}'$ by making the formulae $\{c(\phi) | \phi \in \Sigma_1^j\}$ true. This set of formulae is satisfiable by the second condition of stratification. The truth-value of χ is not affected because by the third condition of stratification the set of propositional variables in $L\chi \in Sf^L(\Sigma_1)$ is disjoint from the variables in $\{c(\phi) | \phi \in \Sigma_1^j\}$. Making the formulae Λ_j true does not affect the valuation of $\{c(\phi) | \phi \in \Sigma_1^j\}$ because the valuation of formulae beginning with L is disjoint from propositional variables. Λ_j is satisfiable because by construction it contains no formula and its negation. The valuation of Λ_j does not conflict with the valuation of $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) = SB_{\Sigma_1^i \cup \Lambda_i}(\chi)$ because the latter is contained in Λ_j as shown by Lemma **B** above. Consequently, because the valuation of $L\chi \in Sf^L(\chi)$ does not change, also χ is not affected.

$(i \geq 1)$. Proof is again by induction on the L-depth s of χ.

$(s = 0)$. For all $L\chi \in Sf^L(\Sigma_{i+1})$ with L-depth 0,

$$\begin{aligned} \Sigma_1^i \cup \Lambda_i \models_L \chi \quad & \text{iff} \quad \Sigma_1^i \cup \Lambda_i \models \chi && (20)\\ & \text{implies} \quad \Sigma_1^j \cup \Lambda_j \models \chi && (21)\\ & \text{iff} \quad \Sigma_1^j \cup \Lambda_j \models_L \chi. && (22) \end{aligned}$$

Equivalence 20 is because $SB_\Delta(\chi) = \emptyset$ for χ with no L operators, and consequently $\models$ coincides with $\models_L$ for χ. Implication 21 is because $\Lambda_i \subseteq \Lambda_j, \Sigma_1^i \subseteq \Sigma_1^j$ for $i \leq j$, and $\models$ is monotonic. Equivalence 22 is again because $\models$ coincides with $\models_L$ for χ.

To establish the equivalence between $\Sigma_1^i \cup \Lambda_i \models_L \chi$ and $\Sigma_1^j \cup \Lambda_j \models_x \chi$ we have to prove the converse of implication 21 above. This is proved by showing that $\Sigma_1^i \cup \Lambda_i \not\models \chi$ implies $\Sigma_1^j \cup \Lambda_j \not\models \chi$. Assume $\Sigma_1^i \cup \Lambda_i \not\models \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \models \Sigma_1^i \cup \Lambda_i$ and $\mathcal{M} \not\models \chi$. We show that then also $\Sigma_1^j \cup \Lambda_j \not\models \chi$ holds by giving a model $\mathcal{M}'$ such that $\mathcal{M}' \models \Sigma_1^j \cup \Lambda_j$ and $\mathcal{M}' \not\models \chi$. The model $\mathcal{M}'$ can be obtained by modifying $\mathcal{M}$ to satisfy $\Sigma_{i+1}^j \cup \Lambda_j$. We show that this modification is possible and that it does not make χ true in $\mathcal{M}'$.

The set Σ_{i+1}^j can be made true in $\mathcal{M}'$ by making the formulae $\{c(\phi) | \phi \in \Sigma_{i+1}^j\}$ true. This set of formulae is satisfiable by the second condition of stratification. The truth-value of χ is not affected because by the third condition of stratification the set of propositional variables in $L\chi \in Sf^L(\Sigma_i)$ is disjoint from the variables in $\{c(\phi) | \phi \in \Sigma_{i+1}^j\}$. Making the formulae $\Lambda_j - \Lambda_i$ true does not affect the valuation of χ or $\{c(\phi) | \phi \in \Sigma_1^j\}$ because the valuation of formulae beginning with L is disjoint from propositional variables. $\Lambda_j - \Lambda_i$ is satisfiable because by construction it contains no formula and its negation.

$(s \geq 1)$. For all $L\chi \in Sf^L(\Sigma_{i+1})$,

$$
\begin{array}{rcll}
\Sigma_1^i \cup \Lambda_i \models_L \chi & \text{iff} & \Sigma_1^i \cup \Lambda_i \cup SB_{\Sigma_1^i \cup \Lambda_i}(\chi) \models \chi & (23)\\
& \text{iff} & \Sigma_1^i \cup \Lambda_i \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \models \chi & (24)\\
& \text{implies} & \Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \models \chi & (25)\\
& \text{iff} & \Sigma_1^j \cup \Lambda_j \models \chi & (26)\\
& \text{iff} & \Sigma_1^j \cup \Lambda_j \models_L \chi. & (27)
\end{array}
$$

Equivalence 23 is by the definition of $\models_L$. Equivalence 24 is by Lemma **C** below. Implication 25 is because $\Lambda_i \subseteq \Lambda_j, \Sigma_1^i \subseteq \Sigma_1^j$ for $i \leq j$, and $\models$ is monotonic. Equivalence 26 is because $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) = SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$ is contained in Λ_j as shown by Lemmata **C** and **D** below. Equivalence 27 is by Lemma **D** and the definition of $\models_L$.

Lemma **C**. For all $j, 0 \leq j < n$ $SB_{\Sigma_1^i \cup \Lambda_i}(\chi) = SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$.
Formulae ϕ such that $L\phi \in Sf^{qL}(\chi)$ are in $Sf^L(\Sigma_1)$ as $L\chi \in Sf^L(\Sigma_1)$, and thus by the induction hypothesis on s we get $\Sigma_1^i \cup \Lambda_i \models_L \phi$ iff $\Sigma_1^j \cup \Lambda_j \models_L \phi$, for formulae $L\phi \in Sf^{qL}(\chi)$. By the definition of SB the claim is immediate.
□

Lemma **D**. For all $i, j, i < j \leq n$, $SB_{\Sigma_1^i \cup \Lambda_i}(\chi) \subseteq \Lambda_j$.

Assume $L\phi \in SB_{\Sigma_1^i \cup \Lambda_i}(\chi)$.

1.	$\chi \in Sf^L(\Sigma_{i+1})$	assumption
2.	$L\phi \in SB_{\Sigma_1^i \cup \Lambda_i}(\chi)$	assumption
3.	$L\phi \in Sf^{qL}(\chi)$	2 and definition of SB
4.	$\Sigma_1^i \cup \Lambda_i \models_L \phi$	2 and definition of SB
5.	$L\phi \in Sf^L(\Sigma_{i+1})$	1 and 3
6.	$\exists k \leq i, L\phi \in (Sf^L(\Sigma_{k+1}) - Sf^L(\Sigma_1^k))$	5
7.	$\Sigma_1^k \cup \Lambda_k \models_L \phi$	4, 6, induction hypothesis on i
8.	$L\phi \in \Lambda_{k+1} \subseteq \Lambda_j$	equations of Theorem 3.10

Similarly with $\neg L\phi$. $\Box$

To establish the equivalence between $\Sigma_1^i \cup \Lambda_i \models_L \chi$ and $\Sigma_1^j \cup \Lambda_j \models_x \chi$ we have to prove the converse of implication 25 above. This is proved by showing that $\Sigma_1^i \cup \Lambda_i \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$ implies $\Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$. Assume $\Sigma_1^i \cup \Lambda_i \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \models \Sigma_1^i \cup \Lambda_i \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$ and $\mathcal{M} \not\models \chi$. We show that then also $\Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi) \not\models \chi$ holds by giving a model $\mathcal{M}'$ such that $\mathcal{M}' \models \Sigma_1^j \cup \Lambda_j \cup SB_{\Sigma_1^j \cup \Lambda_j}(\chi)$ and $\mathcal{M}' \not\models \chi$. The model $\mathcal{M}'$ can be obtained by modifying $\mathcal{M}$ to satisfy $\Sigma_{i+1}^j \cup \Lambda_j$.

We show that this modification is possible and that it does not make χ true in $\mathcal{M}'$.

The set Σ_{i+1}^j can be made true in $\mathcal{M}'$ by making the formulae $\{c(\phi) | \phi \in \Sigma_{i+1}^j\}$ true. This set of formulae is satisfiable by the second condition of stratification. The truth-value of χ is not affected because by the third condition of stratification the set of propositional variables in $L\chi \in Sf^L(\Sigma_i)$ is disjoint from the variables in $\{c(\phi) | \phi \in \Sigma_{i+1}^j\}$.

Making the formulae $\Lambda_j - \Lambda_i$ true does not affect the valuation of $\{c(\phi) | \phi \in \Sigma_1^j\}$ because the valuation of formulae beginning with L is disjoint from propositional variables. Λ_j is satisfiable because by construction it contains no formula and its negation.

The valuation of $\Lambda_j - \Lambda_i$ does not conflict with the valuation of $SB_{\Sigma_1^j \cup \Lambda_j}(\chi) = SB_{\Sigma_1^i \cup \Lambda_i}(\chi)$ because the latter is contained in Λ_j as shown by Lemma **D** above. Consequently, because the valuation of $L\chi \in Sf^L(\chi)$ does not change, also χ is not affected.

$\Box$

Proof of Theorem 3.6: Λ is Σ-full because it satisfies the conditions of Definition 2.6. Condition 1 is immediate, and Condition 3 is true if Condition 2 is, since by construction for all $L\chi \in Sf^L(\Sigma)$ exactly one of $L\chi$ or $\neg L\chi$

is in Λ. Condition 2 holds because $L\chi \in \Lambda$ if and only if there is $i < n$ for which $L\chi \in (Sf^L(\Sigma_{i+1}) - Sf^L(\Sigma_1^i))$ and $\Sigma_1^i \cup \Lambda_i \models_L \chi$. By Lemma 3.7 this is equivalent to $\Sigma_1^n \cup \Lambda \models \chi$. By Theorem 2.9 $SE_\Sigma(\Lambda)$ is a stable expansion of the set of premises Σ, and because Σ is stratified $SE_\Sigma(\Lambda)$ is the unique stable expansion of Σ by Theorem 5.1 of [Marek and Truszczyński, 1991]. $\square$

The following two lemmata are needed for Theorem 3.10 which shows how the unique stable expansion of a stratified set Σ can be computed more efficiently without explicitly constructing the Σ-full set Λ.

Lemma 3.8 *Let Σ be a set of formulae of the form $L\chi_1 \wedge \cdots \wedge L\chi_n \wedge \neg L\chi_{n+1} \wedge \cdots \wedge \neg L\chi_{n+m} \wedge \psi \rightarrow \psi'$ where $\psi, \psi' \in \mathcal{L}$, and let Λ be a set of formulae of the form $L\phi, \neg L\phi$ that contains exactly one of $L\chi$, $\neg L\chi$ for each $L\chi \in Sf^L(\Sigma)$. Define $Red(\Sigma, \Lambda) = \{\psi \rightarrow \psi' | (\phi \wedge \psi \rightarrow \psi') \in \Sigma,$ the conjuncts of ϕ are in $\Lambda\}$. Assume that for all $L\phi, \neg L\phi' \in \Lambda$, $\Sigma \cup \Lambda \models_L \phi$ and $\Sigma \cup \Lambda \not\models_L \phi'$. Then for all $\chi \in \mathcal{L}_{ae}$,*

$$Red(\Sigma, \Lambda) \models_L \chi \textit{ iff } \Sigma \cup \Lambda \models_L \chi.$$

Proof: The equivalence $\phi \wedge \psi \rightarrow \psi' \equiv \phi \rightarrow (\psi \rightarrow \psi')$ justifies proving the lemma using $Red(\Sigma, \Lambda) = \{\psi | (\phi \rightarrow \psi) \in \Sigma$, the conjuncts of ϕ are in $\Lambda\}$. The proof is by induction on the L-depth s of χ.

($s = 0$). For formulae with L-depth 0 there are no L subformulae and therefore $\models_L$ coincides with $\models$. ($\Rightarrow$). The implication $Red(\Sigma, \Lambda) \models_L \chi$ implies $\Sigma \cup \Lambda \models_L \chi$ is shown by contraposition, i.e $\Sigma \cup \Lambda \not\models_L \chi$ implies $Red(\Sigma, \Lambda) \not\models_L \chi$. Suppose $\Sigma \cup \Lambda \not\models \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \models \Sigma \cup \Lambda$ and $\mathcal{M} \not\models \chi$. We show that also $\mathcal{M} \models Red(\Sigma, \Lambda)$ and hence $Red(\Sigma, \Lambda) \not\models \chi$. Suppose $(\phi \rightarrow \psi) \in \Sigma$. If $\psi \in Red(\Sigma, \Lambda)$, then the conjuncts of ϕ are in Λ and consequently $\mathcal{M} \models \phi$. Because $\mathcal{M} \models (\phi \rightarrow \psi)$ also $\mathcal{M} \models \psi$. This shows that $\mathcal{M} \models Red(\Sigma, \Lambda)$, and $Red(\Sigma, \Lambda) \not\models \chi$.

($\Leftarrow$). Suppose $Red(\Sigma, \Lambda) \not\models \chi$, i.e. there is a model $\mathcal{M}$ such that $\mathcal{M} \models Red(\Sigma, \Lambda)$ and $\mathcal{M} \not\models \chi$. Let $\mathcal{M}'$ be $\mathcal{M}$ modified to satisfy Λ. Because there are no subformulae in $Red(\Sigma, \Lambda)$ or in χ that begin with L, the formulae in Λ can be made true without affecting truth-values of $Red(\Sigma, \Lambda)$ or χ. Hence $\mathcal{M}' \models \Lambda$ and $\mathcal{M}' \not\models \chi$. For showing that $\mathcal{M}' \models \Sigma$ the analysis for formulae $(\phi \rightarrow \psi) \in \Sigma$ is divided to two cases. First, if all conjuncts of ϕ are in Λ then ψ is in $Red(\Sigma, \Lambda)$ and therefore $\mathcal{M} \models \psi$, and further, $\mathcal{M}' \models \phi \rightarrow \psi$. Second, if at least one conjunct of ϕ is not in Λ then ϕ is false in $\mathcal{M}'$ (because then the complement of the conjunct is in Λ) and again $\mathcal{M}' \models \phi \rightarrow \psi$. Therefore $\Sigma \cup \Lambda \not\models_L \chi$.

($s \geq 1$). ($\Rightarrow$) Suppose $\Sigma \cup \Lambda \not\models_L \chi$, i.e., there is a model $\mathcal{M}$ such that $\mathcal{M} \models \Sigma \cup \Lambda \cup SB_{\Sigma \cup \Lambda}(\chi)$ and $\mathcal{M} \not\models \chi$. We show that also $\mathcal{M} \models Red(\Sigma, \Lambda) \cup SB_{Red(\Sigma,\Lambda)}(\chi)$

and hence $Red(\Sigma,\Lambda) \not\models_L \chi$. By the induction hypothesis $SB_{Red(\Sigma,\Lambda)}(\chi) = SB_{\Sigma\cup\Lambda}(\chi)$, and hence $\mathcal{M} \models SB_{Red(\Sigma,\Lambda)}(\chi)$. Suppose $(\phi \rightarrow \psi) \in \Sigma$. If $\psi \in Red(\Sigma,\Lambda)$, then the conjuncts of ϕ are in Λ and consequently $\mathcal{M} \models \phi$. Because $\mathcal{M} \models (\phi \rightarrow \psi)$ also $\mathcal{M} \models \psi$. This show that $\mathcal{M} \models Red(\Sigma,\Lambda)$, and finally $Red(\Sigma,\Lambda) \not\models \chi$.

($\Leftarrow$). Suppose $Red(\Sigma,\Lambda) \not\models_L \chi$, i.e. there is a model $\mathcal{M}$ such that $\mathcal{M} \models Red(\Sigma,\Lambda) \cup SB_{Red(\Sigma,\Lambda)}(\chi)$ and $\mathcal{M} \not\models \chi$. Let $\mathcal{M}'$ be $\mathcal{M}$ modified to satisfy Λ. This modification does not affect the truth-value of $Red(\Sigma,\Lambda)$ because there are no formulae beginning with L in $Red(\Sigma,\Lambda)$. That $SB_{Red(\Sigma,\Lambda)}(\chi)$ is true in $\mathcal{M}'$ is seen from the following chain of equivalences. For ϕ such that $L\phi \in Sf^{q^L}(\chi)$ and $L\phi$ or $\neg L\phi$ in Λ,

$$\begin{aligned} L\phi \in SB_{Red(\Sigma,\Lambda)}(\chi) \quad &\text{iff} \quad L\phi \in SB_{\Sigma\cup\Lambda}(\chi) && (28)\\ &\text{iff} \quad \Sigma \cup \Lambda \models_L \phi && (29)\\ &\text{iff} \quad L\phi \in \Lambda && (30) \end{aligned}$$

and for both $L\phi \in Sf^{q^L}(\chi)$ and Λ the formula $\neg L\phi$ is in the set exactly when $L\phi$ is not. Equivalence 28 is by the induction hypothesis. Equivalence 29 is by the definition of SB. Equivalence 30 is by the assumption of the lemma.

The truth value of χ is the same in $\mathcal{M}'$ and $\mathcal{M}$ because the truth-values of propositional variables are the same in both of these models, and the subformulae $Sf^{q^L}(\chi)$ of χ have the same truth values in both models as already shown above by the argument concerning $SB_{\Sigma\cup\Lambda}(\chi)$. Hence $\mathcal{M}' \models \Lambda \cup SB_{\Sigma\cup\Lambda}$ and $\mathcal{M}' \not\models \chi$.

The argument for $\mathcal{M}' \models \Sigma$ is the same as in the base case of the induction. □

Lemma 3.9 *Let ϕ be of the form $L\phi_1 \wedge \cdots \wedge L\phi_n \wedge \neg L\psi_1 \wedge \cdots \wedge \neg L\psi_m$ and $\Sigma \subseteq \mathcal{L}$. Then $\Sigma \models_L \phi$ iff $\Sigma \models_L \phi_i$ for all $i, 1 \leq i \leq n$ and $\Sigma \not\models_L \psi_j$ for all $j, 1 \leq j \leq m$.*

Proof: ($\Rightarrow$) Suppose that for some i, $\Sigma \not\models_L \phi_i$, or for some j, $\Sigma \models_L \psi_j$. Then one of $L\phi_i$ or $\neg L\psi_j$ is not in $SB_\Sigma(\phi)$ and $\Sigma \not\models_L \phi$. ($\Leftarrow$) Suppose that the antecedent is true. Then $SB_\Sigma(\phi) = \{L\phi_1, \ldots, L\phi_n, \neg L\psi_1, \ldots, \neg L\psi_m\}$, and therefore $\Sigma \cup SB_\Sigma(\phi) \models \phi$ and $\Sigma \models_L \phi$. □

Let Λ be the Σ-full set corresponding to the unique stable expansion $SE_\Sigma(\Lambda)$ of a stratified set Σ. The following theorem gives an algorithm for computing directly the set $Red(\Sigma,\Lambda) \subseteq \mathcal{L}$ which characterizes the stable expansion $SE_\Sigma(\Lambda)$.

Theorem 3.10 *Let $\Sigma = \Sigma_1^n$ be a stratified set, and define $Red_L(\Sigma, R) = \{o(\phi) \to c(\phi) | \phi \in \Sigma, R \models_L a(\phi)\}$. Define $R = R_n$ by*

$$
\begin{aligned}
R_0 &= \emptyset \\
R_{i+1} &= R_i \cup Red_L(\Sigma_{i+1}, R_i), 0 \le i < n.
\end{aligned}
$$

The set $\{\phi | R \models_L \phi\}$ is the unique stable expansion of Σ, and $R = Red(\Sigma, \Lambda)$.

Proof: Let Λ be the set computed by the algorithm of Theorem 3.6 for a set Σ. Because Λ is Σ-full it fulfills the conditions of Lemma 3.8, and by that lemma the stable expansion of Σ is $SE_\Sigma(\Lambda) = \{\phi | Red(\Sigma, \Lambda) \models_L \phi\}$. We shall show by induction that for all $i, 0 \le i \le n$, $R_i = Red(\Sigma_1^i, \Lambda_i)$, and this proves the theorem.

($i = 0$) Immediate, as $R_0 = \emptyset = Red(\Sigma_1^0, \Lambda_0)$.

($i \ge 1$).

$$
\begin{aligned}
R_i &= R_{i-1} \cup Red_L(\Sigma_i, R_{i-1}) \\
&= Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \cup Red_L(\Sigma_i, R_{i-1}) \\
&= Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \cup Red(\Sigma_i, \Lambda_i) \\
&= Red(\Sigma_1^i, \Lambda_i)
\end{aligned}
$$

The first equality is by the definition of R_i and the second by the induction hypothesis. The fourth is because $\Lambda_{i-1} \subseteq \Lambda_i$ and for each $L\phi \in \Sigma_1^{i-1}$ there is either $L\phi$ or $\neg L\phi$ in Λ_{i-1}. The third equality is shown as follows. Let $(\psi \to \phi) \in \Sigma_i$. Then $\phi \in Red_L(\Sigma_i, R_{i-1})$ iff for all conjuncts $L\chi$ of ψ,

$$
\begin{aligned}
R_{i-1} \models_L \chi \quad &\text{iff} \quad Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \models_L \chi \\
&\text{iff} \quad \Sigma_1^{i-1} \cup \Lambda_{i-1} \models_L \chi \\
&\text{iff} \quad \exists k \le i, L\chi \in (Sf^L(\Sigma_k) - Sf^L(\Sigma_1^{k-1})), \Sigma_1^{k-1} \cup \Lambda_{k-1} \models_L \chi \\
&\text{iff} \quad L\chi \in \Lambda_k \subseteq \Lambda_i.
\end{aligned}
$$

The first equivalence is by the induction hypothesis. The second is by Lemma 3.8. The third is because either $i = k$ and the equivalence is immediate, or $k < i$ and $L\chi \in Sf^L(\Sigma_k)$, and it follows by Lemma 3.7. The fourth equivalence is by the definition of Λ_j. Similarly with conjuncts $\neg L\chi$ of ψ. By the definition of Red and the equivalences above, equivalently $\phi \in Red(\Sigma_i, \Lambda_i)$. □

Example 3.4 The table below demonstrates the computation of Λ by the algorithm of Theorem 3.6, and the computation of $R = Red(\Sigma, \Lambda)$ by Theorem 3.10.

i	Σ_i	Λ	R
2	$L(p \leftrightarrow q) \rightarrow s$	$L(p \leftrightarrow q)$	s
1	$L\neg Lr \wedge p \rightarrow q$	$\neg Lr, L\neg Lr$	$p \rightarrow q$
	$q \rightarrow p$		$q \rightarrow p$

■

3.3 Tractable Classes of Stratified Theories

In this section we first analyze the complexity of computing the set R of Theorem 3.10 in the general case. In this computation all logical consequence tests are for objective formulae and can be implemented as calls to a theorem-prover for classical propositional logic. This is very convenient from the point of view of implementing a theorem-prover for stratified autoepistemic logic because existing theorem-provers for classical logic can be employed as subroutines without modification. The general complexity analysis and the implementation technique suggested by the analysis also hold for the quantificational version of autoepistemic logic in which quantification into the scope of the L-operator is not allowed; assuming that a closely related definition of stratification for the quantificational case is available.

The general complexity analysis gives a formula for the amount of resources (time or space) parameterized with the complexity $F(x)$ of the theorem-prover employed. The satisfiability of propositional Horn clauses can be tested in linear time, and by assigning $F(x) = ax + c$ for linear time a quadratic time upper bound for the membership testing in the stable expansions of a class of stratified propositional autoepistemic theories based on Horn clauses is obtained. For a more restricted class based on program clauses we observe a specific property that allows membership tests in linear time.

We write $|\phi|$ for the length of a formula ϕ, $|\Sigma|$ for the sum of lengths of the formulae in a set Σ, and $||\Sigma||$ for the cardinality of a set Σ.

As a basis of analyzing the complexity of computing the stable expansions of stratified sets we use the algorithm in Theorem 3.10. The algorithm computes a set $R \subseteq \mathcal{L}$ that characterizes the unique stable expansion of a stratified set. By the next lemma the explicit construction of the sets $SB_\Sigma(\chi)$ can be avoided in the $\models_L$ tests of the algorithm. As a result all consequence tests are for sets of formulae in which no L operators occur.

Lemma 3.11 *Let $\Sigma \subseteq \mathcal{L}$ and $\chi(L\phi)$ be a formula in which the formula $L\phi$ possibly occurs and $\chi(\top)$ the same formula where all occurrences of $L\phi$ have been replaced by the constant true $\top$. Now $\Sigma \cup \{L\phi\} \models \chi(L\phi)$ iff $\Sigma \models \chi(\top)$, and $\Sigma \cup \{\neg L\phi\} \models \chi(L\phi)$ iff $\Sigma \models \chi(\bot)$, where $\bot = \neg\top$.*

Proof: ($\Leftarrow$) Suppose $\Sigma \cup \{L\phi\} \not\models \chi(L\phi)$, i.e., there is a model M such that $\mathcal{M} \models \Sigma \cup \{L\phi\}$ and $\mathcal{M} \not\models \chi(L\phi)$. Clearly $\mathcal{M} \not\models \chi(\top)$. ($\Rightarrow$) Suppose $\Sigma \not\models \chi(\top)$, i.e., $\mathcal{M} \models \Sigma$ and $\mathcal{M} \not\models \chi(\top)$. Since the truth value of $L\phi$ is independent both of other formulae of the form $L\psi$ and of propositional variables, $\top$ can be replaced by $L\phi$ in χ and therefore $\Sigma \cup \{L\phi\} \not\models \chi(L\phi)$. Similarly for $\neg L\phi$ and $\bot$. □

Lemma 3.12 *$F(\chi, \Sigma)$ in Equation 31 gives the amount of resources needed for testing the $\models_L$-consequence of an arbitrary formula χ from a set of formulae $\Sigma \subseteq \mathcal{L}$. $R(x)$ is the amount of resources needed for performing a consequence test of size x.*

$$F(\chi, \Sigma) = R(|\chi| + |\Sigma| - \sum_{L\phi \in Sf^{qL}(\chi)} |\phi|) + \sum_{L\phi \in Sf^{qL}(\chi)} F(\phi, \Sigma) \tag{31}$$

and the equation with recursion removed is

$$\begin{aligned} F(\chi, \Sigma) \quad = \quad & R(|\chi| + |\Sigma| - \sum_{L\phi \in Sf^{qL}(\chi)} |\phi|) \\ & + \sum_{L\phi \in Sf^{L}(\chi)} R(|\phi| + |\Sigma| - \sum_{L\psi \in Sf^{qL}(\phi)} |\psi|). \end{aligned} \tag{32}$$

Proof: The second summand of the rhs of Equation 31 corresponds to the computation of the members of $SB_\Sigma(\chi)$ using $\models_L$. The first summand corresponds to the consequence test $\Sigma \models_L \chi$, i.e. the test of satisfiability of $\{\neg\chi\} \cup \Sigma \cup SB_\Sigma(\chi)$. By Lemma 3.11 the consequence test can be made by replacing $L\phi$ in χ by $\top$ for all $L\phi \in SB_\Sigma(\chi)$, and by $\bot$ for all $\neg L\phi \in SB_\Sigma(\chi)$, thus obtaining $\chi' \in \mathcal{L}$ and then testing the consequence $\Sigma \models \chi'$. This is why $\sum_{L\phi \in Sf^{qL}(\chi)} |\phi|$ is subtracted. □

Next we give an upper bound for the amount of resources needed for consequence tests in the computation of the algorithm in Theorem 3.10. Let $\Sigma = \Sigma_1^n$ be a stratified set and let there be an enumeration of the formulae $\phi_i \in \Sigma, 1 \leq i \leq r$ such that if the stratum of ϕ_i is lower than that of ϕ_j then $i < j$. Let $n_i = |a(\phi_i)|$ and $m_i = |o(\phi_i) \to c(\phi_i)|$. Ignoring the exact boundaries between the strata is an acceptable approximation since we are primarily interested in analyzing the upper bounds of complexity. The size of a stratified set is the sum of the sizes of its formulae

$$\sum_{i=1}^{r}(n_i + m_i + 1). \tag{33}$$

An upper bound for the resources needed for the consequence tests is

$$\sum_{i=1}^{r} F(a(\phi_i), \bigcup_{j=1}^{i-1}\{o(\phi_j) \rightarrow c(\phi_j)\}). \tag{34}$$

Tractable classes of stratified sets of autoepistemic formulae can be found by restricting the syntactic form of the formulae in such a way that the classical theorem proving task becomes tractable. As an example we present a tractable class SHC_{ae} based on Horn clauses, i.e., disjunctions of literals of which at most one is positive. For the members of this class the $\models$-consequence can be tested by using the linear time algorithm of Dowling and Gallier [1984] and for SHC_{ae} the algorithm of Theorem 3.10 runs in polynomial time.

Definition 3.13 *A formula χ is in the class HF_{ae} if it is a disjunction of conjunctions of formulae of the form p, $\neg p$, $L\phi$, and $\neg L\phi$ with at most one $\neg p$ in each disjunct, where each p is a propositional variable and each ϕ is in HF_{ae}.*

Definition 3.14 *SHC_{ae} is the class of finite stratified sets of formulae ϕ of the form $a(\phi) \wedge o(\phi) \rightarrow c(\phi)$ where $a(\phi)$ is a conjunction of zero or more formulae of the form $L\chi$ or $\neg L\chi$ where χ is in HF_{ae}, $o(\phi)$ is a conjunction of zero or more propositional variables, and $c(\phi)$ is a propositional variable or a negated propositional variable.*

The following are examples of the syntactic form of the formulae in sets in SHC_{ae}.

$$a \wedge b \rightarrow c$$
$$\neg L(LLp \vee q \vee \neg r \vee (t \wedge \neg u) \vee (x \wedge y)) \rightarrow \neg a$$

Theorem 3.15 *For a set Σ in SHC_{ae} the set $Red(\Sigma, \Lambda)$, where Λ is the unique Σ-full set, can be computed in $\mathcal{O}(n^2)$ time, where $n = |\Sigma|$.*

Proof: By Proposition 3.5 a stratification can be determined in $\mathcal{O}(|\Sigma|)$ time. The computation of $R = Red(\Sigma, \Lambda)$ by the algorithm in Theorem 3.10 is dominated by the consequence tests: in the computation of $Red_L(\Sigma_{i+1}, R_i)$ the removal of whole formulae in Σ_{i+1}, the removal of the autoepistemic parts

$a(\phi)$, $\phi \in \Sigma_{i+1}$, and the replacement of formulae $L\phi$ by $\top$ or $\perp$ during a $\models_L$-consequence test, are all constant time operations, and the number of these operations is less than the size of Σ_{i+1}.

All consequence tests reduce to satisfiability tests for formulae in conjunctive normal form with at most one positive literal in each conjunct, so that the linear time algorithm of [Dowling and Gallier, 1984] can be used. For logical consequence tests in the computation of the algorithm in Theorem 3.10, instantiate $R(x) = ax + c$ to the second equation of Lemma 3.12:

$$\begin{aligned} F(\chi, \Sigma) &= a(|\chi| + |\Sigma| - \sum_{L\phi \in Sf^{qL}(\chi)} |\phi|) + c \\ &\quad + \sum_{L\phi \in Sf^{L}(\chi)} (a(|\phi| + |\Sigma| - \sum_{L\psi \in Sf^{qL}(\phi)} |\psi|) + c). \end{aligned}$$

By reordering the terms we obtain

$$\begin{aligned} F(\chi, \Sigma) &= a(|\chi| + |\Sigma| + \sum_{L\phi \in Sf^{L}(\chi)} |\Sigma|) + (c + \sum_{L\phi \in Sf^{L}(\chi)} c) \\ &\quad + a(- \sum_{L\phi \in Sf^{qL}(\chi)} |\phi| + \sum_{L\phi \in Sf^{L}(\chi)} |\phi| - \sum_{L\phi \in Sf^{L}(\chi)} \sum_{L\psi \in Sf^{qL}(\phi)} |\psi|). \end{aligned}$$

The value of the third summand is zero since the length of each $L\phi \in Sf^L(\chi)$ is added and subtracted exactly once. From this we get the upper bound

$$\begin{aligned} & a(|\chi| + |\Sigma| + \|Sf^L(\chi)\| \cdot |\Sigma|) + c(1 + \|Sf^L(\chi)\|) \\ & \leq a(|\chi| + |\Sigma| \cdot (1 + \|Sf^L(\chi)\|)) + c \cdot |\chi|) \\ & \leq a(|\chi| + |\Sigma| \cdot |\chi|) + c \cdot |\chi| \\ & < (a + c)(|\chi| + |\Sigma| \cdot |\chi|) \end{aligned}$$

for $F(\chi, \Sigma)$. Using this and Equation 34 (by the definition of $\mathcal{O}$ the constant factor $a + c$ appearing in each term can be left out) we get

$$\begin{aligned} \sum_{i=1}^{r} (n_i + n_i \sum_{j=1}^{i-1} m_i) &\leq (\sum_{i=1}^{r} n_i) + (\sum_{i=1}^{r} n_i)(\sum_{i=1}^{r} m_i) \\ &\leq (\sum_{i=1}^{r} n_i)^2 + 2(\sum_{i=1}^{r} n_i)(\sum_{i=1}^{r} m_i) + (\sum_{i=1}^{r} m_i)^2 \\ &< (\sum_{i=1}^{r} n_i + m_i + 1)^2. \end{aligned}$$

This establishes the $\mathcal{O}(n^2)$ upper bound for the logical consequence tests. $\square$

Theorem 3.16 *Given $Red(\Sigma, \Lambda)$ where $\Sigma \in SHC_{ae}$ and Λ is a Σ-full set, the membership in the unique stable expansion $\{\phi | Red(\Sigma, \Lambda) \models_L \phi\}$ of Σ for formulae ϕ in HF_{ae} can be decided in $\mathcal{O}(n^2)$ time, where $n = |\Sigma| + |\neg\phi|$.*

Proof: By the second equation of Lemma 3.12 the amount of resources needed is bounded by $|\Sigma| \cdot |\neg\phi| + |\neg\phi|$ as shown in the proof of Theorem 3.15, and this is $\mathcal{O}(n^2)$. □

Theorem 3.17 *Let Σ be in SHC_{ae}. The membership problem of the unique stable expansion of Σ for formulae ϕ in HF_{ae} is solvable in $\mathcal{O}(n^2)$ time, where $n = |\Sigma| + |\neg\phi|$.*

The line taken in establishing the above result suggests further tractable classes based on other subsets of propositional logic for which polynomial time satisfiability tests are available, like those presented in [Schaefer, 1978; Gallo and Scutellà, 1988]. Next we investigate an even more restricted class for which the logical consequence testing does not have to be done separately for each formula inside L.

Definition 3.18 *A formula χ is in CF_{ae} if it is a conjunction of one or more formulae of the form p, $L\phi$, and $\neg L\phi$ where each p is a propositional variable and each ϕ is in CF_{ae}.*

Definition 3.19 *SPC_{ae} is the class of finite stratified sets of formulae ϕ of the form $a(\phi) \wedge o(\phi) \rightarrow c(\phi)$ where $a(\phi)$ is a conjunction of zero or more formulae of the form $L\chi$ or $\neg L\chi$ and each χ is in CF_{ae}, $o(\phi)$ is a conjunction of zero or more propositional variables, and $c(\phi)$ is a propositional variable.*

The following formulae illustrate the form of formulae in sets in SPC_{ae}:

$$\neg L(a \wedge b \wedge c \wedge \neg Ld) \rightarrow e$$
$$L(p \wedge Lq \wedge \neg Lr) \wedge \neg L(\neg Ls) \wedge t \rightarrow u$$

In [Dowling and Gallier, 1984] two linear time algorithms are given for testing the satisfiability of a set of Horn clauses. We modify the first one of these to be used in linear time tests for the membership in the unique stable expansions of sets in SPC_{ae}. By Theorem 3.10 explicit construction of the sets Λ_i can be avoided and instead of separately testing the logical consequence of each formula inside L in $a(\phi)$ the whole set of consequences for one stratum of a stratification can be computed by one run of the algorithm.

The function DG is the basis of the efficient algorithm for SPC_{ae} and can be implemented as a variant of Algorithm 2 of [Dowling and Gallier, 1984]. Dowling and Gallier's algorithm works with arbitrary Horn clauses, but ours is restricted to *program clauses*, i.e., disjunctions of literals exactly one of which is positive. Sometimes program clauses are written as $a_1 \wedge \cdots \wedge a_n \rightarrow b$ instead of $\neg a_1 \vee \cdots \vee \neg a_n \vee b$.

In our algorithm sets of propositional variables are represented by their characteristic functions or equivalently arrays. For array indexing Assumption 3.4 is essential.

Proposition 3.20 *Let Σ be a set of program clauses and v a set of propositional variables. Then*

$$DG(\Sigma, v) = \{p | p \text{ is a propositional variable}, \Sigma \cup v \models p\}$$

can be computed in $\mathcal{O}(|\Sigma|)$ time.

Proof: The computation of DG for a set of program clauses Σ and a set of propositional variables v uses the following arrays:

poslitlist Each element n is initialized to the propositional variable appearing in the positive literal of the clause $\phi_n \in \Sigma$. The initialization can be done by one traversal of the set of formulae, which is linear time.

clauselist The element n is the list of clauses of Σ in which the variable in the positive literal of ϕ_n appears in a negative literal. Initialization can be done in linear time.

numargs The element n is initialized to the number of variables p in the negative literals of ϕ_n for which $v(p) = 0$.

The function is computed by the procedure in Figure 4. Under Assumption 3.4 the computation is linear time on the size of Σ. The argument is the same as for the linearity of Algorithm 2 of [Dowling and Gallier, 1984]. The initializations of the procedure for a set of clauses Σ is $\mathcal{O}(|\Sigma|)$. Each clause n of Σ is pushed to the queue at most once, i.e. in the initialization part of the procedure when it is found that $v(p) = 1$ for all body variables p that appear negatively in n, or inside the *while* loop when the value of $v(poslitlist[n])$ is changed from 0 to 1. Each round of the *while* loop corresponds the deletion of negative occurrences of some propositional variable $poslitlist[c]$ for and for each propositional variable there is at most one round of the loop. Hence the amount of computation inside the while loop is proportional to the number of negative literals in Σ. □

Computing $Red_L(\Sigma, v) = \{\phi \rightarrow \phi' | (L\chi_i \wedge \cdots \wedge L\chi_n \wedge \neg L\chi_{n+1} \wedge \cdots \wedge \neg L\chi_{n+m} \wedge \phi \rightarrow \phi') \in \Sigma, v \models_L \chi_1, \ldots, v \models_L \chi_n, v \not\models_L \chi_{n+1}, \ldots, v \not\models_L \chi_{n+m}\}$ can be done in linear time on $|\Sigma|$ for members of SPC$_{ae}$. The consequence tests $v \models_L \chi$, χ in CF$_{ae}$ can be done in linear time on $|\chi|$ by using the following algorithm.

```
function DG(Σ, v : array of {0,1}) : array of {0,1};
begin
  initialize poslitlist, clauselist, numargs;
  initialize queue to the list of clauses n for which numargs[n] = 0;
  for each c in queue do v(poslitlist[c]) := 1;
  while queue ≠ empty do
    clause1 := pop(queue);
    for each clause2 in clauselist[clause1] do
      numargs[clause2] := numargs[clause2] - 1;
      if numargs[clause2] = 0 then
        n := poslitlist[clause2];
        if v(n) = 0 then
          v(n) := 1;
          queue := push(clause2,queue);
        end if
      end if
    end for
  end while
  return v;
end
```

Figure 4: A Version of Algorithm 2 of Dowling and Gallier

$$v \models_L \chi \text{ iff for each conjunct } \phi \text{ of } \chi \begin{cases} \text{if } \phi \text{ is atomic then } \phi \in v \\ \text{if } \phi = L\psi \text{ then } v \models_L \psi \\ \text{if } \phi = \neg L\psi \text{ then } v \not\models_L \psi \end{cases}$$

The following two lemmata are needed for establishing Lemma 3.23 which describes how to compute in $\mathcal{O}(|\Sigma|)$ time the set of propositional variables in the unique stable expansion of a set Σ in SPC$_{ae}$.

Lemma 3.21 *Let $\Sigma \subseteq \mathcal{L}$, $v = \{p | p \text{ is a propositional variable}, \Sigma \models p\}$, and χ be in CF_{ae}. Then $v \models_L \chi$ iff $\Sigma \models_L \chi$.*

Proof: By induction on the L-depth s of χ. ($s = 0$). Suppose $\Sigma \not\models_L \chi$, i.e., there is a model $\mathcal{M}$ for which $\mathcal{M} \models \Sigma$ and $\mathcal{M} \not\models \chi$. Because v is the set of propositional variables true in every model of Σ, also $\mathcal{M} \models v$, and $v \not\models_L \chi$. Suppose $\Sigma \models_L \chi$. Because χ is a conjunction of propositional variables each of which is a logical consequence of Σ, obviously $v \models_L \chi$. ($s \geq 1$). By the induction hypothesis $SB_v(\chi) = SB_\Sigma(\chi)$, and replacing members of $SB_v(\chi)$ (and $SB_\Sigma(\chi)$) according to Lemma 3.11 by $\top$ or $\bot$, χ can be reduced to χ' of L-depth 0 for which $v \models \chi'$ iff $\Sigma \models \chi'$, which is shown as in the case $s = 0$. $\Box$

Lemma 3.22 *Let Σ be a set of program clauses and Γ is the set propositional variables p such that $\Sigma \models p$. Let Δ be a set of program clauses such that the propositional variables in the positive literals of Δ do not occur in the negative literals of Σ. Then for all propositional variables p, $\Sigma \cup \Delta \models p$ iff $\Gamma \cup \Delta \models p$.*

Proof: Suppose $\Sigma \cup \Delta \not\models p$, i.e., for some model $\mathcal{M}$, $\mathcal{M} \models \Sigma \cup \Delta$ and $\mathcal{M} \not\models p$. Clearly $\mathcal{M} \models \Gamma \cup \Delta$ and therefore $\Gamma \cup \Delta \not\models p$. Suppose $\Gamma \cup \Delta \not\models p$, i.e., there is a model $\mathcal{M}$, $\mathcal{M} \models \Gamma \cup \Delta$, $\mathcal{M} \not\models p$. $\mathcal{M}'$ is $\mathcal{M}$ modified to satisfy $\Sigma \cup \Delta$ in the following way. Formulae $\phi = \neg a_1 \vee \cdots \vee \neg a_n \vee b, \phi \in \Sigma$ can be false if $\mathcal{M} \models a_1 \wedge \cdots \wedge a_n$ and $\mathcal{M} \not\models b$. Now $\mathcal{M}'$ can be modified to make ϕ true by making one of $a_i, 1 \leq i \leq n$ false. This can be done without falsifying formulae in $\Gamma \cup \Delta$ because **i)** not all $a_i, 1 \leq i \leq n$ can be logical consequences of Σ. If they were, then $b \in \Gamma$. And **ii)** as propositional variables in the positive literals of Δ do not occur in the negative literals of Σ, this modification falsifies no formula in Δ. Therefore $\mathcal{M}' \models \Sigma \cup \Delta$ and $\Sigma \cup \Delta \not\models p$. □

Lemma 3.23 *For $\Sigma = \Sigma_1^n$ in SPC_{ae} and formulae χ in CF_{ae}, $v_n \models_L \chi$ iff χ belongs to the unique stable expansion of Σ, where v_n is defined by*

$$\begin{aligned} v_0 &= \emptyset \\ v_{i+1} &= DG(Red_L(\Sigma_{i+1}, v_i), v_i), 0 \leq i < n. \end{aligned}$$

Furthermore, v_n, which is the set of propositional variables in the unique stable expansion of Σ, can be computed in $\mathcal{O}(|\Sigma|)$ time.

Proof: We prove $v_i \models_L \chi$ iff $Red(\Sigma_1^i, \Lambda_i) \models_L \chi$, where Λ_i as in Theorem 3.6, by induction on i. $(i = 0)$. Immediate as $v_0 = Red(\Sigma_1^0, \Lambda_0) = \emptyset$. $(i > 0)$. For all propositional variables p,

$$\begin{aligned} v_i \models p \quad &\text{iff} \quad v_{i-1} \cup Red_L(\Sigma_i, v_{i-1}) \models p \\ &\text{iff} \quad Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \cup Red_L(\Sigma_i, v_{i-1}) \models p \\ &\text{iff} \quad Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \cup Red(\Sigma_i, \Lambda_i) \models p \\ &\text{iff} \quad Red(\Sigma_1^i, \Lambda_i) \models p. \end{aligned}$$

The first equivalence is by the definition of v_i, the second and the third respectively by **A** and **B** below. The fourth equivalence is because $\Lambda_{i-1} \subseteq \Lambda_i$ and for each $L\phi \in \Sigma_1^{i-1}$ there is either $L\phi$ or $\neg L\phi$ in Λ_{i-1}. **A.** By the induction hypothesis $v_{i-1} \models p$ iff $Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \models p$. We get the equivalence by Lemma 3.22 taking $\Delta = Red_L(\Sigma_i, v_{i-1}), \Gamma = v_{i-1}, \Sigma = Red(\Sigma_1^{i-1}, \Lambda_i)$. **B.** For $\phi, (\psi \rightarrow \phi) \in \Sigma_i$, $\phi \in Red_L(\Sigma_i, v_{i-1})$ iff for conjuncts $L\chi$ of ψ

$$v_{i-1} \models_L \chi \quad \text{iff} \quad Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \models_L \chi \tag{35}$$

$$\text{iff} \quad \Sigma_1^{i-1} \cup \Lambda_{i-1} \models_L \chi \tag{36}$$

$$\text{iff} \quad L\chi \in \Lambda_i \tag{37}$$

and similarly for conjuncts $\neg L\chi'$ of ψ, which by the definition of Red is equivalent to $\phi \in Red(\Sigma_i, \Lambda_i)$. Therefore $Red_L(\Sigma_i, v_{i-1}) = Red(\Sigma_i, \Lambda_i)$. Equivalence 35 above is by the induction hypothesis. Equivalence 36 is by Lemma 3.8. Equivalence 37 is by Lemma 3.7.

By Lemma 3.21 we get the induction step, i.e. for all χ in CF_{ae}, $v_i \models \chi$ iff $Red(\Sigma_1^i, \Lambda_i) \models \chi$. Thus $v_n \models_L \chi$ iff $Red(\Sigma_1^n, \Lambda_n) \models_L \chi$ iff (by Lemma 3.8) $\Sigma_1^n \cup \Lambda_n \models_L \chi$ iff (by Theorem 2.9) χ is in the unique stable expansion of Σ.

By Proposition 3.5 stratification can be computed in linear time, and using Assumption 3.4 the computation is $\mathcal{O}(|\Sigma|)$ time because by Proposition 3.20 computing $Red_L(\Sigma_i, v_{i-1})$ is linear in $|\Sigma_i|$ and this is done exactly once for each $\Sigma_i \subseteq \Sigma, 1 \leq i \leq n$. The size of $Red_L(\Sigma_i, v_{i-1})$ is smaller than or equal to that of Σ_i and therefore the respective computation with DG is linear in $|\Sigma_i|$. $\square$

Theorem 3.24 *Let Σ be in SPC_{ae} and χ in CF_{ae}. The membership of χ in the unique stable expansion of Σ can be decided in $\mathcal{O}(n)$ time, where $n = |\Sigma| + |\chi|$.*

Proof: By Lemma 3.23 the computation of the set v_n for $\Sigma = \Sigma_1^n$ is $\mathcal{O}(|\Sigma|)$ time. By Lemma 3.23 the membership in the unique stable expansion can be tested by $v_n \models_L \chi$, and this is $\mathcal{O}(|\chi|)$ time. $\square$

4 Implications of the Tractability Results

Here we show that the complexity results for restricted classes of autoepistemic logic can be applied in other areas of nonmonotonic reasoning, e.g. in propositional default logic, propositional logic programs and in nonmonotonic truth-maintenance systems. The applicability in all these cases is based on the existence of translations from these formalisms to autoepistemic logic. For default logic we give a proof of the correctness of the translation in the stratified case and in other cases the translations are immediate consequences of results already presented elsewhere. In the end of this section we make a comparison between the tractable stratified classes of default theories and several unstratified classes that have been shown intractable by [Kautz and Selman, 1991; Stillman, 1990].

4.1 Default Logic

Default logic is a nonmonotonic logic closely related to autoepistemic logic. Konolige [1988] proves that each extension of a default theory is the objective part of a stable expansions of a translation of the default theory to autoepistemic logic. However, there are stable expansions the objective parts of which are not extensions of the corresponding default theory and hence the correspondence is not exact. A notion of stratification can be introduced to default logic in the same way as in autoepistemic logic. For stratified theories the correspondence of extensions and stable expansions under Konolige's translation is exact, as will be shown below. The definition of extensions of default theories is based on a complex fixpoint equation [Reiter, 1980] and for proofs it is more convenient to use a definition of extensions that has a more constructive flavour.

Theorem 4.1 ([Reiter, 1980]) *Let $E \subseteq \mathcal{L}$ be a set of well-formed formulae, and let $T = \langle D, W\rangle$ be a default theory. Define*

$$\begin{aligned} E_0 &= W \\ E_{i+1} &= \{\phi | E_i \models \phi\} \cup appl_{D,E}(E_i), i \geq 0 \end{aligned}$$

where $appl_{D,E}(E_i) = \{w | \dfrac{\alpha : \beta_1, \ldots, \beta_n}{w} \in D, \alpha \in E_i, \neg\beta_1, \ldots, \neg\beta_n \notin E\}$, and $\models$ is the classical logical consequence relation. Then E is an extension for T iff

$$E = \bigcup_{i=0}^{\infty} E_i.$$

The definition of stratification for autoepistemic logic presented [Marek and Truszczyński, 1991] with the translation of a class of default theories to autoepistemic logic of Theorem 4.3 below, suggests a similar definition of stratification for default theories.

Definition 4.2 *A default theory $\langle D, W\rangle$ is stratified if*

1. *Formulae $\phi \in W$ are implications of the form $o(\phi) \rightarrow c(\phi)$ and the rules $d \in D$ are of the form $\alpha_d : \beta_{d1}, \ldots, \beta_{dm}/\gamma_d$.*
2. *The set $\{\gamma_d | d \in D\} \cup \{c(\phi)|\phi \in W\}$ is satisfiable.*
3. *There exists a set of indices $I = \{1, \ldots, n\}$ or $I = \{1, \ldots\}$ and a partition $\Sigma_i, i \in I$ of $\Sigma = D \cup W$ such that for all $j \in I$ the propositional variables occurring in $\{c(\phi)|\phi \in \Sigma_j\} \cup \{\gamma_d | d \in \Sigma_j\}$ do not occur in $\bigcup_{i=1}^{j-1} \Sigma_i$, or in a prerequisite or justification of any default rule in Σ_j.*

Theorem 4.3 *Stratified default theories $\langle D, W\rangle$ can be translated into stratified autoepistemic theories by the following translation.*

$$tr_{dl}(D, W) = W \cup \{L\alpha \wedge \neg L\neg\beta_1 \wedge \cdots \wedge \neg L\neg\beta_n \rightarrow \gamma | \ \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \ \in D\}$$

The set of extensions of $\langle D, W\rangle$ is exactly the set of objective parts of the stable expansions of $tr_{dl}(D, W)$.

Proof: By Proposition 5.2 of [Konolige, 1988] every extension of $\langle D, W\rangle$ is the objective part $E = \Delta \cap \mathcal{L}$ of some stable expansion Δ of $tr_{dl}(D, W)$. The opposite direction, i.e., objective parts $E = \Delta \cap \mathcal{L}$ of stable expansions of $tr_{dl}(D, W)$ are extensions of $\langle D, W\rangle$, is shown by $E = \bigcup_{i=0}^{\infty} E_i$ where the sets E_i are as in Theorem 4.1. First the inclusion $\bigcup_{i=0}^{\infty} E_i \subseteq E$ is proved. We show by induction on i that for all $i \geq 0, E_i \subseteq E$.

$(i = 0)$. $E_0 = W \subseteq E$ because $W \subseteq \Delta$. $(i \geq 1)$. Suppose $\gamma \in \mathrm{appl}_{D,E}(E_i)$. Then $\alpha \in E_{i-1}$ and by the induction hypothesis $\alpha \in E \subseteq \Delta$ and by the definition of stable expansions $L\alpha \in \Delta$, and $\neg\beta_j \notin E$ and as Δ is a stable expansion $\neg L \neg\beta_j \in \Delta$. Because stable expansions are closed by logical consequence also $\gamma \in E$. Thus $\bigcup_{i\geq 0}\{\gamma|\gamma \in \mathrm{appl}_{D,E}(E_i)\} \subseteq E$ and $W \subseteq E$. Like $\bigcup_{i=0}^{\infty} E_1$ also E is closed under logical consequence and therefore $\bigcup_{i=0}^{\infty} E_i \subseteq E$.

Then we show the inclusion $E \subseteq \bigcup_{i=0}^{\infty} E_i$. This part of the proof rests on the stratifiedness of $\Sigma = tr_{dl}(D, W)$. Let $\Sigma_i, 1 \leq i \leq n$ be the strata of $tr_{dl}(D, W)$, and $\Lambda_i, 0 \leq i \leq n$ as in Theorem 3.6. By Theorem 2.9 and Lemma 3.8 the stable expansion of a stratified set Σ is $SE_{\Sigma}(\Lambda_n) = \{\phi | Red(\Sigma, \Lambda_n) \models_L \chi\}$. By the definition of $\models_L$ the objective part E of the stable expansion of Σ

is $\{\phi | Red(\Sigma, \Lambda) \models \phi\}$. Because $\bigcup_{i=0}^{\infty} E_i$ is logically closed it suffices to show that $Red(\Sigma, \Lambda) \subseteq \bigcup_{i=0}^{\infty} E_i$. This will be proved by induction on the strata $i, 0 \leq i \leq n$.

$(i = 0)$. $Red(\Sigma_1^0, \Lambda_0) = \emptyset \subseteq \bigcup_{i=0}^{\infty} E_i$. $(i \geq 1)$. By definition $W \subseteq E$ is also contained in $\bigcup_{i=0}^{\infty} E_i$. Suppose that for $\gamma \in Red(\Sigma_1^i, \Lambda_i)$ there is a formula $(L\alpha \wedge \neg L\neg\beta_1 \wedge \cdots \wedge \neg L\neg\beta_s \rightarrow \gamma) \in \Sigma_i$. By Lemma 3.7 and the definitions of Red and Λ_i this means that for $L\alpha$, $Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \models \alpha$ and for $\neg L\neg\beta_j, 1 \leq j \leq s$, $Red(\Sigma_1^{i-1}, \Lambda_{i-1}) \not\models \neg\beta_j$. By Lemma 3.7 $Red(\Sigma_1^n, \Lambda_n) \not\models \neg\beta_j$ and hence $\neg\beta_j \notin E$, and because of the inclusion $\bigcup_{i=0}^{\infty} E_i \subseteq E$ shown above finally $\neg\beta_j \notin \bigcup_{i=0}^{\infty} E_i$. Because α is in $\bigcup_{i=0}^{\infty} E_i$ by the induction hypothesis and $\neg\beta_j$ are not, $\gamma \in \bigcup_{i=0}^{\infty} E_i$. □

Under the above translation stratified default theories map to stratified autoepistemic theories, and the counterparts of the quadratic and linear time classes of propositional autoepistemic logic in default logic are obtained. Define SHC_{dl} to be the class of default theories $\langle D, W\rangle$ where W is a set of Horn clauses, i.e., disjunctions of literals at most one of which is positive, and D a set of default rules of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$$

where α is a disjunction of conjunctions of literals with at most one negative literal in each conjunct, each β_i is a conjunction of disjunctions of literals with at most one positive literal in each disjunct, and γ is a literal. By noticing that the translation of Theorem 4.3 is linear time we get by Theorems 3.17 and 4.3 the following result ($|\Sigma|$ and $|\phi|$ denote of the sum of the lengths of the formulae involved).

Theorem 4.4 *For disjunctions of conjunctions of literals ϕ with at most one negative literal in each conjunct, the membership in the unique extensions of default theories T in SHC_{dl} can be decided in $\mathcal{O}(n^2)$ time, where $n = |T| + |\neg\phi|$.*

The class SPC_{dl} is like SHC_{dl} except that α are conjunctions of propositional variables, and β_j are disjunctions of negated propositional variables, and W is a set of program clauses, i.e. set of disjunctions of literals, with exactly one positive literal in each disjunction. By Theorem 3.24 the following is immediate.

Theorem 4.5 *For conjunctions of propositional variables ϕ the membership in the unique extensions of default theories T in SPC_{dl} can be decided in $\mathcal{O}(n)$ time, where $n = |T| + |\phi|$.*

4.2 Propositional Logic Programs

Gelfond and Lifschitz [1988] present a semantics for general logic programs called *the stable model semantics*, and show that it is essentially an autoepistemic semantics in the sense that there is a bijection between the stable models of a logic program and the stable expansions of its translation into autoepistemic logic. The translation is as follows.

$$\begin{aligned} tr_{lp}(P) \quad = \quad & \{p_1 \wedge \cdots \wedge p_n \wedge \neg Lq_1 \wedge \cdots \wedge \neg Lq_m \rightarrow r | \\ & (r \leftarrow p_1, \ldots, p_n, \text{not } q_1, \ldots, \text{not } q_m) \in P\} \end{aligned}$$

By noticing that the clauses of the program are a special case of the formulae in theories in SPC_{ae} we immediately get the following result.

Theorem 4.6 *The unique stable model of stratified propositional logic programs P can be computed in time $\mathcal{O}(n)$, where $n = |P|$.*

Proof: By Theorem 3 of [Gelfond and Lifschitz, 1988] the unique stable model of P is the set of propositional variables in the stable expansion of $tr_{lp}(P)$. By our Theorem 3.23 this set can be computed in $\mathcal{O}(|P|)$ time. □

This results improves Kautz and Selman's [1991] $\mathcal{O}(n^3)$ result which is based on an algorithm that computes an extension for a restricted class of ordered default theories and a translation of stratified logic program into this class of theories. The algorithm is based on an ordering of literals given by the orderedness condition.
Our linear-time result makes a striking contrast with the complexity of propositional general logic programs in the general case.

Theorem 4.7 ([Marek and Truszczyński, 1991]) *The problem of determining whether a propositional general logic program has a stable model is NP-complete.*

4.3 Truth-Maintenance Systems

Elkan [1990] shows how sets of justifications of truth-maintenance systems of Doyle [1979] can be seen as general propositional logic programs and also as a subclass of propositional autoepistemic theories. The result that if a set of justifications is stratified then the computation of the unique grounded model is linear time, is immediate.

Theorem 4.8 *The unique grounded model of a stratified set of justifications Π can be computed in $\mathcal{O}(n)$ time, where $n = |\Pi|$.*

Proof: By Theorem 4.9 of [Elkan, 1990] M is a grounded model of Π iff M is the set of propositional variables in some stable expansion of the autoepistemic theory corresponding to Π. In the stratified case there is exactly one stable expansion, and by Lemma 3.23 the set can be computed in linear time on the size of the theory, i.e. the set of justifications. □

As for logic programs a similar NP-completeness result for the general case exists as expected.

Theorem 4.9 ([Urbanski, 1988]) *The problem of whether a set of justifications has a grounded model is NP-complete.*

4.4 Complexity of Restricted Default Theories

The complexity of nonmonotonic reasoning in syntactically restricted cases was first investigated by Kautz and Selman [1991] in the framework of default logic and later their results were refined by Stillman [1990]. By an analogy from classical logics it was assumed that by a straightforward syntactic restriction tractable classes of default logic could be easily obtained. However, surprisingly it turned out that this analogy does not hold and that only extreme restriction brings the reasoning to tractable level.

The forms of rules analyzed by Kautz and Selman are as follows (Ordered theories below fulfill the ordering condition presented in [Etherington, 1987] which guarantees the existence of at least one extension. The letters a, b, c denote literals and the letters p, q, r denote propositional variables).

Disjunction free	$\dfrac{a_1 \wedge \cdots \wedge a_l : b_1 \wedge \cdots \wedge b_n \wedge c_1 \wedge \cdots \wedge c_m}{b_1 \wedge \cdots \wedge b_n}$
Unary	$\dfrac{p : q}{q} \quad \dfrac{p : q \wedge \neg r}{q} \quad \dfrac{p : \neg q}{\neg q}$
Disjunction free ordered	$\dfrac{a_1 \wedge \cdots \wedge a_l : b_1 \wedge \cdots \wedge b_n \wedge c_1 \wedge \cdots \wedge c_m}{b_1 \wedge \cdots \wedge b_n}$
Ordered unary	$\dfrac{p : q}{q} \quad \dfrac{p : q \wedge \neg r}{q} \quad \dfrac{p : \neg q}{\neg q}$
Disjunction free normal	$\dfrac{a_1 \wedge \cdots \wedge a_l : b_1 \wedge \cdots \wedge b_n}{b_1 \wedge \cdots \wedge b_n}$
Horn	$\dfrac{p_1 \wedge \cdots \wedge p_n : q}{q} \quad \dfrac{p_1 \wedge \cdots \wedge p_n : \neg q}{\neg q}$
Normal unary	$\dfrac{p : q}{q} \quad \dfrac{p : \neg q}{\neg q}$

Stillman [1990] extends the analysis to rules without prerequisites.

Prerequisite free	$\frac{: b_1 \wedge \cdots \wedge b_n \wedge c_1 \wedge \cdots \wedge c_m}{b_1 \wedge \cdots \wedge b_n}$
Prerequisite free ordered	$\frac{: b_1 \wedge \cdots \wedge b_n \wedge c_1 \wedge \cdots \wedge c_m}{b_1 \wedge \cdots \wedge b_n}$
Prerequisite free unary	$\frac{: q \wedge \neg r}{q} \quad \frac{: \neg q}{\neg q}$
Prerequisite free ordered unary	$\frac{: q \wedge \neg r}{q} \quad \frac{: \neg q}{\neg q}$
Prerequisite free normal	$\frac{: b_1 \wedge \cdots \wedge b_n}{b_1 \wedge \cdots \wedge b_n}$
Prerequisite free normal unary	$\frac{: q}{q} \quad \frac{: \neg q}{\neg q}$
Prerequisite free positive normal unary	$\frac{: q}{q}$

The hierarchy of these classes of default theories ordered by inclusion is depicted in Figure 5. The upper dashed line marks the boundary between NP-hardness or co-NP-hardness and polynomiality for cautious reasoning of default theories $\langle D, W \rangle$ where W is a set of literals. The lower dashed line is for cautious reasoning of default theories for which W is a set of disjunctions of two literals. When W is a set of Horn clauses cautious reasoning is NP-hard for all these classes. This means that in practice there are no tractable classes of default theories (wrt. cautious reasoning) for which W is a set of Horn clauses.

For comparing the effect of stratification on the complexity of default reasoning we would like to investigate the stratified counterparts of these classes. One would assume that because the classical reasoning required in the decision procedures of these classes is very easy, in the stratified cases these classes would be tractable. And this is in fact the case. The stratification condition of Section 4.1 cannot immediately be applied because all rules in these classes are either normal or seminormal, i.e., the same propositional variable occurs both in the justification and the consequence, and theories containing these kinds of rules clearly are not stratified. However, the reason why semi-normal rules together with the orderedness condition and normal rules are used is that they guarantee the existence of extensions. The definition of stratification is also a sufficient condition – although more restrictive – for the existence of extensions[1]. Thus the duplication of the consequent in the justification is unnecessary and for comparison we may just omit those parts of the justifications,

[1] As we can see: for a stratified default theory $\langle D, W \rangle$ the set of heads of the implications in W and the conclusions of the default rules are required to be simultaneously satisfiable, and consequently no rule application can produce an inconsistency

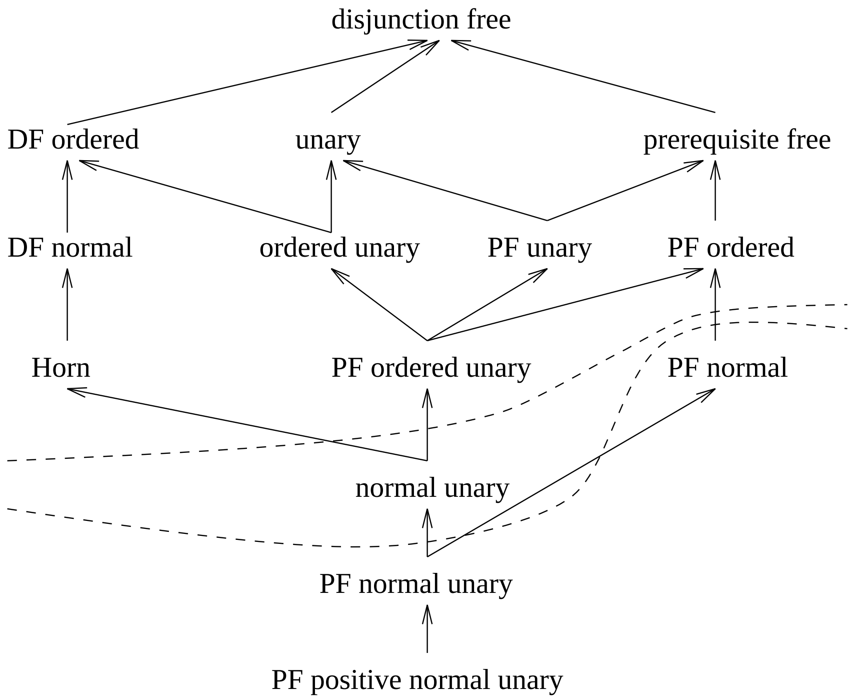

Figure 5: A Hierarchy of Default Theories

To show that in the stratified case all classes analyzed by [Kautz and Selman, 1991; Stillman, 1990] are tractable, we show that the most general classes, i.e., the ones based on disjunction-free default rules and Horn clauses and the ones based on disjunction-free default rules and 2-literal clauses, are tractable. The notion of the size of a default theory $\langle D, W\rangle$ can be defined for example as the sum of the sizes of the formulae in W and in the prerequisites, justifications, and conclusions of the default rules in D.

Theorem 4.10 *Let ϕ be a disjunction of conjunctions of literals with at most one negative literal in each conjunct (resp. a disjunction of conjunctions of 2 literals). In the stratified case deciding whether a formula ϕ belongs to the unique extension of disjunction-free default theory $\langle D, W\rangle$ where W is a set of Horn clauses (resp. 2-literal clauses) is solvable in $\mathcal{O}(n^2)$ time, where n is the sum of the sizes of the theory and ϕ.*

Proof: Theories with disjunction-free default rules are translated into autoepistemic logic using the translation

$$tr_{dlsn}\langle D, W\rangle = \{L\alpha_1 \wedge \cdots \wedge L\alpha_n \wedge \neg L \neg\beta \rightarrow \gamma \mid \frac{\alpha_1 \wedge \cdots \wedge \alpha_n : \beta \wedge \gamma}{\gamma} \in D\} \cup W$$

and we define that a disjunction-free default theory is stratified if the set of autoepistemic formulae obtained by the tr_{dlsn} translation is stratified. The translation differs in two respects from the Konolige translation given in Theorem 4.3. First, the translation ignores the normality part of the justification. Second, we have replaced $L(\alpha_1 \wedge \cdots \wedge \alpha_n)$ by the equivalent (in autoepistemic logic) formula $L\alpha_1 \wedge \cdots \wedge L\alpha_n$.

We show that in the computation of the algorithm of Theorem 3.10 all $\models_L$-consequence tests reduce to satisfiability testing of Horn-clauses when W is a set of Horn clauses, and to satisfiability testing of 2-literal clauses when W is a set of 2-literal clauses. Satisfiability testing in these cases is $\mathcal{O}(n)$ time [Dowling and Gallier, 1984; Aspvall *et al.*, 1979]. The sets R_i (see Theorem 3.10) consist of formulae that are in W and formulae that are consequenes of default rules in D. Formulae in W are Horn clauses (resp. 2-literal clauses) and the consequences of disjunction-free default rules are conjunctions of literals $\gamma_1 \wedge \cdots \wedge \gamma_n$ which are equivalent to sets of literals $\{\gamma_1, \ldots, \gamma_n\}$. Single literals are a special case of Horn clauses (resp. 2-literal clauses). Hence, the sets R_i consist of Horn clauses (resp. 2-literal clauses).

In the computation the $\models_L$-consequence of formulae inside L from a set R_i is tested. The formulae inside L are single literals α. By definition, testing $R_i \models_L \alpha$ reduces to $R_i \cup SB_{R_i}(L\alpha) \models L\alpha$, which reduces to $R_i \models \alpha$, i.e., a classical logical consequence test of a literal α from a set of formulae R_i of classical propositional logic. This logical consequence test can be implemented as a satisfiability test of $R_i \cup \{\neg\alpha\}$, which is a set of Horn clauses (resp. 2-literal clauses).

Hence, by the complexity analysis given in Section 3.3 the computation of the full set associated with the disjunction-free default theory $\langle D, W\rangle$ is $\mathcal{O}(n^2)$ time, where n is the size of the default theory. The membership of formulae ϕ the negations of which are logically equivalent to finite sets of Horn clauses (resp. 2-literal clauses) in the unique extension can be tested in $\mathcal{O}(m)$ time, where m is the sum of the sizes of $\langle D, W\rangle$ and ϕ. All this is $\mathcal{O}(m^2)$ time. $\square$

We conclude that as in stratified autoepistemic logic, also in stratified default logic the complexity of reasoning is determined by the complexity of the underlying classical reasoning. The stratified counterparts of the intractable restricted classes of default theories identified by [Kautz and Selman, 1991; Stillman, 1990] are all tractable, and the possibility of further identification of such classes is immediate just like in autoepistemic logic.

5 Implementation of the Decision Procedure

The iterative decision procedure of stratified autoepistemic theories described in Section 3 and its $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ subcases have been implemented in Prolog. The implementation of the algorithms and related programs are described in this section and the program listings are given in the appendix. The division of the program code into files is as follows.

file	contents
algorithm.pl	the general algorithm of Theorem 3.10
lineartime.pl	predicates for the polynomial time classes,
	satisfiability testing of propositional Horn clauses
sat.pl	satisfiability testing of arbitrary propositional formulae

5.1 Representation of Formulae

Formulae of propositional autoepistemic logic are usually represented as follows.

$af(P)$	a propositional variable P
$op(-, \phi)$	$\neg\phi$
$op('L', \phi)$	$L\phi$
$co(\otimes, \phi, \psi)$	$\phi \oplus \psi$, $\oplus = \wedge, \vee, \rightarrow, \leftrightarrow$ respectively for $\otimes =$ '&','V','=>','<=>'

The formulae ϕ in stratified sets are of the form $a(\phi) \wedge o(\phi) \rightarrow c(\phi)$, and are represented as

$$sf(A, O, C)$$

where A is a list of formulae, O is either a formula with no occurrences of L or the atom *none*, and C is a formula with no occurrences of L. For the polynomial time classes discussed in Section 3.3 more specialized representations are used.

5.2 Computation of a Stratification

The $\mathcal{O}(n)$ strong components algorithm of Section 3.1 has been translated in a straigthforward way to Prolog. Prolog does not have arrays or similar data structures with arbitrary size and $\mathcal{O}(1)$ access time to arbitrary elements, and therefore arrays with logarithmic access time implemented as balanced trees has been used instead. The asymptotic time penalty due to this lifts the time complexity of the program to $\mathcal{O}(n \log n)$. This program has been used in the

$$\frac{P \vee Q}{P \mid Q} \quad \frac{P \rightarrow Q}{\neg P \mid Q} \quad \frac{\neg(P \wedge Q)}{\neg P \mid \neg Q}$$

$$\frac{P \wedge Q}{\begin{array}{c} P \\ Q \end{array}} \quad \frac{\neg(P \vee Q)}{\begin{array}{c} \neg P \\ \neg Q \end{array}} \quad \frac{\neg(P \rightarrow Q)}{\begin{array}{c} P \\ \neg Q \end{array}} \quad \frac{\neg\neg P}{P}$$

Figure 6: Rules of the Analytic Tableau for Propositional Logic

implementation of the following predicate for computing a stratification for a set of formulae.

```
stratification(Formulae,Stratification)
```

If the set `Formulae` is stratified, the predicate succeeds and `Stratification` is instantiated to a list in which each element corresponds to one stratum of the stratification. Each element is the list of formulae in the corresponding stratum.

5.3 Testing for Logical Consequence

The general implementation of the algorithm for computing the Σ-full sets for stratified sets Σ uses an algorithm for satisfiability testing that is based on the method of analytic tableau of Smullyan [1968]. Smullyan's tableau is a variant of tableau methods presented earlier by Beth and Hintikka, and it can be seen as a method of constructing proofs in Gentzen's sequent calculus in bottom-up direction. Analytic tableau is based on the set of rules depicted in Figure 6 by using which a proof tree is constructed. The root of the tree is the negation of a formula to be proved, and if the formula was indeed provable every branch of the proof tree contains a complementary pair, i.e., a propositional variable and the negation of that variable. The following predicate uses Smullyan's tableau and can be used for the satisfiability testing of finite sets of propositional formulae.

```
satisfiable(Formula,Satisfiable)
```

`Formula` is a formula in the format described earlier, and `Satisfiable` is *true* if the formula is satisfiable, and *false* if it is not. The main predicate used in the implementation of *satisfiable* is

```
analyze(Formula,Queue,NewQueue)
```

which corresponds to the rules of the tableau shown in Figure 6. `Formula` is a formula and `NewQueue` is the priority queue `Queue` extended with the formulae obtained from `Formula` by the rules. As in [Smullyan, 1968] on p. 18, α formulae (formulae that do not cause the tree to branch) are given a higher priority than β formulae. This way the branching is postponed as far as possible, and as a consequence the size of the tree will be smaller.

5.4 Computation of Stable Expansions

The general algorithm for computing a set characterizing the unique stable expansion of a stratified set is implemented as the predicate

```
reduct(Strata,Reduct)
```

where `Strata` is a stratification of a strafied set as returned by the predicate `stratification`, and `Reduct` is a set of objective formulae, i.e., formulae of the classical propositional calculus, that characterizes the unique stable expansion as shown in Theorem 3.10. The main predicate used in the implementation of `reduct` is

```
l_o_consequence(Formula,Theory,TF)
```

which tests the $\models_L$-consequence of the formula `Formula` from an objective theory `Theory`. If Theory $\models_L$ Formula then `TF` is instantiated to `true`, otherwise to `false`.

5.5 Predicates for the Polynomial-Time Classes

The predicates

```
shc(Formulae)
spc(Formulae)
```

check whether the formulae in the set `Formulae` are of the form allowed in the classes SHC_{ae} and SPC_{ae}, respectively. The next predicate computes the set `Variables` of propositional variables in the unique stable expansion of `Strata`, where `Strata` is a member of SPC_{ae}. Membership of formulae in CF_{ae} can be tested using the predicate `holds`.

```
spcReduct(Strata,Variables)
holds(Variables,Formula)
```

The implementation of `spcReduce` is based on the predicate

```
dg(Hornclauses,Variables1,Variables2)
```

that correspond to the function DG presented in Proposition 3.20. The predicate `spcReduct` below computes for a set of formulae in SHC_{ae} its reduct `HCReduct` which is a set of Horn clauses. The membership of of formulae in HF_{ae} can be tested using the predicate `shc_l_o_consequence` that instantiates the variable `TrueFalse` to *true* or *false* depending on whether `Formula` belongs to the stable expansion determined by the reduct `HCReduct`.

```
shcReduct(Strata,HCReduct)
shc_l_o_consequence(Formula,HCReduct,TrueFalse)
```

Given the predicates described in this section, the membership of an arbitrary formula $\chi \in \mathcal{L}_{ae}$ in the unique stable expansion of a stratified theory Σ can be tested using the query

```
stratification(Σ,S), reduct(S,R), l_o_consequence(χ,R,true)
```

5.6 An Example Application: The Yale Shooting Problem

As an example application we present a solution to the Yale Shooting Problem of Hanks and McDermott [1986] which was constructed to show that the solution of the frame problem using nonmonotonic logics is not as trivial as had been thought earlier.

The Yale Shooting problem is formalized as a simple situation calculus which is embedded in ordinary first-order logic. The binary predicate *holds(F,S)* is used to state that F is a fact that holds in situation *S*. The result of the function application *result(A,S)* is a new situation that represents the situation S after performing the action A. For example, the fact that loading a gun causes the gun become loaded can be stated as the axiom

$$\forall \mathrm{S}\ \mathrm{holds(loaded,result(load,S))}.$$

Hanks and McDermott use the abnormality predicate and the axiom of inertia proposed by [McCarthy, 1986] for the solution of the frame problem. The axiom of inertia says that an action A does not change a fact P unless it is abnormal for P not to change by A.

$$\forall P,A,S\ \neg ab(P,A,S) \wedge holds(P,S) \rightarrow holds(P,result(A,S))$$

For facts that really are changed by an action there are axioms stating that it is abnormal for them not to change. Now the task of stating the abnormalities that do not hold seems to require great numbers of axioms, but this is something that can be done using nonmonotonic formalisms: in circumscription the abnormality predicate is minimized thus minimizing the abnormal changes of facts.

Hanks and McDermott's scenario consists of a person called John and a gun pointed at John, and the actions of loading the gun and shooting at John. The axioms for the scenario are the following ones.

$$\begin{array}{l} holds(alive,s0) \\ \forall S\ holds(loaded, result(load,S)) \\ \forall S\ holds(loaded, S) \rightarrow holds(dead,result(shoot,S)) \\ \forall S\ holds(loaded, S) \rightarrow ab(alive,shoot,S) \\ \forall P,A,S\ \neg ab(P,A,S) \wedge holds(P,S) \rightarrow holds(P,result(A,S)) \end{array}$$

Here $s0$ is the initial situation of the world where John is alive. We are interested in the sequence of events *load*, *wait*, and *shoot*, and the final situation *result(shoot,result(wait,result(load,s0)))*. Because loading the gun causes the gun to become loaded, and after shooting with a loaded gun John will be dead, we expect that in the final situation John is indeed dead. But the formalizations of these axioms in circumscription , default logic, and nonmonotonic logic of McDermott and Doyle, yield also another possible outcome: John does not die because the gun becomes unloaded during waiting. The technical reason for this is that e.g. in circumscription where the abnormalities are minimized by minimizing the predicate *ab*, there are surprisingly two minimal models with respect to *ab*: in the first model it is abnormal to survive a shot from a gun, in the second unexpected one it is abnormal for a gun to stay loaded during the action of waiting. Consequently Hanks and McDermott claimed that these at that time leading formalizations of nonmonotonic reasoning are "...inherently incapable of representing this kind of default reasoning". This claim raised several responses trying to show that the problem is irrelevant or that the scenario can be correctly formalized using other formalisms or formalizations.

Gelfond [1988] shows that autoepistemic logic gives the desired conclusions about the shooting scenario if the last formula of the problem statement is modified to

$$\forall\ P,A,S\ \neg L ab(P,A,S) \wedge holds(P,S) \rightarrow holds(P,result(A,S)).$$

We use this formalization of the Yale Shooting Problem to test the implementations of the algorithms we have developed for stratified autoepistemic theories. The problem is stated in the quantificational version of autoepistemic logic, but we can remove the quantifiers and instantiate the free variables to obtain a set of formulae the atomic sentences of which can be treated like propositional variables. The instantiation process means substituting the relevant terms for the free variables occurring in the formulae. The relevant actions over which variables A vary are *load*, *wait*, and *shoot*, the relevant situations for variables S are *s0*, *result(load,s0)*, *result(wait,result(load,s0))*, and *result(shoot,result(wait,result(load,s0)))*, and the relevant facts are *loaded*, *dead*, and *alive*. The ground instantiation of the problem consists of 49 formulae in which 75 distinct propositional variables occur.

Although the original set is not stratified because the predicates *ab* and *holds* are defined using each other through L, its ground instantiation is. Furthermore, the formulae are of the simple form allowed in the linear time class SPC_{ae}. The stratification of the ground instantiation of the problem is depicted[2] in Figure 7 on page 53. The intuitive reason for the stratifiedness for the set of formulae is that the consequents of the implications describe the state of affairs in a later time than the antecedents, and because there are no cycles in the causal chains the formulae whose consequents describe earlier situations are lower in the stratification.

In Figure 8 some execution times are shown. All times are for program code compiled on Sparcstation ELC in Sicstus Prolog in native code compilation mode. The times have been measured using an internal facility of Sicstus Prolog. The first row identifies the problem, the second gives the number of distinct propositional variables in the problem, the third the number of formulae, the fourth the execution time of the algorithm computing a stratification for the set of formulae, and the fifth column gives the execution time of computing the propositional variables that are belong to the unique stable expansion of the set of formulae. All times are in seconds and are approximate.

The first row named YSP gives the runtimes of the solution to Yale Shooting Problem described above. Rows named 1,2,3 describe the execution of other variants obtained from the formula scheme of the Yale Shooting Problem. Instead of considering only the four relevant situations, all sequences of the actions *load*, *wait*, *shoot* of certain length are considered. This way the number of propositional variables and the number of formulae are significantly larger. The times for these modified problems are shown in Figure 8 as rows 1–3. The first example is for arbitrary sequences of three actions, the second for

[2] For typographical reasons the predicates *holds* and *results* are abbreviated as h and r, respectively, and the actions *load*, *wait*, and *shoot* as l, w, and s, respectively.

1	$h(alive, s0)$
2	$h(loaded, s0) \rightarrow h(dead, r(s, s0))$ $\neg Lab(dead, s, s0) \land h(dead, s0) \rightarrow h(dead, r(s, s0))$
3	$h(loaded, s0) \rightarrow ab(alive, s, s0)$
4	$\neg Lab(alive, s, s0) \land h(alive, s0) \rightarrow h(alive, r(s, s0))$
5	$\neg Lab(loaded, s, s0) \land h(loaded, s0) \rightarrow h(loaded, r(s, s0))$
6	$\neg Lab(dead, w, s0) \land h(dead, s0) \rightarrow h(dead, r(w, s0))$
7	$\neg Lab(alive, w, s0) \land h(alive, s0) \rightarrow h(alive, r(w, s0))$
8	$\neg Lab(loaded, w, s0) \land h(loaded, s0) \rightarrow h(loaded, r(w, s0))$
9	$\neg Lab(dead, l, s0) \land h(dead, s0) \rightarrow h(dead, r(l, s0))$
10	$\neg Lab(alive, l, s0) \land h(alive, s0) \rightarrow h(alive, r(l, s0))$
11	$h(loaded, r(l, s0))$ $\neg Lab(loaded, l, s0) \land h(loaded, s0) \rightarrow h(loaded, r(l, s0))$
12	$h(loaded, r(l, s0)) \rightarrow h(dead, r(s, r(l, s0)))$ $\neg Lab(dead, s, r(l, s0)) \land h(dead, r(l, s0)) \rightarrow h(dead, r(s, r(l, s0)))$
13	$h(loaded, r(l, s0)) \rightarrow ab(alive, s, r(l, s0))$
14	$\neg Lab(alive, s, r(l, s0)) \land h(alive, r(l, s0)) \rightarrow h(alive, r(s, r(l, s0)))$
15	$\neg Lab(loaded, s, r(l, s0)) \land h(loaded, r(l, s0)) \rightarrow h(loaded, r(s, r(l, s0)))$
16	$\neg Lab(dead, w, r(l, s0)) \land h(dead, r(l, s0)) \rightarrow h(dead, r(w, r(l, s0)))$
17	$\neg Lab(alive, w, r(l, s0)) \land h(alive, r(l, s0)) \rightarrow h(alive, r(w, r(l, s0)))$
18	$\neg Lab(loaded, w, r(l, s0)) \land h(loaded, r(l, s0)) \rightarrow h(loaded, r(w, r(l, s0)))$
19	$\neg Lab(dead, l, r(l, s0)) \land h(dead, r(l, s0)) \rightarrow h(dead, r(l, r(l, s0)))$
20	$\neg Lab(alive, l, r(l, s0)) \land h(alive, r(l, s0)) \rightarrow h(alive, r(l, r(l, s0)))$
21	$h(loaded, r(l, r(l, s0)))$ $\neg Lab(loaded, l, r(l, s0)) \land h(loaded, r(l, s0)) \rightarrow h(loaded, r(l, r(l, s0)))$
22	$h(loaded, r(w, r(l, s0))) \rightarrow h(dead, r(s, r(w, r(l, s0))))$ $\neg Lab(dead, s, r(w, r(l, s0))) \land h(dead, r(w, r(l, s0))) \rightarrow h(dead, r(s, r(w, r(l, s0))))$
23	$h(loaded, r(w, r(l, s0))) \rightarrow ab(alive, s, r(w, r(l, s0)))$
24	$\neg Lab(alive, s, r(w, r(l, s0))) \land h(alive, r(w, r(l, s0))) \rightarrow h(alive, r(s, r(w, r(l, s0))))$
25	$\neg Lab(loaded, s, r(w, r(l, s0))) \land h(loaded, r(w, r(l, s0))) \rightarrow h(loaded, r(s, r(w, r(l, s0))))$
26	$\neg Lab(dead, w, r(w, r(l, s0))) \land h(dead, r(w, r(l, s0))) \rightarrow h(dead, r(w, r(w, r(l, s0))))$
27	$\neg Lab(alive, w, r(w, r(l, s0))) \land h(alive, r(w, r(l, s0))) \rightarrow h(alive, r(w, r(w, r(l, s0))))$
28	$\neg Lab(loaded, w, r(w, r(l, s0))) \land h(loaded, r(w, r(l, s0))) \rightarrow h(loaded, r(w, r(w, r(l, s0))))$
29	$\neg Lab(dead, l, r(w, r(l, s0))) \land h(dead, r(w, r(l, s0))) \rightarrow h(dead, r(l, r(w, r(l, s0))))$
30	$\neg Lab(alive, l, r(w, r(l, s0))) \land h(alive, r(w, r(l, s0))) \rightarrow h(alive, r(l, r(w, r(l, s0))))$
31	$h(loaded, r(l, r(w, r(l, s0))))$ $\neg Lab(loaded, l, r(w, r(l, s0))) \land h(loaded, r(w, r(l, s0))) \rightarrow h(loaded, r(l, r(w, r(l, s0))))$
32	$h(loaded, r(s, r(w, r(l, s0)))) \rightarrow h(dead, r(s, r(s, r(w, r(l, s0)))))$ $\neg Lab(dead, s, r(s, r(w, r(l, s0)))) \land h(dead, r(s, r(w, r(l, s0)))) \rightarrow h(dead, r(s, r(s, r(w, r(l, s0)))))$
33	$h(loaded, r(s, r(w, r(l, s0)))) \rightarrow ab(alive, s, r(s, r(w, r(l, s0))))$
34	$\neg Lab(alive, s, r(s, r(w, r(l, s0)))) \land h(alive, r(s, r(w, r(l, s0)))) \rightarrow h(alive, r(s, r(s, r(w, r(l, s0)))))$
35	$\neg Lab(loaded, s, r(s, r(w, r(l, s0)))) \land h(loaded, r(s, r(w, r(l, s0)))) \rightarrow h(loaded, r(s, r(s, r(w, r(l, s0)))))$
36	$\neg Lab(dead, w, r(s, r(w, r(l, s0)))) \land h(dead, r(s, r(w, r(l, s0)))) \rightarrow h(dead, r(w, r(s, r(w, r(l, s0)))))$
37	$\neg Lab(alive, w, r(s, r(w, r(l, s0)))) \land h(alive, r(s, r(w, r(l, s0)))) \rightarrow h(alive, r(w, r(s, r(w, r(l, s0)))))$
38	$\neg Lab(loaded, w, r(s, r(w, r(l, s0)))) \land h(loaded, r(s, r(w, r(l, s0)))) \rightarrow h(loaded, r(w, r(s, r(w, r(l, s0)))))$
39	$\neg Lab(dead, l, r(s, r(w, r(l, s0)))) \land h(dead, r(s, r(w, r(l, s0)))) \rightarrow h(dead, r(l, r(s, r(w, r(l, s0)))))$
40	$\neg Lab(alive, l, r(s, r(w, r(l, s0)))) \land h(alive, r(s, r(w, r(l, s0)))) \rightarrow h(alive, r(l, r(s, r(w, r(l, s0)))))$
41	$h(loaded, r(l, r(s, r(w, r(l, s0)))))$ $\neg Lab(loaded, l, r(s, r(w, r(l, s0)))) \land h(loaded, r(s, r(w, r(l, s0)))) \rightarrow h(loaded, r(l, r(s, r(w, r(l, s0)))))$

Figure 7: A Stratification of The Yale Shooting Problem

problem	variables	formulae	stratification (s)	consequences (s)
YSP	75	49	0.39	0.12
1	723	481	1.33	0.41
2	2181	1453	4.51	1.33
3	6555	4369	14.5	4.34

Figure 8: Execution Times for a Number of Examples

four actions, and the third for five actions.

Execution times do not grow linearly because of the $\mathcal{O}(n \log n)$ implementation of arrays, and memory consumption in running these examples was fairly high. Arrays with constant access time and destructive updates would reduce both execution times and memory consumption significantly.

6 Conclusions

In this work we have discussed some of the issues involved in making more expressive and efficient methods for knowledge representation available. One of the central theoretical problems in knowledge representation are notions of defaults and negation as unprovability, which by their nature require nonmonotonicity. Hence they cannot be naturally formalized in classical logics, which are monotonic. Knowledge representation is the application area of nonmonotonic logics.

Nonmonotonic reasoning has been shown to be almost certainly more complex than classical logical reasoning [Gottlob, 1991; Stillman, 1992], and intractable even in syntactically very restricted cases [Kautz and Selman, 1991; Stillman, 1990]. The contribution of this work is the elucidation of the impact of stratification on the complexity of nonmonotonic reasoning. The framework in which the results are established is propositional autoepistemic logic. We have given an algorithm for computing the unique stable expansions of stratified autoepistemic theories, and by restricting the complexity of the classical reasoning required by that algorithm, we have been able to identify tractable classes of autoepistemic theories. The algorithm is iterative, i.e., it does not involve backtracking unlike similar algorithms for the general unstratified case, and its effectiveness in achieving tractability can be traced to this very property. The tools applied in developing automated theorem-proving methods for nonmonotonic logics have been very different from the ones in developing theorem-provers for classical logics. For example, Junker and Konolige [1990] base their theorem-prover for default logic on truth-maintenance systems, and Ben-Eliyahu and Dechter [1992a; 1992b] on techniques developed for solving constraint networks. For our algorithm we present techniques that directly lead to implementations that use theorem-provers for classical logics as subroutines (this is possible also in the more general unstratified case, see [Niemelä, 1990]).

As autoepistemic logic is closely related to several other nonmonotonic formalisms, the tractability results given here in autoepistemic logic imply similar results e.g. for propositional default logic, propositional logic programs, and truth-maintenance systems. To evaluate the effect of stratification on the complexity of nonmonotonic reasoning the restricted default theories investigated by [Kautz and Selman, 1991; Stillman, 1990] have been compared to the attained complexity results. The stratified counterparts of the syntactically very restricted but still mostly intractable classes all prove to be tractable, and most are in fact subclasses of our quadratic-time class.

Although stratified theories in autoepistemic logic and default logic have many good properties, there are examples showing that stratification is sometimes too strong a restriction. Some of these examples give rise to only one set of conclusions, but for some applications the existence of several alternative sets of conclusions is exactly what is desired. As seems to be always the case in knowledge representation, there is a trade-off between expressivity and complexity also with stratification. Stratification should be seen as something to which one should confine to – because of the good complexity properties – if possible. Unfortunately, not all data and knowledge can be represented in stratified form and then one has to resort to more general procedures with inferior time complexity.

For future research several possibilities exist. The notion of stratification in the context of nonmonotonic logics is not yet completely understood. The definition of stratification used in this work can be seen as a straightforward generalization of the corresponding definition for logic programs. This definition can be further generalized and other similar structural properties can supposedly be exhibited with different degrees of favourable properties. Known results about the complexity of reasoning in default logic indicate that the restriction on the syntactic form of sentences of a theory does not alone yield tractability and that also stratification or some similar structural condition is needed. However, not all theories are stratified or stratifiable and one direction of research could be the investigation of the possibilities of reducing this kind of theories to essentially stratified ones. Such reduction could be based on an analysis of the self reference through negation in the theories and on some extra-logical means of breaking the cycles, e.g. priorities. Our restriction to the propositional cases of nonmonotonic reasoning in this work is not completely unproblematic. Solution of many practical problems requires the notions of relation, variable, constant, and function. How the results are best extended to languages supporting these notions should also be studied.

Acknowledgements

This report is the result of my work in the Digital Systems Laboratory at Helsinki University of Technology in 1992. I am grateful to Ilkka Niemelä for helpful discussions and comments, and especially for pointing out the equations of Theorem 3.6. Other parts of the work to which Ilkka has especially contributed are the *Red* operator of Lemma 3.8 and the proof of Theorem 4.3. Results of Section 3 have been published as the conference paper [Niemelä and Rintanen, 1992].

References

[Aho *et al.*, 1974] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

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

[Aspvall *et al.*, 1979] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

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

[Ben-Eliyahu and Dechter, 1992a] R. Ben-Eliyahu and R. Dechter. Inference in inheritance networks using propositional logic and constraint network techniques. In *Proceedings of Artificial Intelligence '92 Conference*, Vancouver, 1992.

[Ben-Eliyahu and Dechter, 1992b] R. Ben-Eliyahu and R. Dechter. Propositional semantics for default logic. In *The Fourth International Workshop on Nonmonotonic Reasoning*, pages 13–27, Vermont, May 1992.

[Bidoit and Froidevaux, 1991] N. Bidoit and C. Froidevaux. General logical databases and programs: Default logic semantics and stratification. *Information and Computation*, 91:15–54, 1991.

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

[Dowling and Gallier, 1984] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[Elkan, 1990] C. Elkan. A rational reconstruction of nonmonotonic truth maintenance systems. *Artificial Intelligence*, 43:219–234, 1990.

[Etherington, 1987] D. W. Etherington. Formalizing nonmonotonic reasoning systems. *Artificial Intelligence*, 31:41–85, 1987.

[Gallo and Scutellà, 1988] G. Gallo and M. G. Scutellà. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29:221–226, November 1988.

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

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, August 1988. The MIT Press.

[Gelfond, 1987] M. Gelfond. On stratified autoepistemic theories. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 207–211, Seattle, July 1987. American Association for Artificial Intelligence.

[Gelfond, 1988] M. Gelfond. Autoepistemic logic and formalization of commonsense reasoning, preliminary report. In *Proceedings of the 2nd Workshop on Non-Monotonic Reasoning*, pages 176–186, Grassau, Germany, June 1988. Springer-Verlag.

[Gottlob, 1991] G. Gottlob. Complexity results for nonmonotonic logics. Technical Report CD-TR 91/24, Christian Doppler Laboratory for Expert Systems, Institut für Informationssysteme, Technische Universität Wien, Vienna, August 1991.

[Hanks and McDermott, 1986] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logic, and frame problem. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 328–333, Philadelphia, August 1986.

[Hayes-Roth, 1987] F. Hayes-Roth. Rule-based systems. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 2, pages 963–973. John Wiley & Sons, 1987.

[Johnson, 1990] D. S. Johnson. A catalog of complexity classes. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A. Algorithms and Complexity, pages 67–161. Elsevier Science Publisher, 1990.

[Junker and Konolige, 1990] U. Junker and K. Konolige. Computing the extensions of autoepistemic and default logics with a truth maintenance system. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 278–283, Boston, 1990. Morgan Kaufmann Publishers.

[Kakas and Mancarella, 1990] A. C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In L. C. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 385–391, Stockholm, August 1990. Pitman Publishing.

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

[Konolige, 1988] K. Konolige. On the relation between default and autoepistemic logic. *Artificial Intelligence*, 35:343–382, 1988.

[Konolige, 1989] K. Konolige. On the relation between autoepistemic logic and circumscription. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1213–1218, Detroit, August 1989. Morgan Kaufmann Publishers.

[Lassez *et al.*, 1987] C. Lassez, K. McAloon, and G. Port. Stratification and knowledge base management. In *Proceedings of the 4th International Conference on Logic Programming*, volume 1, pages 136–151, 1987.

[Marek and Truszczyński, 1989] W. Marek and M. Truszczyński. Relating autoepistemic and default logics. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*, pages 276–288, Toronto, May 1989.

[Marek and Truszczyński, 1991] W. Marek and M. Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38:588–619, 1991.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[McCarthy, 1980] J. McCarthy. Circumscription – a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[McCarthy, 1986] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

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

[Moore, 1983] R. C. Moore. Semantical considerations on nonmonotonic logic. In A. Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 272–279, Karlsruhe, Germany, August 1983. Morgan Kaufmann Publishers.

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

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

[Niemelä, 1990] I. Niemelä. Towards automatic autoepistemic reasoning. In J. Van Eijck, editor, *Proceedings of the European Workshop on Logics in Artificial Intelligence—JELIA'90*, number 478 in Lecture Notes in Artificial Intelligence, pages 428–443, Amsterdam, September 1990. Springer-Verlag.

[Niemelä, 1991] I. Niemelä. Constructive tightly grounded autoepistemic reasoning. In J. Mylopoulos, editor, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 399–404, Sydney, August 1991. Morgan Kaufmann Publishers.

[Przymusinska and Przymusinski, 1990] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 321–367. North-Holland, Amsterdam, 1990.

[Przymusinski, 1988] T. Przymusinski. Declarative semantics of deductive databases. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers, 1988.

[Reiter, 1978] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, New York, 1978.

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

[Schaefer, 1978] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.

[Smullyan, 1968] R.M. Smullyan. *First-Order Logic*. Springer-Verlag, Berlin, 1968.

[Stalnaker, 1980] R. Stalnaker. A note on non-monotonic modal logic. Department of Philosophy, Cornell University. Unpublished, 1980.

[Stillman, 1990] J. Stillman. It is not my default: the complexity of reasoning in default logic. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 571–578, Boston, August 1990. Morgan Kaufmann Publishers.

[Stillman, 1992] J. Stillman. The complexity of propositional default logics. In *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 794–799, San Jose, CA, 1992.

[Tarjan, 1972] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing*, 1(2):146–160, 1972.

[Urbanski, 1988] A. Urbanski. Formalizing non-monotonic truth maintenance systems. In T. O'Shea and V. Sgurev, editors, *Artificial Intelligence III: Methodology, Systems, Principles*, pages 43–50. Elsevier Science Publishers, 1988.

[Van Gelder *et al.*, 1988] A. Van Gelder, K. A. Ross, and J. S. Schlipf. Unfounded sets and the well-founded semantics for general logic programs. In *Proceedings of the 7th Symposium on Principles of Database Systems*, pages 221–230, Austin, Texas, March 1988. ACM Press.

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

Appendix: Program Listings

File algorithm.pl

```
/* May 1992, Jussi Rintanen, Helsinki University of Technology

reduct(Strata,Reduct) if Reduct is the reduct of a stratified
set which is stratified as Strata.

The stratification is represented as a list of strata, each strata
being a list of formulae of the form sf(A,O,C). The first element in
the stratification list is the lowest stratum.

*/

reduct(Strata,Reduct) :- reduct([],Strata,Reduct).

reduct(R,[],R).
reduct(Reduct0,[Stratum|Strata],Reduct) :-
        reduce(Stratum,Reduct0,Reduct0,Reduct1),
        reduct(Reduct1,Strata,Reduct).

% Removing the autoepistemic parts A containing the L operator from
% a single stratum of a stratification

reduce([],_,Reduct,Reduct).
reduce([sf(A,O,C)|Fs],Lower,Reduct0,Reduct) :-
        fulfilled(A,Lower,TF),
        reduce1(O,C,TF,Fs,Lower,Reduct0,Reduct).
reduce1(none,C,true,Fs,Lower,Reduct0,Reduct) :- !,
        reduce(Fs,Lower,[C|Reduct0],Reduct).
reduce1(O,C,true,Fs,Lower,Reduct0,Reduct) :-
        reduce(Fs,Lower,[co('=>',O,C)|Reduct0],Reduct).
reduce1(_,_,false,Fs,Lower,Reduct0,Reduct) :-
        reduce(Fs,Lower,Reduct0,Reduct).

% Testing whether the formulae inside L in A are L-consequences of
% the (already reduced) lower levels of the stratification

fulfilled([],_,true).
fulfilled([op('L',E)|Rest],Lower,TF) :-
        l_o_consequence(E,Lower,TF1),
        fulfilled1(Rest,Lower,TF1,TF).
fulfilled([op(-,op('L',E))|Rest],Lower,TF) :-
        l_o_consequence(E,Lower,TF1),
        converse(TF1,TF0),
        fulfilled1(Rest,Lower,TF0,TF).

fulfilled1(_,_,false,false).
fulfilled1(Rest,Lower,true,TF) :- fulfilled(Rest,Lower,TF).

% L-consequence of an arbitrary formula from an objective theory
```

```
%
% l_o_consequence(Formula,Theory)

l_o_consequence(Formula,Theory,TF) :-
        objectivize(Formula,Theory,Formula2),
        F = op(-,Formula2),
        satisfiable1([F|Theory],TF0),
        converse(TF0,TF).

satisfiable1([H|T],TF) :-
        conjunctivize([H|T],F),
        satisfiable(F,TF).

% A trick for making the L-consequence testing more efficient is to
% replace the subformulae Lp by the logical constant True (resp. False)
% when p has (resp. has not) been found to be a L-consequence of the
% set of formula. Further details in the thesis.

objectivize(C,_,C) :- C=af(_).
objectivize(op(-,F), Theory, op(-,OF)) :- objectivize(F,Theory,OF).
objectivize(co(C,A,B), Theory, co(C,OA,OB)) :-
        objectivize(A,Theory,OA), objectivize(B,Theory,OB).
objectivize(op('L',F), Theory, OF) :-
        l_o_consequence(F,Theory, TF),
        objectivize1(TF, OF).
objectivize1(true, co('V',af(a),op(-,af(a)))).          % T
objectivize1(false, co('&',af(a),op(-,af(a)))).         % F
```

File lineartime.pl

```
/* May 1992, Jussi Rintanen, Helsinki University of Technology

Predicates for the polynomial time classes of stratified autoepistemic logic.

*/

% Definitions of the polynomial time classes

% Quadratic time class SHC

shc([]).
shc([[]|Ss]) :- shc(Ss).
shc([[sf(A,O,C)|Fs]|Ss]) :-
        shc([Fs|Ss]),
        hfList(A),
        propositionConjunction(O),
        literal(C).

% Linear time class SPC

spc([]).
```

```
spc([[]|Ss]) :- spc(Ss).
spc([[sf(A,O,C)|Fs]|Ss]) :-
        spc([Fs|Ss]),
        cfList(A),
        propositionConjunction(O),
        C = af(_).

literal(af(_)).
literal(op(-,af(_))).

propConjunction(none).
propConjunction(af(_)).
propConjunction(co(&,A,B)) :- propConjunction(A), propConjunction(B).

% Class of formulae HF (used in SHC)

hfList([]).
hfList([F|Fs]) :- conjunctivize([F|Fs],C), hf(C).

hf(co(&,A,B)) :- hfDisjunct(co('&',A,B),0,_).
hf(co('V',A,B)) :- hf(A), hf(B).
hf(af(_)).
hf(op('L',A)) :- hf(A).

hfDisjunct(af(_),Flag,Flag).
hfDisjunct(op(-,af(_)),0,1).
hfDisjunct(op('L',A),Flag,Flag) :- hf(A).
hfDisjunct(op(-,op('L',A)),Flag,Flag) :- hf(A).
hfDisjunct(co(&,A,B),Flag,Flag1) :-
        hfDisjunct(A,Flag,Flag0),
        hfDisjunct(B,Flag0,Flag1).

% Class of formulae CF (used in SPC)

cfList([]).
cfList([F|Fs]) :- conjunctivize([F|Fs],C), cf(C).

cf(af(_)).
cf(co(&,A,B)) :- cf(A), cf(B).
cf(op('L',A)) :- cf(A).
cf(op(-,op('L',A))) :- cf(A).

%
% Procedure of Theorem 4.27
%

spcReduct(Stratification,Variables) :-
        new_array(V),
        spcReduce(V,Stratification,Variables).

spcReduce(V,[],V).
spcReduce(V,[Stratum|Strata],V2) :-
        spcReduct(V,Stratum,PClauses),
```

```
        dg(PClauses,V,V1),
        spcReduce(V1,Strata,V2).
spcReduct(_,[],[]).
spcReduct(V,[sf(A,O,af(C))|Fs],[(Lhs,C)|PClauses]) :-
        holds(V,A), !,
        variables(O,[],Lhs),
        spcReduct(V,Fs,PClauses).
spcReduct(V,[_|Fs],PClauses) :- spcReduct(V,Fs,PClauses).

variables(none,L,L).
variables(af(V),Ac,[V|Ac]).
variables(co(&,A,B),Ac,Ac1) :-
        variables(A,Ac,Ac0),
        variables(B,Ac0,Ac1).

holds(_,[]).
holds(V,[A|As]) :- holds1(V,A), holds(V,As).
holds1(V,af(Var)) :- arefl(Var,V,1).
holds1(V,co(&,A,B)) :- holds1(V,A), holds1(V,B).
holds1(V,op(-,op('L',A))) :- not(holds1(V,A)).
holds1(V,op('L',A)) :- holds1(V,A).

%
% A nlogn version of DG (Algorithm 2 of [Dowling and Gallier])
%
% Hornclauses is a set of terms of the form ([n1,n2,n3,n4,...,nm],p)
% where ni are greater than zero and p is greater than or equal to zero
%

dg(Hornclauses,Base,Base1) :-
        new_array(Numargs),
        new_array(Poslitlist),
        new_array(CL),
        initDG(0,Hornclauses,Base,Numargs,Poslitlist,[],Base1,CL).

initDG(N,[],Base,Numargs,Poslitlist,Queue,Base1,CL) :-
        initDG2(N,Queue,Numargs,Poslitlist,Base,Base1,CL).
initDG(N,[(Lhs,Rhs)|Cs],Base,Numargs,Poslitlist,Queue,Base1,CL) :-
        aset(N,Poslitlist,Rhs,Poslitlist1),
        missingliterals(Base,Lhs,NN),
        aset(N,Numargs,NN,Numargs1),
        queuebaseupdate(NN,N,Rhs,Queue,Queue1,Base,Base0),
        clupdate(Lhs,N,CL,CL1),
        N1 is N+1,
        initDG(N1,Cs,Base0,Numargs1,Poslitlist1,Queue1,Base1,CL1).
queuebaseupdate(N,_,_,Queue,Queue,Base,Base) :- N>0.
queuebaseupdate(0,N,Head,Queue,[N|Queue],Base,Base0) :-
        aset(Head,Base,1,Base0).
missingliterals(_,[],0).
missingliterals(Base,[L|Ls],N) :-
        arefl(L,Base,V),
        V \= 1,
        missingliterals(Base,Ls,N0),
```

```
        N is N0+1.
missingliterals(Base,[L|Ls],N) :-
        arefl(L,Base,1), missingliterals(Base,Ls,N).
clupdate([],_,CL,CL).
clupdate([L|Ls],N,CL,CL1) :-
        arefl(L,CL,Clauses),
        aset(L,CL,[N|Clauses],CL0),
        clupdate(Ls,N,CL0,CL1).

initDG2(N,Queue,Numargs,Poslitlist,Base,Base1,CL) :-
        clauselists(N,CL,Poslitlist,Clauselist),
        propagate(Queue,Numargs,Poslitlist,Clauselist,Base,Base1).

clauselists(0,_,_,Cl) :- new_array(Cl).
clauselists(N1,CL,Pll,Clausel) :- N1>0,
        N is N1 - 1,
        clauselists(N,CL,Pll,Clausel0),
        arefl(N,Pll,Lit),
        arefl(Lit,CL,Clauses),
        aset(N,Clausel0,Clauses,Clausel).

propagate([],_,_,_,Base,Base).
propagate([Clause|Queue],Numargs,Poslitlist,Clauselist,Base0,Base1) :-
        arefl(Clause,Clauselist,Clause2s),
        propagate1(Clause2s,Queue,Numargs,Poslitlist,Clauselist,Base0,Base1).
propagate1([Clause2|Cs],Queue,Numargs,Poslitlist,Clauselist,Base0,Base1) :-
        arefl(Clause2,Numargs,N), N1 is N - 1,
        aset(Clause2,Numargs,N1,Numargs1),
        queueupdate(N1,Clause2,Queue,Queue1,Poslitlist,Base0,Base2),
        propagate1(Cs,Queue1,Numargs1,Poslitlist,Clauselist,Base2,Base1).
propagate1([],Queue,Numargs,Poslitlist,Clauselist,Base0,Base1) :-
        propagate(Queue,Numargs,Poslitlist,Clauselist,Base0,Base1).
queueupdate(N,_,Queue,Queue,_,Base,Base) :- N \= 0.
queueupdate(0,Clause2,Queue,[Clause2|Queue],Poslitlist,Base,Base1) :-
        arefl(Clause2,Poslitlist,N),
        aset(N,Base,1,Base1).


%
% The algorithm of Theorem 4.13 for the class SHC (runs in O(n^2) time)
%

shcReduct(Strata0,Reduct) :-
        shcTransform(Strata0,Strata),
        shcReduct([],Strata,Reduct).
shcTransform([],[]).
shcTransform([S|Ss],[TS|TSs]) :-
        shcTransform1(S,TS),
        shcTransform(Ss,TSs).
shcTransform1([],[]).
shcTransform1([sf(A,O,C)|Fs],[(A,OC)|TFs]) :-
        shcObjectivepart(O,C,OC),
        shcTransform1(Fs,TFs).
```

```
shcObjectivepart(O,af(A),(OA,A)) :- variables(O,[],OA).
shcObjectivepart(O,op(-,af(A)),([A|OA],0)) :- variables(O,[],OA).

shcReduct(R,[],R).
shcReduct(Reduct0,[Stratum|Strata],Reduct) :-
        shcReduce(Stratum,Reduct0,Reduct0,Reduct1),
        shcReduct(Reduct1,Strata,Reduct).

% Removing the autoepistemic parts A containing the L operator from
% a single stratum of a stratification

shcReduce([],_,Reduct,Reduct).
shcReduce([(A,O)|Fs],Lower,Reduct0,Reduct) :-
        shcFulfilled(A,Lower,TF),
        shcReduce1(O,TF,Fs,Lower,Reduct0,Reduct).
shcReduce1(O,true,Fs,Lower,Reduct0,Reduct) :-
        shcReduce(Fs,Lower,[O|Reduct0],Reduct).
shcReduce1(_,false,Fs,Lower,Reduct0,Reduct) :-
        shcReduce(Fs,Lower,Reduct0,Reduct).

% Testing whether the formulae inside L in A are L-consequences of
% the (already reduced) lower levels of the stratification

shcFulfilled([],_,true).
shcFulfilled([op('L',E)|Rest],Lower,TF) :-
        shc_l_o_consequence(E,Lower,TF1),
        shcFulfilled1(Rest,Lower,TF1,TF).
shcFulfilled([op(-,op('L',E))|Rest],Lower,TF) :-
        shc_l_o_consequence(E,Lower,TF1),
        opposite(TF1,TF0),
        shcFulfilled1(Rest,Lower,TF0,TF).

shcFulfilled1(_,_,false,false).
shcFulfilled1(Rest,Lower,true,TF) :- shcFulfilled(Rest,Lower,TF).

% L-consequence of an HF formula from a set of Horn clauses
%
% shc_l_o_consequence(Formula,Theory)

shc_l_o_consequence(Formula,Theory,TF) :-
        shcObjectivize(Formula,Theory,Formula2),
        append(Formula2,Theory,Theory1),
        new_array(Tmp),
        dg(Theory1,Tmp,Tmp1),
        arefl(0,Tmp1,TF0),      % Test whether falsum follows
        vqewr(TF0,TF).

vqewr([],false). vqewr(1,true). vqewr(0,false).

% A trick for making the L-consequence testing more efficient is to
% replace the subformulae Lp by the logical constant True (resp. False)
% when p has (resp. has not) been found to be a L-consequence of the
% set of formula. Further details in the thesis.
```

```
shcObjectivize(F,Th,F2) :-
        disjuncts_of(F,[],Ds),
        shcObjectivize0(Ds,Th,F2).
disjuncts_of(co('V',A,B),Ds0,Ds) :-
        disjuncts_of(A,Ds0,Ds1),
        disjuncts_of(B,Ds1,Ds).
disjuncts_of(C,Ds,[C|Ds]) :- (C = co(&,_,_) ; C = af(_) ; C = op(-,_) ).

shcObjectivize0([],_,[]).
shcObjectivize0([D|Ds],Th,HC) :-
        conjuncts_of(D,Dc),
        shcObjectivize1(Dc,Ds,Th,([],0),HC).

shcObjectivize1([],Ds,Th,Clause,[Clause|HC]) :- shcObjectivize0(Ds,Th,HC).
shcObjectivize1([af(A)|Cs],Ds,Th,(Ant,0),HC) :-
        shcObjectivize1(Cs,Ds,Th,(Ant,A),HC).
shcObjectivize1([op(-,af(A))|Cs],Ds,Th,(Ant,Con),HC) :-
        shcObjectivize1(Cs,Ds,Th,([A|Ant],Con),HC).
shcObjectivize1([op('L',F)|Cs],Ds,Th,Clause,HC) :-
        shc_l_o_consequence(F,Th,TF),
        shcObjectivize2(TF,Cs,Ds,Th,Clause,HC).
shcObjectivize1([op(-,op('L',F))|Cs],Ds,Th,Clause,HC) :-
        shc_l_o_consequence(F,Th,TF),
        opposite(TF,TF1),
        shcObjectivize2(TF1,Cs,Ds,Th,Clause,HC).

shcObjectivize2(true,Cs,Ds,Th,Cl,HC) :- shcObjectivize1(Cs,Ds,Th,Cl,HC).
shcObjectivize2(false,_,Ds,Th,_,HC) :- shcObjectivize0(Ds,Th,HC).
```

File `sat.pl`

```
/* May 1992, Jussi Rintanen, Helsinki University of Technology

Predicate "satisfiable" for satisfiability testing. It is based on
the analytic tableau of R. Smullyan, First-order Logic, 1968.

satisfiable(F) iff F is satisfiable. F is a propositional formula.

*/

% Rules of the analytic tableau

analyze(co('V',P,Q),pl(A,B),pl(A,B2)) :- conc(B,beta(P,Q),B2).
analyze(co(&,P,Q),pl(A,B),pl(A2,B)) :- conc(A,alpha(P,Q),A2).
analyze(co(=>,P,Q),pl(A,B),pl(A,B2)) :-
        negate(P,NP), conc(B,beta(NP,Q),B2).
analyze(co(<=>,P,Q),pl(A,B),pl(A,B2)) :-
        negate(P,NP), negate(Q,NQ),
        conc(B,beta(co(&,P,Q),co(&,NP,NQ)),B2).
analyze(op(-,co('V',P,Q)),pl(A,B),pl(A2,B)) :-
```

```
        negate(P,NP), negate(Q,NQ),
        conc(A,alpha(NP,NQ),A2).
analyze(op(-,co(&,P,Q)),pl(A,B),pl(A,B2)) :-
        negate(P,NP), negate(Q,NQ),
        conc(B,beta(NP,NQ),B2).
analyze(op(-,co(=>,P,Q)),pl(A,B),pl(A2,B)) :-
        negate(Q,NQ), conc(A,alpha(P,NQ),A2).
analyze(op(-,co(<=>,P,Q)),pl(A,B),pl(A,B2)) :-
        negate(P,NP), negate(Q,NQ),
        conc(B,beta(co(&,NP,Q),co(&,P,NQ)),B2).
analyze(op(-,op(-,P)),pl(A,B),pl(A2,B)) :-
        conc(A,alpha(P),A2).
analyze(af(_),P,P).
analyze(op(-,af(_)),P,P).
analyze(op('L',_),P,P).
analyze(op(-,op('L',_)),P,P).

negate(op(-,P),P).
negate(P,op(-,P)) :- P\=op(-,_).

% Prioritization

empty([]).
first([A|P],P,A).

next(pl(A,B),pl(A,B),none) :- empty(A), empty(B).
next(pl(A,B),pl(A,B2),F) :- empty(A), first(B,B2,F).
next(pl(A,B),pl(A2,B),F) :- first(A,A2,F).

conc(A1,F,A2) :- F=alpha(_), append(A1,[F],A2).
conc(A1,F,A2) :- F=alpha(F1,F2), append(A1,[alpha(F1),alpha(F2)],A2).
conc(A1,F,A2) :- F=beta(_,_), append(A1,[F],A2).

emptyPlist(pl([],[])).

%

satisfiable(Formula,TF) :-
        emptyPlist(E),
        analyze(Formula,E,Plist),
        construction(Plist,[Formula],TF).

construction(Plist,Branch,TF) :-
        next(Plist,Plist2,Formula),
        sat(Formula,Plist2,Branch,T),
        TF = T.

sat(Formula,Plist,Branch,false) :- notsat(Formula,Plist,Branch), !.
sat(_,_,_,true).

notsat(alpha(A),_,Branch) :-
        negate(A,NA),
        member(NA,Branch).
```

```
notsat(alpha(A),Plist,Branch) :-
        negate(A,NA),
        not(member(NA,Branch)),
        analyze(A,Plist,Plist0),
        next(Plist0,Plist1,Formula),
        notsat(Formula,Plist1,[A|Branch]).

notsat(beta(B1,B2),Plist,Branch) :-
        notsat(alpha(B1),Plist,Branch),
        notsat(alpha(B2),Plist,Branch).

% Testing for logical consequence

logical_consequence(Theory,Formula,TF) :-
        conjunctivize([op(-,Formula)|Theory],F),
        satisfiable(F,TF0), converse(TF0,TF).
converse(true,false).
converse(false,true).

conjunctivize([H],H).
conjunctivize([I,J|T],co('&',I,K)) :- conjunctivize([J|T],K).
```