Super Logic Programs

STEFAN BRASS
University of Giessen
JÜRGEN DIX
The University of Manchester
and
TEODOR C. PRZYMUSINSKI
University of California, Riverside

The Autoepistemic Logic of Knowledge and Belief (AELB) is a powerful nonmonotonic formalism introduced by Teodor Przymusinski in 1994. In this paper, we specialize it to a class of theories called "super logic programs". We argue that these programs form a natural generalization of standard logic programs. In particular, they allow disjunctions and default negation of arbitrary positive objective formulas.

Our main results are two new and important characterizations of the static semantics of these programs, one syntactic, and one model-theoretic. The syntactic fixed point characterization is much simpler than the fixed point construction of the static semantics for arbitrary AELB theories. The model-theoretic characterization via Kripke models allows one to construct finite representations of the inherently infinite static expansions.

Both characterizations can be used as the basis of algorithms for query answering under the static semantics. We describe a *query-answering interpreter* for super programs which we developed based on the model-theoretic characterization and which is available on the web.

Categories and Subject Descriptors: F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*; *Modal logic*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Logic Programming*; *Nonmonotonic reasoning and belief revision*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Modal logic*

General Terms: Theory, Algorithms, Languages

Additional Key Words and Phrases: Non-monotonic reasoning, logics of knowledge and beliefs, semantics of logic programs and deductive databases, disjunctive logic programming, negation, static semantics, well-founded semantics, minimal models

1. INTRODUCTION

The relationship between logic programs and non-monotonic knowledge representation formalisms can be briefly summarized as follows. Any non-monotonic formalism for knowledge representation has to contain some form of *default negation*, whether it is defined as an explicit negation operator or is implicitly present in

Author's Address: S. Brass, Inst. f. Informatik, Univ. Giessen, Arndtstr. 2, D-35392 Giessen, Germany, EMAIL: brass@acm.org.

the form of beliefs, disbeliefs or defaults. Since the main difference between logic programs and classical (monotonic) logic is the presence of default negation, logic programs can be viewed, in this sense, as the simplest non-monotonic extension of classical monotonic logic.

However, standard logic programs suffer from some important limitations. Most importantly, they are unable to handle disjunctive information[1]. Yet, in natural discourse as well as in various programming applications we often use *disjunctive statements*. One particular example of such a situation is *reasoning by cases.* Other obvious examples include:

(1) *Approximate information:* for instance, an age "around 30" can be 28, 29, 30, 31, or 32;
(2) *Legal rules:* the judge always has some freedom for his decision, otherwise he/she would not be needed; so laws cannot have unique models;
(3) *Diagnosis:* only at the end of a fault diagnosis we may know exactly which part of some machine was faulty; but as long as we are searching, different possibilities exist;
(4) *Biological inheritance:* if the parents have blood groups A and 0, the child must also have one of these two blood groups (example from [Lipski 1979]);
(5) *Natural language understanding:* here there are many possibilities for ambiguity and they are represented most naturally by multiple intended models;
(6) *Reasoning about concurrent processes:* since we do not know the exact sequence in which certain operations are performed, again multiple models come into play;
(7) *Conflicts in multiple inheritance:* if we want to keep as much information as possible, we should assume the disjunction of the inherited values, see [Brass and Lipeck 1993].

Disjunctive logic programming was used e.g. in the following projects for real applications:

(1) In the DisLoP project [Aravindan et al. 1997], a logical description of a system together with a particular observation is transformed into a disjunctive program so that an interpreter for disjunctive logic programs can compute the minimal diagnosis [Baumgartner et al. 1997].
(2) The view update problem in deductive and relational databases can also be transformed into a problem of computing minimal models of disjunctive programs [Aravindan and Baumgartner 1997].
(3) Another important application of disjunctive techniques in the database area is to glue together different heterogeneous databases to provide a single unified view to the user. With the ever expanding world wide web technology, millions and millions of data in thousands of different formats are thrown at a user who clearly needs some tools to put together information of interest to him. In

[1]The stable model semantics permits multiple models, which is something similar. But it has to be used with caution, since it can easily become inconsistent. Depending on the application, disjunctions can often be more natural.

[Schäfer and Neugebauer 1997; Thomas 1998] it is shown how disjunctive logic programs can be used as mediators and thus as an important tool to create a powerful query language for accessing the web.

(4) In [Stolzenburg and Thomas 1996; 1998] disjunctive logic programming was used for analyzing rule sets for calculating banking fees. A credit institution sells stocks and shares to its customers and charges their accounts. The fee depends on the value of the transaction, the customer type and various other parameters. All this is formulated in a set of rules in natural language. The problem is to translate this rule set into a formal language and then to analyze its various properties. For example does the rule set allow us to calculate a fee for each business deal? Is that fee unique? A straightforward translation into propositional logic and then using a theorem prover did not work. However, it turned out that using disjunctive logic programs solved the problem [Stolzenburg and Thomas 1996; 1998].

Formalisms promoting disjunctive reasoning are more *expressive* and *natural to use* since they permit direct translation of disjunctive statements from natural language and from informal specifications. Consequently, considerable interest and research effort[2] has been recently given to the problem of finding a suitable extension of the *logic programming paradigm* beyond the class of normal logic programs that would ensure a proper treatment of *disjunctive information*. However, the problem of finding a suitable semantics for disjunctive programs and deductive databases proved to be far more complex than it is in the case of normal, non-disjunctive programs[3].

We believe that in order to demonstrate that a class of programs can be justifiably called an *extension of logic programs* one should be able to argue that:

(1) the proposed *syntax* of such programs resembles the syntax of logic programs but it applies to a significantly broader class of programs, which includes the class of disjunctive logic programs as well as the class of logic programs with strong (or "classical") negation [Gelfond and Lifschitz 1991; Alferes et al. 1998];

(2) the proposed *semantics* of such programs constitutes an intuitively natural extension of one (or more) well-established semantics of normal logic programs;

(3) there exists a reasonably simple procedural mechanism allowing, at least in principle, to compute the semantics[4];

(4) the proposed class of programs and their semantics is a special case of a more general non-monotonic formalism which would clearly link it to other well-established non-monotonic formalisms.

[2]It suffices just to mention several recent workshops on *Extensions of Logic Programming* specifically devoted to this subject ([Dix et al. 1995; 1997; Dyckhoff et al. 1996; Dix et al. 1998]).

[3]The book by Minker et. al. [Lobo et al. 1992] provides a detailed account of the extensive research effort in this area. See also [Dix 1995b; Minker 1993; Przymusinski 1995a; 1995b; Minker 1996].

[4]Observe that while such a mechanism cannot – even in principle – be efficient, due to the inherent NP-completeness of the problem of computing answers just to positive disjunctive programs, it can be efficient when restricted to specific subclasses of programs and queries and it can allow efficient approximation methods for broader classes of programs.

In this paper we propose a specific class of such extended logic programs which will be (immodestly) called *super logic programs* or just *super programs*. We will argue that the class of super programs satisfies all of the above conditions, and, in addition, is sufficiently flexible to allow various application-dependent extensions and modifications. It includes all (monotonic) *propositional theories*, all *disjunctive logic programs* and all *extended logic programs* with strong negation. We also provide a description of an implementation of a *query-answering interpreter* for the class of super programs which is available on the world wide web[5].

The class of super logic programs is closely related to other non-monotonic formalisms. In fact, super logic programs constitute a special case of a yet more expressive non-monotonic formalism called the *Autoepistemic Logic of Knowledge and Beliefs*, *AELB*, introduced earlier in [Przymusinski 1994; 1998]. The logic *AELB* isomorphically includes the well-known non-monotonic formalisms of Moore's *Autoepistemic Logic* and McCarthy's *Circumscription*. Through this embedding, the semantics of super logic programs is clearly linked to other well-established non-monotonic formalisms.

At the same time, as we demonstrate in this paper, in spite of their increased expressiveness, super logic programs still admit natural and simple *query answering mechanisms* which can be easily implemented and tested. We discuss one such existing implementation in this paper. Needless to say, the problem of finding suitable inference mechanisms, capable to model human common-sense reasoning, is one of the major research and implementation problems in Artificial Intelligence.

The paper is organized as follows. In Section 2 we recall the definition and basic properties of *non-monotonic knowledge bases*, and we introduce the class of super logic programs as a special subclass of them. We also establish basic properties of super programs. In the following Sections 3 and 4 we describe two characterizations of the semantics of super programs, one of which is syntactic and the other model-theoretic. Due to the restricted nature of super programs, these characterization are significantly simpler than those applicable to arbitrary non-monotonic knowledge bases. In Section 5 we describe our implementation of a *query-answering interpreter* for super programs which is based on the previously established model-theoretic characterization of their semantics. Section 6 mentions related work. We conclude with some final remarks in Section 7. For the sake of clarity, most proofs are contained in the Appendix.

In the theoretical parts of this paper, we restrict our attention to *propositional* programs. Since we can always consider a propositional instantiation of the program, this does not limit the generality of the obtained results from a semantic standpoint. Of course, many practical applications need rules with variables. The current version of our interpreter for super-logic programs is already supporting variables that satisfy the allowedness/range-restriction condition: Every variable that occurs in a rule must also appear in at least one positive body literal of the rule. Then an intelligent grounding mechanism can be applied, see Section 5.

[5]See `http://www.informatik.uni-giessen.de/staff/brass/slp/`. If this URL should be unavailable, try `http://www.sis.pitt.edu/~sbrass/slp/` and `http://purl.oclc.org/NET/slp/`. It is not necessary to download the interpreter and install it locally, one can simply submit a super logic program and a query via an HTML form. However, a local installation is also possible.

2. SUPER LOGIC PROGRAMS

We first define the notion of a *non-monotonic knowledge base*. Super logic programs are special knowledge bases.

The language is based on the *Autoepistemic Logic of Knowledge and Beliefs*, *AELB*, introduced in [Przymusinski 1994; 1998]. However, in this paper we use the default negation operator *not* instead of the belief operator $\mathcal{B}$. Actually, *not* F can be seen as an abbreviation for $\mathcal{B}(\neg F)$, so this is only a slight syntactic variant[6]. Consequently, all of our results apply to the original AELB as well. First, let us briefly recall the basic definitions of AELB.

2.1 Syntax

Consider a fixed propositional language $\mathcal{L}$ with standard connectives ($\vee$, $\wedge$, $\neg$, $\rightarrow$, $\leftarrow$, $\leftrightarrow$) and the propositional letters *true* and *false*. We denote the set of its propositions by $At_{\mathcal{L}}$. Extend the language $\mathcal{L}$ to a propositional modal language $\mathcal{L}_{not}$ by augmenting it with a modal operator *not* , called the *default negation* operator. The formulae of the form *not* F, where F is an arbitrary formula of $\mathcal{L}_{not}$, are called *default negation atoms* or just *default atoms* and are considered to be atomic formulae in the extended propositional modal language $\mathcal{L}_{not}$. The formulae of the original language $\mathcal{L}$ are called *objective*, and the elements of $At_{\mathcal{L}}$ are called *objective atoms*. Any propositional theory in the modal language $\mathcal{L}_{not}$ will be called a *non-monotonic knowledge base* (or "knowledge base" for short):

Definition 2.1 *(Non-monotonic Knowledge Bases).* By a *non-monotonic knowledge base* we mean an arbitrary (possibly infinite) theory in the propositional language $\mathcal{L}_{not}$. By using standard logical equivalences, the theory can be transformed into a set of clauses of the form:

$$B_1 \wedge \ldots \wedge B_m \wedge not\, G_1 \wedge \ldots \wedge not\, G_n \rightarrow A_1 \vee \ldots \vee A_k \vee not\, F_1 \vee \ldots \vee not\, F_l$$

where $m, n, k, l \geq 0$, A_i's and B_i's are objective atoms and F_i's and G_i's are arbitrary formulae of $\mathcal{L}_{not}$.

By an *affirmative* knowledge base we mean any such theory all of whose clauses satisfy the condition that $k \neq 0$.

By a *rational* knowledge base we mean any such theory all of whose clauses satisfy the condition that $l = 0$.

In other words, *affirmative* knowledge bases are precisely those theories that satisfy the condition that all of their clauses contain at least one *objective* atom in their heads. This means that one cannot derive contradictions, and there is at least one propositional model. *Rational* knowledge bases are those theories that do not contain any default atoms *not* F_i in heads of their clauses. Intuitively, this means that default negation is defined only by the static semantics, and not by the clauses of the program (*not* is a kind of "built-in predicate"). Observe that arbitrarily deep level of *nested default negations* is allowed in the language $\mathcal{L}_{not}$.

[6] *not* F was introduced as an abbreviation for $\mathcal{B}(\neg F)$, and not $\neg\mathcal{B}(F)$, because otherwise super logic programs would not be rational knowledge bases (see below). Then important properties would not hold.

Super logic programs are a subclass of non-monotonic knowledge bases. There are three restrictions:

(1) Only rational knowledge bases are allowed (i.e. no default negation in the head).
(2) Nested default negation is excluded.
(3) Default negation can only be applied to positive formulas, e.g. $not\,(\neg p)$ and $not\,(p \rightarrow q)$ cannot be used in super logic programs.

Definition 2.2 *(Super Logic Programs).* A Super Logic Program (or "super program") is a non-monotonic knowledge base consisting of (possibly infinitely many) super-clauses of the form:

$$F \leftarrow G \wedge not\, H$$

where F, G and H are arbitrary positive objective formulae (i.e. formulae that can be represented as a disjunction of conjunctions of atoms).

In Proposition 2.5, it will be shown that simpler clauses can be equivalently used. However, one does not need a rule/clause form, but any formula can be permitted that is equivalent to such clauses. Our current implementation accepts all formulae that satisfy the following two conditions:

(1) inside *not* only $\vee$, $\wedge$, and objective atoms are used, and
(2) *not* does not appear in positive context:
   —An atom A (objective atom or default negation atom) appears in the propositional formula A in positive context.
   —If A appears in F in positive context, it also appears in $F \wedge G$, $G \wedge F$, $F \vee G$, $G \vee F$, $F \leftarrow G$, $G \rightarrow F$, $F \leftrightarrow G$, $G \leftrightarrow F$ in positive context (where G is any formula). The same holds for "positive" replaced by "negative".
   —If A appears in F in positive context, it appears in $\neg F$, $F \rightarrow G$, $G \leftarrow F$, $F \leftrightarrow G$, $G \leftrightarrow F$ in negative context (where G is any formula). The same holds with "positive" and "negative" exchanged.

Clearly the class of super programs contains all (monotonic) propositional theories and is syntactically significantly broader than the class of normal logic programs. In fact, it is somewhat broader than the class of programs usually referred to as *disjunctive logic programs* because:

(1) It allows constraints, i.e., headless rules. In particular it allows the addition of *strong* negation to such programs, as shown in Section 2.6, by assuming the *strong negation axioms* $\leftarrow A \wedge \text{-}A$, which themselves are program rules (rather than meta-level constraints).
(2) It allows premises of the form $not\, C$, where C is not just an atom but a conjunction of atoms. This proves useful when reasoning with default assumptions which themselves are *rules*, such as $not\,(man \wedge \text{-}human)$, which can be interpreted as stating that we can assume by default that every man is a human (note that $\text{-}human$ represents strong negation of $human$).[7]

[7]It might be possible to avoid the negation of structured formulas by introducing new predicates.

2.2 Implication in the Modal Logic

The semantics of super logic programs can be seen as an instance of the semantics of arbitrary nonmonotonic knowledge bases which we introduce now. AELB assumes the following two simple axiom schemata and one inference rule describing the arguably obvious properties of default atoms[8]:

(CA) Consistency Axiom.

$$not\,(false) \quad \text{and} \quad \neg not\,(true) \tag{1}$$

(DA) Distributive Axiom. For any formulae F and G:

$$not\,(F \vee G) \leftrightarrow not\,F \wedge not\,G \tag{2}$$

(IR) Invariance Inference Rule. For any formulae F and G:

$$\frac{F \leftrightarrow G}{not\,F \leftrightarrow not\,G} \tag{3}$$

The consistency axiom (CA) states that $false$ is assumed to be false by default but *true* is not. The second axiom (DA) states that default negation *not* is distributive with respect to disjunctions. The invariance inference rule (IR) states that if two formulae are known to be equivalent then so are their default negations. In other words, the meaning of $not\,F$ does not depend on the specific form of the formula F, e.g., the formula $not\,(F \wedge \neg F)$ is equivalent to $not\,(false)$ and thus is true by (CA).

We do not assume the distributive axiom for conjunction: $not\,(F \wedge G) \leftrightarrow not\,F \vee not\,G$. This would conflict with the intended meaning of *not* (falsity in all minimal models), see Remark 2.15.

Of course, besides the above axioms and inference rule, also propositional consequences can be used. The simplest way to define this is via propositional models. A (propositional) interpretation of $\mathcal{L}_{not}$ is a mapping

$$\mathcal{I}\colon\ At_{\mathcal{L}} \cup \{not\,(F)\ :\ F \in \mathcal{L}_{not}\} \to \{true, false\},$$

i.e. we simply treat the formulas $not\,(F)$ as new propositions. Therefore, the notion of a model carries over from propositional logic. A formula $F \in \mathcal{L}_{not}$ is a propositional consequence of $T \subseteq \mathcal{L}_{not}$ iff for every interpretation $\mathcal{I}$: $\mathcal{I} \models T \implies \mathcal{I} \models F$. In the examples, we will represent models by sets of literals showing the truth values of only those objective and default atoms which are relevant to our considerations.

Definition 2.3 *(Derivable Formulae).* For any knowledge base T we denote by $Cn_{not}(T)$ the smallest set of formulae of the language $\mathcal{L}_{not}$ which contains T, all (substitution instances of) the axioms (CA) and (DA) and is closed under both propositional consequence and the invariance rule (IR).

We say that a formula F is *derivable* from the knowledge base T if F belongs to $Cn_{not}(T)$. We denote this fact by $T \vdash_{not} F$. A knowledge base T is *consistent* if $Cn_{not}(T)$ is consistent, i.e., if $T \nvdash_{not} false$.

[8]When replacing $not\,(F)$ in these axioms by $\mathcal{B}(\neg F)$, one gets the axioms of AELB as stated in [Brass et al. 1999]. That paper also proves the equivalence to the original axioms of [Przymusinski 1994; 1998].

The following technical lemma follows by transliteration from a lemma stated in [Brass et al. 1999] for AELB:

PROPOSITION 2.4. *For any knowledge base T and any formula F of $\mathcal{L}_{not}$:*

(1) $T \vdash_{not} (not\, F \rightarrow \neg not\, \neg F)$
(2) If $T \vdash_{not} F$ *then* $T \vdash_{not} not\, \neg F$.

Since the operator *not* obeys the *distributive law for disjunction* (DA), the default atom $not\, H$ in the super logic program rules can be replaced by the conjunction $not\, C_1 \wedge \ldots \wedge not\, C_n$ of default atoms $not\, C_i$, where each of the C_i's is a conjunction of objective atoms. Together with standard logical equivalences, this allows us to establish the following useful fact:

PROPOSITION 2.5. *A super logic program P can be equivalently defined as a set of (possibly infinitely many) clauses of the form:*

$$A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n,$$

where A_i's and B_i's are objective atoms and C_i's are conjunctions of objective atoms. If $k \neq 0$, for all clauses of P, then the program is called affirmative.

2.3 Intended Meaning of Default Negation: Minimal Models

As the name indicates, non-monotonic knowledge bases must be equipped with a *non-monotonic semantics*. Intuitively this means that we need to provide a meaning to the default negation atoms $not\, F$. We want the intended meaning of default atoms $not\, F$ to be based on the principle of *predicate minimization* (see [McCarthy 1980; Minker 1982; Gelfond et al. 1989]):

$not\, F$ holds if $\neg F$ is minimally entailed
or, equivalently:
$not\, F$ holds if F is false in all minimal models.

In order to make this intended meaning precise we first have to define what we mean by a minimal model of a knowledge base.

Definition 2.6 *(Minimal Models).* A model M is *smaller* than a model N if it contains the same default atoms but has fewer objective atoms, i.e.

$$\{p \in At_{\mathcal{L}} : M \models p\} \subset \{p \in At_{\mathcal{L}} : N \models p\},$$
$$\{F \in \mathcal{L}_{not} : M \models not\,(F)\} = \{F \in \mathcal{L}_{not} : N \models not\,(F)\}.$$

By a *minimal model* of a knowledge base T we mean a model M of T with the property that there is no smaller model N of T. If a formula F is true in all minimal models of T then we write $T \models_{\min} F$ and say that F is *minimally entailed* by T.

In other words, minimal models are obtained by first assigning *arbitrary* truth values to the default atoms and then *minimizing* the objective atoms (see also [You and Yuan 1993]). For readers familiar with *circumscription*, this means that we are considering circumscription $CIRC(T; At_{\mathcal{L}})$ of the knowledge base T in which objective atoms are minimized while the default atoms $not\, F$ are fixed, i.e., $T \models_{\min} F$ iff $CIRC(T; At_{\mathcal{L}}) \models F$.

Example 2.7. Consider the following simple knowledge base T:

$$\begin{array}{rl} & \rightarrow \ car. \\ car \wedge not\ broken & \rightarrow \ runs. \end{array}$$

Let us prove that T minimally entails $\neg broken$, i.e., $T \models_{min} \neg broken$. Indeed, in order to find minimal models of T we need to assign an *arbitrary* truth value to the only default atom *not broken*, and then *minimize* the objective atoms *broken*, *car* and *runs*. We easily see that T has the following two minimal models (truth values of the remaining belief atoms are irrelevant and are therefore omitted):

$$\begin{array}{rl} M_1 & = \{not\ broken,\ car,\ runs,\ \neg broken\}; \\ M_2 & = \{\neg not\ broken,\ car,\ \neg runs,\ \neg broken\}. \end{array}$$

Since in both of them *car* is true, and *broken* is false, we deduce that $T \models_{min} car$ and $T \models_{min} \neg broken$.

2.4 Static Expansions

The semantics of arbitrary knowledge bases is defined by means of static expansions. The characterizations given in Sections 3 and 4 show that for the subclass of super logic programs, simpler definitions are possible. However, in order to prove the equivalence (and to appreciate the simplication), we first need to repeat the original definition (adepted to use *not* instead of $\mathcal{B}$).

As in Moore's Autoepistemic Logic, the intended meaning of default negation atoms in non-monotonic knowledge bases is enforced by defining suitable expansions of such knowledge bases.

Definition 2.8 *(Static Expansions of Knowledge Bases).* A knowledge base $T^\diamond$ is called a *static expansion* of a knowledge base T if it satisfies the following fixed-point equation:

$$T^\diamond = Cn_{not}\big(T \ \cup \{not\ F : T^\diamond \models_{\min} \neg F\}\big),$$

where F ranges over all formulae of $\mathcal{L}_{not}$.

A static expansion $T^\diamond$ of T must therefore coincide with the knowledge base obtained by: (i) adding to T the default negation $not\ F$ of every formula F that is false in all minimal models of $T^\diamond$, and, (ii) closing the resulting database under the consequence operator Cn_{not}.

As the following proposition shows, the definition of static expansions enforces the intended meaning of default atoms. The implication "$\Leftarrow$" is a direct consequence of the definition, the direction "$\Rightarrow$" holds for rational knowledge bases[9] and thus super logic programs. The proposition is proven in [Przymusinski 1994; 1998]:

Proposition 2.9 (Semantics of Default Negation). *Let $T^\diamond$ be a static expansion of a* rational *knowledge base T. For any formula F of $\mathcal{L}_{not}$ we have:*

$$T^\diamond \models not\ F \quad \textit{iff} \quad T^\diamond \models_{\min} \neg F.$$

[9]In non-rational knowledge bases, the user can explicitly assert default negation atoms $not\ F$, even when F is not minimally entailed.

It turns out that every knowledge base T has the *least* (in the sense of set-theoretic inclusion) static expansion $\overline{T}$ which has an *iterative* definition as the *least fixed point* of the monotonic[10] default closure operator:

$$\Psi_T(S) = Cn_{not}\big(T \cup \{not\, F : S \ \models_{\min} \neg F\}\big),$$

where S is an arbitrary knowledge base and the F's range over all formulae of $\mathcal{L}_{not}$. The following proposition is a transliteration of a theorem from [Przymusinski 1994; 1998; 1995b]. In Section 3 we will show that it is sufficient to consider only formulas F of a special from in $\Psi_T(S)$ when we use it for super logic programs.

Proposition 2.10 (Least Static Expansion). *Every knowledge base T has the* least *static expansion, namely, the least fixed point $\overline{T}$ of the monotonic default closure operator Ψ_T.*

More precisely, the least static expansion $\overline{T}$ of T can be constructed as follows. Let $T^0 = T$ and suppose that T^α has already been defined for any ordinal number $\alpha < \beta$. If $\beta = \alpha + 1$ is a successor ordinal then define:

$$T^{\alpha+1} = \Psi_T(T^\alpha) =_{def} Cn_{not}\big(\ T \cup \{not\, F : T^\alpha \ \models_{\min} \ \neg F\}\ \big),$$

where F ranges over all formulae in $\mathcal{L}_{not}$. Else, if β is a limit ordinal then define $T^\beta = \bigcup_{\alpha<\beta} T^\alpha$. The sequence $\{T^\alpha\}$ is monotonically increasing and has a limit $\overline{T} = T^\lambda = \Psi_T(T^\lambda)$, for some ordinal λ.

We were able to show in [Brass et al. 1999] that for finite knowledge bases, the fixed point is in fact already reached after the first iteration of the default closure operator Ψ_T. In other words, $\overline{T} = T^1$.

Observe that the *least* static expansion $\overline{T}$ of T contains those and only those formulae which are true in *all* static expansions of T. It constitutes the so called *static completion* of a knowledge base T.

Definition 2.11 *(Static Completion and Static Semantics).* The least static expansion $\overline{T}$ of a knowledge database T is called the *static completion* of T. It describes the *static semantics* of a knowledge base T.

It is time now to discuss some examples. For simplicity, unless explicitly needed, when describing static expansions we ignore nested defaults and list only those elements of the expansion that are relevant to our discussion, thus, for example, skipping such members of the expansion as $not\,(F \wedge \neg F)$, $not\,\neg not\,(F \wedge \neg F)$, etc.

Example 2.12. Consider the knowledge base discussed already in Example 2.7:

$$\begin{aligned} & car. \\ car \wedge not\ broken \ &\rightarrow\ runs. \end{aligned}$$

We already established that T minimally entails $\neg broken$. As a result, the static completion $\overline{T}$ of T contains *not broken*. Consequently, as expected, the static completion $\overline{T}$ of T derives *not broken* and *runs* and thus coincides with the perfect model semantics [Przymusinski 1995a] of this stratified program [Apt et al. 1988].

[10] Strictly speaking the operator is only monotone [Przymusinski 1994; 1998] on the lattice of all theories of the form $Cn_{not}(T \cup N)$, where N is a set of default atoms. See also Lemma C.1.

Example 2.13. Consider a slightly more complex knowledge base T:

$$\begin{array}{rl} \mathit{not\ broken} & \rightarrow\ \mathit{runs}. \\ \mathit{not\ fixed} & \rightarrow\ \mathit{broken}. \end{array}$$

In order to iteratively compute its static completion $\overline{T}$ we let $T^0 = T$. One easily checks that $T^0 \models_{min} \neg \mathit{fixed}$. Since

$$T^1 = \Psi_T(T^0) = Cn_{not}\big(T \cup \{\mathit{not}\, F : T^0 \models_{\min} \neg F\}\big),$$

it follows that $\mathit{not\ fixed} \in T^1$ and therefore $\mathit{broken} \in T^1$. Since,

$$T^2 = \Psi_T(T^1) = Cn_{not}\big(T \cup \{\mathit{not}\, F : T^1 \models_{\min} \neg F\}\big),$$

it follows that $\mathit{not}\, \neg \mathit{broken} \in T^2$. Proposition 2.4 implies $\neg \mathit{not\ broken} \in T^2$ and thus $T^2 \models_{\min} \neg \mathit{runs}$. Accordingly, since:

$$T^3 = \Psi_T(T^2) = Cn_{not}\big(T \cup \{\mathit{not}\, F : T^2 \models_{\min} \neg F\}\big),$$

we infer that $\mathit{not\ runs} \in T^3$. As expected, the static completion $\overline{T}$ of T, which contains T^3, asserts that the car is considered not to be fixed and therefore broken and thus is not in a running condition. Again, the resulting semantics coincides with the perfect model semantics of this stratified program.[11]

Example 2.14. Consider the following super program P:

$$\begin{array}{rcl} \mathit{visit\_europe} \vee \mathit{visit\_australia} & \leftarrow & . \\ \mathit{happy} & \leftarrow & \mathit{visit\_europe} \vee \mathit{visit\_australia}. \\ \mathit{bankrupt} & \leftarrow & \mathit{visit\_europe} \wedge \mathit{visit\_australia}. \\ \mathit{prudent} & \leftarrow & \mathit{not}\,(\mathit{visit\_europe} \wedge \mathit{visit\_australia}). \\ \mathit{disappointed} & \leftarrow & \mathit{not}\,(\mathit{visit\_europe} \vee \mathit{visit\_australia}). \end{array}$$

Obviously, the answer to the query $\mathit{visit\_europe} \vee \mathit{visit\_australia}$ must be positive, while the answer to the query $\mathit{visit\_europe} \wedge \mathit{visit\_australia}$ should be negative. As a result, we expect a positive answer to the queries *happy* and *prudent* and a negative answer to the queries *bankrupt* and *disappointed*. This means that default negation interprets disjunctions in an exclusive way, as usual in disjunctive logic programming approaches based on minimal models.

Observe that the query $\mathit{not}\,(\mathit{visit\_europe} \wedge \mathit{visit\_australia})$ intuitively means *"can it be assumed by default that we don't visit both Europe and Australia?"* and thus it is different from the query $\mathit{not\ visit\_europe} \wedge \mathit{not\ visit\_australia}$ which says *"can it be assumed by default that we don't visit Europe and that we don't visit Australia?"*.

It turns out that the static semantics produces precisely the intended meaning discussed above. Indeed, clearly *happy* must belong to the static completion of P. It is easy to check that $\neg \mathit{visit\_europe} \vee \neg \mathit{visit\_australia}$, i.e.

$$\neg(\mathit{visit\_europe} \wedge \mathit{visit\_australia})$$

holds in all minimal models of the program P and therefore

$$\mathit{not}\,(\mathit{visit\_europe} \wedge \mathit{visit\_australia})$$

[11] It follows from our result in [Brass et al. 1999] that *not runs* is in fact already contained in T^1. This is shown by concluding that equivalences similar to Clark's completion are contained in T^1.

must be true in the completion. This proves that *prudent* must hold in the completion. Moreover, since *bankrupt* is false in all minimal models *not bankrupt* must also belong to the completion. Finally, since $visit\_europe \vee visit\_australia$ is true, we infer from Proposition 2.4 that $not\, \neg(visit\_europe \vee visit\_australia)$ and thus also $\neg not\, (visit\_europe \vee visit\_australia)$ holds. But then $\neg disappointed$ is minimally entailed, thus *not disappointed* belongs to the completion.

Remark 2.15. In general, we do not assume the distributive axiom for conjunction:

$$not\, (F \wedge G) \leftrightarrow not\, F \vee not\, G.$$

It is not difficult to prove that for a rational knowledge base T and for any two formulae F and G:

$$T^\diamond \models (not\, F \vee not\, G) \quad \text{iff} \quad (T^\diamond \models not\, F \quad \text{or} \quad T^\diamond \models not\, G).$$

But together with the axiom in question this would mean

$$T^\diamond \models not\, (F \wedge G) \quad \text{iff} \quad (T^\diamond \models not\, F \quad \text{or} \quad T^\diamond \models not\, G).$$

This is too restrictive. For example, given the fact that we can assume by default that we do not drink and drive (i.e., $T^\diamond \models not\, (drink \wedge drive)$) at the same time, we do not necessarily want to conclude that we can either assume by default that we don't drink (i.e., $T^\diamond \models not\, (drink)$) or assume by default that we don't drive (i.e., $T^\diamond \models not\, (drive)$).

In rational knowledge bases, we only assume $not\, (F \wedge G)$ if $F \wedge G$ is false in all minimal models. But it might well be that F is true in some, and G is true in some, but they are never true together. So the axiom in question conflicts with the intended meaning of default negation. On some evenings we drink, on some we drive, but we never do both. If this is all the information we have, $not\, (drink \wedge drive)$ should be implied, but neither $not\, (drink)$ nor $not\, (drive)$ should follow.

Note, however, that the implication in one direction, namely $not\, (F \wedge G) \leftarrow not\, F \vee not\, G$, easily follows from our axioms. In some specific applications, the inclusion of the distributive axiom for conjunction may be justified; in such cases it may simply be added to the above listed axioms.

2.5 Basic Properties of Super Logic Programs

The next theorem summarizes properties of super programs. It is an immediate consequence of results established in [Przymusinski 1994; 1998]:

Theorem 2.16 (Basic Properties of Super Programs).

(*1*) *Let P be any super logic program and let $\overline{P}$ be its static completion. For any formula F, $\overline{P} \models not\, F$ iff $\overline{P} \models_{min} \neg F$.*

(*2*) *Let P be any super logic program and let $\overline{P}$ be its static completion. For any two formulae F and G holds:*

$$\overline{P} \models (not\, F \vee not\, G) \quad \textit{iff} \quad (\overline{P} \models not\, F \quad \textit{or} \quad \overline{P} \models not\, G).$$

(*3*) *Every affirmative super program has a consistent static completion. In particular, this applies to all disjunctive logic programs.*

(4) *For normal logic programs, static semantics coincides with the well-founded semantics [Van Gelder et al. 1991]. More precisely, there is a one-to-one correspondence between consistent static expansions of a normal program and its partial stable models [Przymusinski 1990]. Under this correspondence, (total) stable models [Gelfond and Lifschitz 1988] (also called answer sets) correspond to those consistent static expansions which satisfy the axiom* $not\, A \vee not\, \neg A$*, for every objective atom* A*.*

(5) *For positive disjunctive logic programs, static semantics coincides with the minimal model semantics.*

2.6 Adding Strong Negation to Knowledge Bases

As pointed out in [Gelfond and Lifschitz 1991; Alferes et al. 1998] in addition to default negation, $not\, F$, non-monotonic reasoning requires another type of negation $\text{–}F$ which is similar to classical negation $\neg F$ but does not satisfy the law of the excluded middle (see [Alferes et al. 1998] for more information).

Strong negation $\text{–}F$ can be easily added to super logic programs by: (1) Extending the original objective language $\mathcal{L}$ by adding to it new objective propositional symbols $\text{–}A$, called *strong negation atoms*, for all $A \in At_{\mathcal{L}}$. (2) Adding to the super logic program the following *strong negation clause*, for any strong negation atom $\text{–}A$ that appears in it: $A \wedge \text{–}A \rightarrow false$, which says that A and $\text{–}A$ cannot be both true. Since the addition of strong negation clauses simply results in a new super logic program, our the results apply as well to programs with strong negation.

3. FIXED POINT CHARACTERIZATION OF STATIC COMPLETIONS

Due to Proposition 2.5, we can consider the language of super logic programs to be restricted to the subset $\mathcal{L}^*_{not} \subseteq \mathcal{L}_{not}$, which is the propositional logic over the following set of atoms:

$$At_{\mathcal{L}} \cup \{not\, E : E \text{ is a conjunction of objective atoms from } At_{\mathcal{L}}\}.$$

In particular, the language $\mathcal{L}^*_{not}$ does not allow any nesting of default negations: Default negation can be only applied to conjunctions of objective atoms. As a special case, we allow also the "empty conjunction", so $not\,(true)$ is contained in $\mathcal{L}^*_{not}$. The conjunction E can be treated as a set — because of the invariance inference rule $not\, E_1$ and $not\, E_2$ must have the same truth value if E_1 and E_2 differ only in the sequence or multiplicity of the atoms in the conjunction. In this way, if $At_{\mathcal{L}}$ is finite, only a finite number of default negation atoms must be considered.

In order to answer queries about a super program P, we only need to know which formulae of the restricted language $\mathcal{L}^*_{not}$ belong to the static completion of P. In other words, we only need to compute the restriction $\overline{P}|\mathcal{L}^*_{not}$ of the static completion $\overline{P}$ to the language $\mathcal{L}^*_{not}$. It would be nice and computationally a lot more feasible to be able to compute this restriction without having to first compute the full completion, which involves arbitrarily deeply nested default negations and thus is inherently infinite even for finite programs. The following result provides a positive solution to this problem in the form of a much simplified syntactic fixed point characterization of static completions. In the next section we provide yet another solution in the form of a model-theoretic characterization of static completions.

THEOREM 3.1 (SYNTACTIC FIXED-POINT CHARACTERIZATION). *The restriction of the static completion $\overline{P}$ of a finite super program P to the language $\mathcal{L}^*_{not}$ can be constructed as follows. Let $P^0 = P$ and suppose that P^n has already been defined for some natural number n. Define P^{n+1} as follows:*

$$Cn\big(\ P \cup \{not\, E_1 \wedge \ldots \wedge not\, E_m \rightarrow not\, E_0 : \\ P^n \ \models_{\min} \neg E_1 \wedge \ldots \wedge \neg E_m \rightarrow \neg E_0\}\ \big),$$

where E_i's range over all conjunctions of objective atoms and Cn denotes the standard propositional consequence operator. We allow the special case that E_0 is the empty conjunction (i.e., true) and identify not true with false.

The sequence $\{P^n\}$ is monotonically increasing and has a limit $P^{n_0} = P^{n_0+1}$, for some natural number n_0. Moreover, $P^{n_0} = \overline{P}|\mathcal{L}^*_{not}$, i.e., P^{n_0} is the restriction of the static completion of P to the language $\mathcal{L}^*_{not}$.

PROOF. Contained in the Appendix C. □

The above theorem represents a considerable simplification over the original characterization of static completions given in Proposition 2.10. First of all, instead of using the modal consequence operator Cn_{not}, only the standard propositional consequence operator Cn is used. Moreover, instead of ranging over the set of all (arbitrarily deeply nested) formulae of the language $\mathcal{L}_{not}$ it involves only conjunctions of objective atoms. These two simplifications greatly enhance the implementability of static semantics of finite programs. On the other hand, due to the restriction to the language $\mathcal{L}^*_{not}$, we cannot expect the fixed point to be reached in just one step, as would be the case with the more powerful operator $\Psi_T(S)$ [Brass et al. 1999].

4. MODEL-THEORETIC CHARACTERIZATION OF STATIC COMPLETIONS

In this section, we complement Theorem 3.1 by providing a *model-theoretic* characterization of static completions of finite super programs. More precisely, we characterize models of static completions restricted to the narrower language $\mathcal{L}^*_{not}$. The resulting characterization was directly used in a prototype implementation of a query answering interpreter for static semantics described in the next section.

Definition 4.1 *(Reduced Interpretations).*

(1) Let $\mathcal{OBJ}$ be the set of all propositional valuations of the objective atoms $At_{\mathcal{L}}$.
(2) Let $\mathcal{NOT}$ be the set of all propositional valuations of the default negation atoms $\{not\,(p_1 \wedge \cdots \wedge p_n) : p_i \in At_{\mathcal{L}}\}$ that interpret $not\,(true)$ as false[12].
(3) We call an interpretation I of the language $\mathcal{L}^*_{not}$, i.e., a valuation of $At_{\mathcal{L}} \cup \{not\,(p_1 \wedge \cdots \wedge p_n) : p_i \in At_{\mathcal{L}}\}$, a reduced interpretation and write it as as $I = I_{obj} \cup I_{not}$ with $I_{obj} \in \mathcal{OBJ}$ and $I_{not} \in \mathcal{NOT}$.
(4) In order to emphasize the difference, we sometimes call an interpretation $\mathcal{I}$ of the complete language $\mathcal{L}_{not}$, i.e., a valuation for $At_{\mathcal{L}} \cup \{not\,(F) : F \in \mathcal{L}_{not}\}$, a full interpretation.
(5) Given a full interpretation $\mathcal{I}$, we call its restriction $I = I_{obj} \cup I_{not}$ to the language $\mathcal{L}^*_{not}$ the reduct of $\mathcal{I}$.

[12]Throughout this section, whenever we mention a sequence $p_1, \ldots, p_n$, we allow n to be 0 as well. The empty conjunction is *true*.

Since a reduced interpretation $I = I_{obj} \cup I_{not}$ assigns truth values to all atoms occurring in a super program, the "is model of" relation between such interpretations and super programs is well defined. We call $I = I_{obj} \cup I_{not}$ a minimal model of a super program P if there is no interpretation $I_{obj} \in \mathcal{O}BJ$ with

$$\{p \in At_{\mathcal{L}} \,:\, I'_{obj} \models p\} \;\subset\; \{p \in At_{\mathcal{L}} \,:\, I_{obj} \models p\}$$

such that $I' = I'_{obj} \cup I_{not}$ is also a model of P. This is completely compatible with the corresponding notion for full interpretations.

Our goal is to characterize the reducts of the full models of the static completion $\overline{P}$ of a finite super program P. In fact, once we have the possible interpretations of the default atoms, finding the objective parts of the reducts is easy. The idea how to compute such possible interpretations is closely related to Kripke structures (which is not surprising for a modal logic, for more details see Appendix A). The worlds in such a Kripke structure are marked with interpretations of the objective atoms $At_{\mathcal{L}}$, and the formula $not\,(p_1 \wedge \cdots \wedge p_n)$ is true in a world w iff $\neg(p_1 \wedge \cdots \wedge p_n)$ is true in all worlds w' which can be "seen" from w. Due to the consistency axiom, every world w must see at least one world w'. The static semantics ensures that every world w sees only worlds marked with minimal models. So given a set $\mathcal{O}$ of (objective parts of) minimal models, an interpretation I_{not} of the default atoms is possible iff there is some subset $\mathcal{O}' \subseteq \mathcal{O}$ (namely the interpretations in the worlds w') such that $I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n)$ iff $I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n)$ for all $I_{obj} \in \mathcal{O}'$. Conversely, $\mathcal{O}$ depends on the possible interpretations of the default atoms (since a minimal model contains an objective part and a default part). So when we restrict the possible interpretations for the default negation atoms, we also get less minimal models. This results in a fixed-point computation, formalized by the following operators:

Definition 4.2 *(Possible Interpretations of Default Atoms).* Let P be a super program.

(1) The operator $\Omega_P \colon 2^{\mathcal{N}OT} \to 2^{\mathcal{O}BJ}$ yields objective parts of minimal models given possible interpretations of the default atoms. For every $\mathcal{N} \subseteq \mathcal{N}OT$, let

$$\Omega_P(\mathcal{N}) := \big\{ I_{obj} \in \mathcal{O}BJ \,:\, \text{there is } I_{not} \in \mathcal{N} \text{ such that } I = I_{obj} \cup I_{not} \text{ is minimal model of } P \big\}.$$

(2) Let $\mathcal{O} \subseteq \mathcal{O}BJ$ and $I_{not} \in \mathcal{N}OT$. We call I_{not} given by $\mathcal{O}$ iff for every $p_1, \ldots, p_n \in At_{\mathcal{L}}$:

$$I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n) \iff \text{for every } I_{obj} \in \mathcal{O} \colon\; I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n).$$

(3) The operator $\Pi_P \colon 2^{\mathcal{O}BJ} \to 2^{\mathcal{N}OT}$ yields possible interpretations of the default atoms, given objective parts of minimal models. For every $\mathcal{O} \subseteq \mathcal{O}BJ$, let

$$\Pi_P(\mathcal{O}) := \big\{ I_{not} \in \mathcal{N}OT \,:\, \text{there is a non-empty } \mathcal{O}' \subseteq \mathcal{O} \text{ such that } I_{not} \text{ is given by } \mathcal{O}' \text{ and there is } I_{obj} \text{ such that } I = I_{obj} \cup I_{not} \text{ is a model of } P. \big\}.$$

(4) The operator $\Theta_P \colon 2^{\mathcal{N}OT} \to 2^{\mathcal{N}OT}$ is the composition $\Theta_P := \Pi_P \circ \Omega_P$ of Ω_P and Π_P.

Note that the additional constraint in the definition of Π_P (that it must be possible to extend I_{not} to a model of P) is trivially satisfied for affirmative programs, i.e. programs having at least one head literal in every rule.

Now we are ready to state the main result of this section.

THEOREM 4.3 (MODEL-THEORETIC CHARACTERIZATION). *Let P be a finite super program:*

(1) *The operator Θ_P is monotone (in the lattice of subsets of $\mathcal{NOT}$ with the order $\supseteq$) and thus its iteration beginning from the set $\mathcal{NOT}$ (all interpretations of the default atoms, the bottom element of this lattice) has a fixed point $\mathcal{N}^\diamond$.*

(2) *$\mathcal{N}^\diamond$ consists exactly of all $I_{not} \in \mathcal{NOT}$ that are default parts of a full model $\mathcal{I}$ of $\overline{P}$.*

(3) *A reduced interpretation $I_{obj} \cup I_{not}$ is a reduct of a full model $\mathcal{I}$ of $\overline{P}$ iff $I_{not} \in \mathcal{N}^\diamond$ and $I_{obj} \cup I_{not} \models P$.*

PROOF. Contained in the Appendix B. □

Example 4.4. Let us consider the following logic program P:

$$\begin{aligned} p \vee q &\leftarrow \mathit{not}\ r. \\ q &\leftarrow \mathit{not}\ q. \\ r &\leftarrow q. \end{aligned}$$

Until the last step of the iteration, it suffices to consider only those default atoms which occur in P and thus have influence on the objective atoms. But for the sake of demonstration, we start with the set $\mathcal{NOT}$ containing all eight possible valuations of the three default atoms *not p*, *not q*, and *not r* (however, we do not consider conjunctive default atoms here). These default interpretations can be extended to the minimal reduced models listed in the following table. Note that models numbered 6 to 10 are equal to models numbered 1 to 5, except that *not p* is true in them (this shows that *not p* is really superfluous at this stage).

No.	*not p*	*not q*	*not r*	*p*	*q*	*r*
1.	false	false	false	false	false	false
2.	false	false	true	true	false	false
3.	false	false	true	false	true	true
4.	false	true	false	false	true	true
5.	false	true	true	false	true	true
6.	true	false	false	false	false	false
7.	true	false	true	true	false	false
8.	true	false	true	false	true	true
9.	true	true	false	false	true	true
10.	true	true	true	false	true	true

For instance, when *not q* and *not r* are both interpreted as false, p, q and r are false in a minimal model. If *not q* is interpreted as true, q and r are true and p is false by minimality. If *not q* is false and *not r* is true, there are two possibilities for a minimal model: either p can be true, or q and r together. So there are only three

possible valuations for the objective parts of minimal models, i.e. $\Omega_P(\mathcal{N}OT)$ is:

p	q	r
false	false	false
true	false	false
false	true	true

Note that although there is a minimal model in which p, q, and r are all false, there can be other minimal models based on different interpretations of the default atoms. Now the above three interpretations are the input for the next application of Π_P. Of course, from every minimal objective interpretation we immediately get a possible default interpretation if we translate the truth of p to the falsity of *not* p and so on. But we can also combine minimal objective parts conjunctively and let *not* p be true only if $\neg p$ is true in all elements of some set of minimal models. Thus, $\Pi_P\big(\Omega_P(\mathcal{N}OT)\big)$ is:

not p	*not* q	*not* r
true	true	true
false	true	true
true	false	false
false	false	false

This means that only the models numbered $1, 5, 6, 10$ remain possible given the current knowledge about the defaults. Their objective parts are:

p	q	r
false	false	false
false	true	true

Finally, p is false in all of these models, so *not* p must be assumed, and only the following two valuations of the default atoms are possible in the fixed point $\mathcal{N}^\diamond$:

not p	*not* q	*not* r
true	true	true
true	false	false

The reduced models of the least static expansion $\overline{P}$ consist therefore of all reduced interpretations that include one of these two default parts and are models of P itself.

Obviously, computing the reduct of a full interpretation is easy. Conversely, in the Appendix, Definition B.1, we define a Kripke structure which allows us to extend the reduced interpretations to full interpretations, which are models of the static completion. However, let us observe that not all full models of the least static expansion can be reconstructed in this way. Instead, we obtain only one representative from every equivalence class with the same reduct. For example, one can easily verify that the program $P := \{p \leftarrow \mathit{not}\, p\}$ has infinitely many different full models (see Example A.6 in the Appendix).

5. IMPLEMENTATION OF THE MODEL-THEORETIC CHARACTERIZATION

We implemented a query-answering interpreter for super logic programs under the static semantics which based on the above model-theoretic characterization[13]. The interpreter has a web interface, so it is not necessary to install it locally in order to try it.

5.1 Parsing and Clause Transformation

As mentioned before, the interpreter does not require clause form. One can use the following logical operators:

Name	Notation	Alt.	Priority	Associativity
Classical Negation ($\neg$)	`~`		1	right assiciative
Default Negation (*not*)	`not`		1	cannot be nested
Conjunction (and, $\wedge$)	`&`	`,`	2	right associative
Disjunction (or, $\vee$)	`|`	`; v`	3	right associative
Implication (then, $\rightarrow$)	`->`		4	not associative
Implication (if, $\leftarrow$)	`<-`	`:-`	4	not associative
Equivalence (if and only if, $\leftrightarrow$)	`<->`		4	not associative

Default negation can only be used in negative context, and inside default negation, only disjunction and conjunction are permitted. As in Prolog, atoms are sequences of letters, digits, and underscores that start with a lowercase letter. Alternatively, one can use any sequence of characters enclosed in single quotes. Every formula must end in a full stop ".". For instance, the last line of Example 2.14 would be entered as:

```
disappointed <- not(visit_europe | visit_australia).
```

The first task of the program is to read the input, check it for syntactical correctness, and translate it into clauses as shown in Proposition 2.5. This is done with standard techniques. For instance, the above input formula is internally represented as

```
disappointed <- not(visit_europe), not(visit_australia).
```

5.2 Intelligent Grounding

As mentioned above, the current implementation permits rules with variables, but every variable must appear in a positive body literal ("allowedness", "range-restriction"). For instance, the rule

```
p(X,a) | p(X,b) :- q(1,X), not r(X).
```

is syntactically valid, since `X` appears in `q(1,X)`. This check is done after the clause transformation, so "positive body literal" is not literally required in the input. As in Prolog, variables start with an uppercase letter, and the anonymous variable "_" is also understood. Predicate arguments can be atoms, integers, or variables.

[13]See `http://www.informatik.uni-giessen.de/staff/brass/slp/`. Actually, there are two implementations: A prototype written in Prolog (1325 lines, 608 lines of code), and a version with web user interface and better performance written in C++ (21809 lines, 10201 lines of code). The source code of both versions is freely available. The C++ version is still being further developed.

Function symbols (term/list constructors) are not permitted in order to ensure that the ground instantiation is finite.

However, our implementation does not compute the complete ground instantiation of the input program, since this will often be prohibitively large. Instead, the set of derivable conditional facts is computed:

Definition 5.1 *(Conditional Fact).* Conditional facts are formulae of the form

$$A_1 \vee \ldots \vee A_k \leftarrow not\, C_1 \wedge \ldots \wedge not\, C_n,$$

where the A_i are objective atoms (positive ground literals) the C_i are conjunctions of objective atoms (positive ground literals), $k \geq 0$, and $n \geq 0$. In the following, we treat head and body of conditional facts as sets of literals, and write them as $\mathcal{A} \leftarrow \mathcal{C}$ with $\mathcal{A} = \{A_1, \ldots, A_k\}$ and $\mathcal{C} = \{not\, C_1, \ldots, not\, C_n\}$.

So conditional facts are rules without positive body literals, and because of the allowedness condition, they also cannot contain variables. Conditional facts were introduced in [Bry 1989; 1990; Dung and Kanchanasut 1989].

The derivation is done with the hyperresolution operator, which is a generalization of the standard T_P-operator. In the non-disjunctive case, the T_P-operator applies a rule $A \leftarrow B_1 \wedge \ldots \wedge B_m$ by finding facts $B'_1, \ldots, B'_m$ that match the body literals, i.e. $B_i\sigma = B'_i$ for a ground substitution σ, and then deriving the corresponding instance $A\sigma$ of the head literal. This is applied e.g. for the bottom-up evaluation in deductive databases. The generalization for conditional facts is as follows:

Definition 5.2 *(Hyperresolution Operator).* Let P be a set of super logic program rules as in Proposition 2.5 and $\mathcal{F}$ be a set of conditional facts. Then $H_P(\mathcal{F})$ is the set of conditional facts

$$\{A_1\sigma, \ldots, A_k\sigma\} \cup (\mathcal{A}_1 \setminus \{B_1\sigma\}) \cup \ldots \cup (\mathcal{A}_m \setminus \{B_m\sigma\}) \leftarrow \{not\, C_1\sigma, \ldots, not\, C_n\sigma\} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_m$$

such that there is a rule

$$A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n$$

in P, conditional facts $\mathcal{A}_i \leftarrow \mathcal{C}_i$, $i = 1, \ldots, m$ in $\mathcal{F}$, and a subsitution σ with $B_i\sigma \in \mathcal{A}_i$.

In other words, the positive body literals are matched with any literal in the head[14] of a conditional fact, and then the remaining head literals are added to the corresponding instance of the rule head, as the conditions are added to the corresponding instance of the negative body literals. For example, if the rule

$$p_1(X) \vee p_2(Y) \leftarrow q_1(a, X) \wedge q_2(X, Y) \wedge not\ r(Y)$$

is matched with $\underline{q_1(a,a)} \vee s(b)$ and $t(a) \vee \underline{q_2(a,b)} \leftarrow not\ u(c)$, the derived conditional fact is

$$p_1(a) \vee p_2(b) \vee s(b) \vee t(a) \leftarrow not\ r(b) \wedge not\ u(c).$$

[14] Actually, one can make the resolvable literal unique, see [Brass 1996]. However, this is not yet done in the current implementation.

Again, head and body of conditional facts are treated as sets of literals, so duplicate literals are removed. Also for the hyperresolution step itself, the rule is instantiated only as far as there are matching conditional facts.

The hyperresolution operator is applied iteratively until no new conditional facts can be derived. Since there are only finitely many possible ground literals[15] and no duplicates are permitted in head or body, the process is guranteed to terminate. The importance of the hyperresolution fixpoint is that it has the same minimal models as the original program:

THEOREM 5.3. *Let P be a super logic program, $\mathcal{F}_0 := \emptyset$, and $\mathcal{F}_{i+1} := H_P(\mathcal{F}_i)$, and n be a natural number such that $\mathcal{F}_{n+1} = \mathcal{F}_n$. Then the following holds for all Herbrand interpretations I: I is a minimal model of the ground instantiation P^* of P if and only if I is a minimal model of $\mathcal{F}_n$.*

PROOF. See Appendix D. □

Of course, for the termination it is important that the implementation discovers if the same conditional fact was derived again. However, also non-minimal conditional facts can be eliminated: A conditional fact $\mathcal{A} \leftarrow \mathcal{C}$ is non-minimal if there is another conditional fact $\mathcal{A}' \leftarrow \mathcal{C}'$ with $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{C}' \subseteq \mathcal{C}$, where at least one inclusion is proper. Since in this case the non-minimal conditional fact $\mathcal{A} \leftarrow \mathcal{C}$ is logically implied by the stronger conditional fact $\mathcal{A}' \leftarrow \mathcal{C}'$, the set of models is not changed if $\mathcal{A} \leftarrow \mathcal{C}$ is deleted.

5.3 Algorithm for Detection for Duplicate and Non-Minimal Conditional Facts

Since this redundancy test is needed often, it is essential that it is implemented efficiently. E.g. [Seipel 1994] noted that the elimination of duplicate and subsumed conditional facts took more than half of the total running time of query evaluation.

The subsumption check used in our implementation when a newly derived conditional fact $\mathcal{A} \leftarrow \mathcal{C}$ is inserted into the set $\mathcal{F}$ runs in the time

$$O(\text{number of occurrences of literals from } \mathcal{A} \leftarrow \mathcal{C} \text{ in } \mathcal{F}).$$

We use the algorithm from [Brass and Dix 1995] that is shown in Figure 1.

The basic idea is as follows: We store for each existing conditional fact an overlap counter. When a new conditional fact $\mathcal{A} \leftarrow \mathcal{C}$ is produced, we access for each literal in $\mathcal{A}$ and $\mathcal{C}$ all conditional facts $\mathcal{A}' \leftarrow \mathcal{C}'$ that contain the same literal and increment the overlap counter of these conditional facts. If the overlap counter (i.e. the number of common literals) reaches the length (number of literals) of $\mathcal{A}' \leftarrow \mathcal{C}'$, then $\mathcal{A} \leftarrow \mathcal{C}$ is redundant. Otherwise, if it is equal to the length of $\mathcal{A} \leftarrow \mathcal{C}$, then $\mathcal{A}' \leftarrow \mathcal{C}'$ is redundant.

It would be inefficient if we had to set the overlap counters of all existing conditional facts back to 0 before each redundancy test. Therefore, we do a "lazy initialization": Each test is assigned a unique number, and we store for each conditional fact the value of the counter at the last time the conditional fact was accessed.

[15] As mentioned before, no function symbols are permitted. Also, the conjunctions inside default negation are treated as sets of objective atoms: The implementation discovers e.g. that $not\,(p \wedge q)$ and $not\,(q \wedge p \wedge q)$ are really the same.

```
testno := 0;
F := ∅;

procedure insert(A ← C)
begin
    if A = ∅ and C = ∅ then
        report inconsistency and exit;
    testno = testno + 1;
    for each A ∈ A do
        for each A' ← C' ∈ F with A ∈ A' do
            if inc_overlap(A' ← C', |A| + |C|) then
                return; /* A ← C is non-minimal */
    for each C ∈ C do
        for each A' ← C' ∈ F with C ∈ C' do
            if inc_overlap(A' ← C', |A| + |C|) then
                return; /* A ← C is non-minimal */
    /* A ← C is minimal, insert it:  */
    F := F ∪ {A ← C};
    length[A ← C] := |A| + |C|;
    lastset[A ← C] := 0;
end;

function inc_overlap(A' ← C', l):  bool
begin
    /* Increment or initialize the overlap counter:  */
    if lastset[A' ← C'] = testno then
        overlap[A' ← C'] := overlap[A' ← C'] + 1;
    else
        overlap[A' ← C'] := 1;
        lastset[A' ← C'] := testno;
    /* Check for subsumption:  */
    if overlap[A' ← C'] = length[A' ← C'] then
        return true; /* A ← C is non-minimal */
    if overlap[A' ← C'] = l then
        F := F \ {A' ← C'}; /* A' ← C' is non-minimal */
    return false;
end;
```

Fig. 1. Algorithm for Detecting Duplicate and Non-Minimal Conditional Facts

In order to reduce duplicates, our implementation also does a seminaive evaluation by requiring that at least one body literal is matched with a conditional fact that was newly derived in the last hyperresolution round. Although it is still possible that the same conditional fact is derived more than once, it is at least never produced twice in the same way.

5.4 Residual Program

In the end, when we apply the model-theoretic characterization for a program that contains n different default negation literals, we need to consider 2^n possible interpretations of these literals and compute minimal models for them. That is obviously very expensive and is only possible for relatively small n.

Therefore, our implementation evaluates the simple cases directly before it starts the expensive algorithm on the remaining difficult cases. This is done by means

of reduction operators defined and analyzed in [Brass and Dix 1999]). Positive reduction means that *not p* can be evaluated to true if p does not occur in any head of a conditional fact:

Definition 5.4 *(Positive Reduction).* A set $\mathcal{F}_1$ of conditional facts is transformed by positive reduction into a set $\mathcal{F}_2$ of conditional facts if there is an atom p and a conditional fact $\mathcal{A} \leftarrow \mathcal{C} \in \mathcal{F}$ such that p does not occur in any head in $\mathcal{F}$ and

$$\mathcal{F}_2 = \big(\mathcal{F}_1 \setminus \{\mathcal{A} \leftarrow \mathcal{C}\}\big) \cup \{\mathcal{A} \leftarrow \mathcal{C}'\},$$

where $\mathcal{C}' = \mathcal{C} \setminus \{not\ p\}$.

Negative reduction means that $not\ p_1 \wedge \ldots \wedge not\ p_k$ can be evaluated to false if there is an unconditional fact $p_1 \vee \ldots \vee p_k \leftarrow true$:

Definition 5.5 *(Negative Reduction).* A set $\mathcal{F}_1$ of conditional facts is transformed by negative reduction into a set $\mathcal{F}_2$ of conditional facts if there is $p_1 \vee \ldots \vee p_k \leftarrow true$ in $\mathcal{F}_1$ and $\mathcal{F}_2 = \mathcal{F}_1 \setminus \{\mathcal{A} \leftarrow \mathcal{C}\}$ where $\{not\ p_1, \ldots, not\ p_k\} \subseteq \mathcal{C}$.

For instance, in Example 2.14 there is an unconditional fact

$$visit\_europe \vee visit\_australia.$$

As mentioned above, the last rule in that example is internally represented as

$$disappointed \leftarrow not\,(visit\_europe) \wedge not\,(visit\_australia).$$

It can be deleted, because its condition can never be true. In the example, this also eliminates two default negation atoms, which means that fewer interpretations must be considered in the next step of the algorithm.

The current version of our implementation does not evaluate default negation literals that contain conjunctions of atoms. Since the reduction step is only an optimization, this is possible: negations of conjunctions remain for the general algorithm. But we of course plan to strengthen these optimizations in future versions. The more default negations can be eliminated with relatively cheap transformations, the less important it becomes that the algorithm for the general case is expensive.

Applying negative reduction can make positive reduction applicable, and vice versa. The two transformations are applied as long as possible, and the result is called the residual program. It is uniquely determined, the exact sequence of applications of the two transformations is not important. The reduction in the size of the program and the number of distinct default negations can be significant. For instance, no default negations remain if the input program is stratified and non-disjunctive.

Negative reduction can be implemented with a method that is very similar to the overlap counting shown above for the elimination of nonminimal clauses. Positive reduction is implemented by managing for each objective atom a counter for the number of occurrences in conditional fact heads, as well as a list of atoms for which this counter is 0.

Note that positive and negative reduction do change the set of minimal models, since even if there is the fact p, the interpretation of *not p* in a minimal model is arbitrary. Only the static semantics ensures that *not p* is false in this case. The

following simple cumulation theorem ensures that positive and negative reduction as well as other such optimizations do not change the static semantics:

THEOREM 5.6 (INVARIANCE OF STATIC SEMANTICS).

(1) *Let T be a knowledge base with $T \models_{\min} \neg F$. Then T and $T \cup \{not\, F\}$ have the same static expansions.*

(2) *Let T_1 and T_2 be knowledge bases with $Cn_{not}(T_1) = Cn_{not}(T_2)$. Then T_1 and T_2 have the same static expansions.*

For instance, consider the program $\{p \leftarrow not\, q\}$. Since q is false in all minimal models, we can add $not\, q$ by (1) without changing the static semantics. Both formulas together are logically equivalent to $\{p, not\, q\}$. By (2), this does not change the static semantics. Finally, since also $\{p\}$ alone implies minimally $\neg q$, by (1) we get that $\{p\}$ has the same static semantics as $\{p, not\, q\}$. This proves that the given program can indeed be reduced to $\{p\}$ (an example of positive reduction).

For an example of negative reduction, consider the program T_1:

$$p \vee q.$$
$$s \leftarrow not\, p \wedge not\, q \wedge not\, r.$$

Negative reduction transforms this program into $T_2 = \{p \vee q\}$. In order to show that this does not change the static semantics, we apply part (2) of the theorem. We must show that T_2 is Cn_{not}-equivalent to T_1: Proposition 2.4 tells us that $not\,(\neg(p \vee q))$ is contained in $Cn_{not}(T_2)$, and then we also get that $\neg not\,(p \vee q) \in Cn_{not}(T_2)$ (by Proposition 2.4, the invariance inference rule (IR) and propositional consequences). By applying the distributive axiom (DA) $\neg(not\, p \wedge not\, q)$ follows. But now the rule $s \leftarrow not\, p \wedge not\, q \wedge not\, r$ is a propositional consequence, and therefore also contained in $Cn_{not}(T_2)$.

COROLLARY 5.7. *Positive and negative reduction do not change the static semantics of a program.*

5.5 Application of the Model-Theoretic Characterization

We apply the definition of the operators Ω_P and Π_P from Section 4 quite literally. Of course, we use not the input program P, but the residual program $\mathcal{F}$.

As mentioned before, it suffices to consider only those default negation atoms that occur explicitly in the residual program. We call these the "critical" default negation atoms. Other default atoms have no influence on the minimal models and we can always extend a valuation of the critical default atoms consistently to a valuation of all default atoms. The objective atoms that occur inside the default negations in the residual program are called the critical objective atoms.

The algorithm shown in Figure 2 computes the static interpretations for the critical default negation atoms, i.e. the fixed point $\mathcal{N}^\diamond$ of $\Theta_\mathcal{F}$ (reduced to critical default negation atoms). The fixpoint computation starts with the set of all possible interpretations for the critical default negation atoms. This is the most expensive part of the algorithm, so real improvements must attack this problem. In order to keep at least the memory complexity in reasonable limits the set $\mathcal{N}$ is materialized only after the first filtering step (application of $\Theta_\mathcal{F}$). Without this optimization,

```
Let CritNeg be the set of all default negation atoms that appear in F;
O := ∅; /* Objective parts of minimal models, filled by procedure modgen */

function static(F): Set of static interpretations for CritNeg
begin
    for each interpretation I_not of CritNeg do
        D := {A : A ← C ∈ F, I_not ⊨ C};
        modgen(D); /* Inserts minimal models of D into O */
    N := ∅;
    for each interpretation I_not of CritNeg do
        if possible(I_not, O) then
            if there is no false ← C ∈ F with I_not ⊨ C then
                N := N ∪ {I_not};
    Changed := true;
    while Changed do
        Changed := false;
        O := ∅;
        for each I_not ∈ N do
            D := {A : A ← C ∈ F, I_not ⊨ C};
            modgen(D);
        for each I_not ∈ N do
            if not possible(I_not, O) then
                N := N \ {I_not};
                Changed := true;
    return N;
end;

function possible(I_not, O): bool
begin
    I_∩ := CritNeg; /* Empty intersection: All default negations are true */
    NotEmpty := false;
    for each I_obj ∈ O do
        /* Interpretations are identified with the set of true default atoms */
        I'_not := {not (p_1 ∧ ... ∧ p_n) ∈ CritNeg : I_obj ⊨ ¬(p_1 ∧ ... ∧ p_n)};
        if I_not ⊆ I'_not then
            I_∩ := I_∩ ∩ I'_not;
            NotEmpty := true;
    return I_not = I_∩ and NotEmpty;
end;
```

Fig. 2. Application of the Model-Theoretic Characterization

the first half of the procedure `static(`$\mathcal{F}$`)` could simply be replaced by assigning $\mathcal{N}$ all possible interpretations.

Now the fixpoint computation starts (the **while**-loop in Figure 2). In each step, first the set $\mathcal{O} = \Omega_{\mathcal{F}}(\mathcal{N})$ of minimal models, reduced to the objective atoms is computed. The default negation part of each such minimal model must be an element of $\mathcal{N}$.

Our implementation computes $\mathcal{O} = \Omega_{\mathcal{F}}(\mathcal{N})$ as follows: For each default negation interpretation in $\mathcal{N}$, we evaluate the conditions of the conditional facts in the residual program and get a set $\mathcal{D}$ of positive disjunctions. For this, any minimal model generator can be used. We use the algorithm explained in the next subsection. A nice feature of it is that it computes not interpretations for all objective atoms, but only for the critical ones, i.e. those objective atoms that appear inside

default negations in the residual program. Minimal model generators proposed in the literature are, e.g., [Bry and Yahya 1996; Niemelä and Simons 1996; Niemelä 1996].

Once we have computed $\mathcal{O}$, interpretations are eliminated from $\mathcal{N}$ that do not satisfy the condition of $\Pi_{\mathcal{F}}(\mathcal{O})$. However, it would be very inefficient to consider all subsets $\mathcal{O}' \subseteq \mathcal{O}$, as required in Definition 4.2. However, in order to check whether a given interpretation I_{not} of the default atoms is selected by $\Theta_{\mathcal{F}}$, only the maximal $\mathcal{O}'$ is of interest. It is the set of all $I_{obj} \in \mathcal{O}$ that satisfy, for every default atom, the condition:

$$\text{if } I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n), \text{ then } I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n).$$

Obviously, the inclusion of other I_{obj} into $\mathcal{O}'$ would immediately destroy the required property, namely that I_{not} is the "intersection" of all models in $\mathcal{O}'$:

$$I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n) \iff \text{for all } I_{obj} \in \mathcal{O}\text{: } I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n).$$

Of course, it must also be tested that the set $\mathcal{O}'$ is non-empty. In the procedure `possible` shown in Figure 2, the set $\mathcal{O}'$ is not explicitly constructed, but instead the interpretation $I_\cap$ is constructed as the intersection of all elements of $\mathcal{O}'$.

5.6 Computation of Minimal Models

In order to compute $\Omega_{\mathcal{F}}(\mathcal{N})$, we must compute the objective parts of minimal models, of which the default negation part is one of the interpretations in $\mathcal{N}$. As explained above, we do this by looping over the interpretations $I_{not} \in \mathcal{N}$, evaluating the conditions $\mathcal{C}$ of the formulas $\mathcal{A} \leftarrow \mathcal{C} \ \in \ \mathcal{F}$ in I_{not}, and computing minimal models of the resulting positive disjunctions of objective atoms[16]. The algorithm that is used to compute minimal models of these disjunctions is shown in Figure 3. We only compute the truth values of critical objective atoms, i.e. objective atoms that appear inside default negations in $\mathcal{F}$[17]. Of course, it is important that this partial interpretation can be extended to a total minimal model.

So let p be a critical objective atom that is not yet assigned a truth value. The two simple cases are: (1) p does not appear in any disjunction in $\mathcal{D}$: Then p must be false in every minimal model. (2) p appears as a (non-disjunctive) fact in $\mathcal{D}$: Then p must be true in all minimal models.

The difficult case is when p appears in one or more proper disjunctions. Then it can be true or false, and we backtrack over the different possibilities (the backtracking is simulated with local variables and "call by value" parameters in Figure 3, in the actual implementation, several stacks are used). The model construction is based on the following theorem:

THEOREM 5.8. *Let $\mathcal{D}$ be a set of disjunctions of objective atoms, which does not contain the empty disjunction false, and which does not contain two disjunctions $\mathcal{A}$*

[16]One "last minute" idea was to compute directly minimal models of $\mathcal{F}$, which could be done with an algorithm similar to the one shown in Figure 3. This might relieve us from the need to consider exponentially many interpretations, but depending on the given rules, it might backtrack over even more different cases. Improvements in this direction are subject of future research.

[17]The procedure `modgen` shown in Figure 3 can assign "false" to additional (non-critical) objective atoms, if this is required as a reason for a critical atom being true.

```
procedure modgen(𝒟)
begin
    I := ∅; /* Completely undefined interpretation */
    modgen_rec(𝒟, I);
end;

procedure modgen_rec(𝒟, I)
begin
    if all critical objective atoms have a truth value in I then
        𝒪 := 𝒪 ∪ {I};
    else
        select a critical objective atom p that is still undefined in I;
        eliminate non-minimal disjunctions from 𝒟;
        if p does not appear in a disjunction in 𝒟 then
            /* p is surely false */
            modgen_rec(𝒟, I ∪ {¬p});
        else if p appears as a non-disjunctive fact in 𝒟 then
            /* p is surely true */
            modgen_rec(𝒟, I ∪ {p});
        else
            /* p can be false or true */
            𝒟' := {A \ {p} : A ∈ 𝒟}; /* Remove p from all disjunctions */
            modgen_rec(𝒟', I ∪ {¬p});
            for each disjunction p_1 ∨ ... ∨ p_{i-1} ∨ p ∨ p_{i+1} ∨ ... ∨ p_n ∈ 𝒟 do
                I' := I ∪ {p, ¬p_1, ..., ¬p_{i-1}, ¬p_{i+1}, ..., ¬p_n};
                𝒟' := {A \ {p_1, ..., p_{i-1}, p_{i+1}, ..., p_n} : A ∈ 𝒟} ∪ {p};
                modgen_rec(𝒟', I');
end;
```

Fig. 3. Minimal Model Generator

and $\mathcal{A}'$, such that $\mathcal{A} \subset \mathcal{A}'$ (i.e. $\mathcal{A}'$ is non-minimal). Let I be a partial interpretation such that atoms interpreted as false do not appear in $\mathcal{D}$ and atoms interpreted as true appear as facts in $\mathcal{D}$. Let p be an atom that appears in the proper disjunction

$$p_1 \vee \cdots \vee p_{i-1} \vee p \vee p_{i+1} \vee \cdots \vee p_n$$

(and possibly more such disjunctions). Then

(1) There is a minimal model of $\mathcal{D}$ that extends I and interprets p as false.

(2) There is a minimal model of $\mathcal{D}$ that extends I and interprets p as true and $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ as false.

Proof. See Appendix D. □

First, p can be false. In order to ensure that the theorem is again applicable for the next atom, we eliminate p from all disjunctions in $\mathcal{D}$ (which also might eliminate further disjunctions that now become non-minimal). Alternatively, p can be true. However, p can only be true in a minimal model if for one of the disjunctions in which p appears, all the remaining atoms are false. This ensures that there is a reason why p must be true. Again, the assigned truth values are reflected by changing the set of disjunctions: p is added as a fact and $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ are removed from all disjunctions. This might make disjunctions in $\mathcal{D}$ non-minimal, they are removed with the same overlap counting technique as shown in Figure 1.

In this way, assumed truth values are immediately propagated in the disjunctions, which often makes truth values of further critical atoms unique.

The algorithm can also be understood as applying a generalization of Clark's completion to disjunctions: One treats the disjunction $p_1 \vee \cdots \vee p_n$ like the n rules

$$p_i \leftarrow \neg p_1 \wedge \cdots \wedge \neg p_{i-1} \wedge \neg p_{i+1} \wedge \cdots \wedge \neg p_n$$

and then applies the standard completion (which basically translates $\leftarrow$ to $\leftrightarrow$). Whereas in general, Clark's completion does not correspond to minimal models, in this particular case, the completion enforces exactly the minimal models [Brass et al. 1999]. For instance, suppose that p appears in the disjunctions $p \vee q$ and $p \vee r \vee s$. Then the completion contains the formula

$$p \leftrightarrow \neg q \vee (\neg r \wedge \neg s).$$

The above model generation algorithm distinguishes now three cases:

(1) p is false. Then it is removed from the disjunctions, so q and $r \vee s$ are assumed.
(2) p is true and q is false.
(3) p is true and r and s are both false.

In summary, nice features of the algorithm are:

(1) It never runs into dead ends: All assumed truth values are indeed possible.
(2) No additional minimality test for the generated models is needed.
(3) It can generate partial minimal models, i.e. truth value assignments for any subset of the atoms that can be extended to minimal models.
(4) The space complexity is polynomial in the size of the input disjunctions.

On the negative side, the algorithm may generate the same model several times.

5.7 Query Evaluation

Queries are written in the form

```
? p(X), q(X,b), not r(X).
```

They are internally handled by adding a rule with the special predicate `$answer` in the head:

```
$answer(X) <- p(X), q(X,b), not r(X).
```

It is also possible to specify the answer variables explicitly, e.g.

```
? X: q(X,Y).
```

Then only values for `X` will be printed:

```
$answer(X) <- q(X,Y).
```

With these additional rules, the hyperresolution fixpoint is computed as usual. When positive reduction is applied, and when minimal models are generated, conditional facts that contain `$answer` in the head are ignored. Conditional facts in the residual program with only `$answer` literals in the head correspond to possible answers. Their conditions are evaluated in each of the models in $\mathcal{N}^\diamond$. If the condition is true in all $I_{not} \in \mathcal{N}^\diamond$, the corresponding answer is printed.

Note that disjunctive answers are possible. E.g. consider the program

```
p(a) | p(b).
```

Then the answer for the query `? p(X)` will be `X=a | X=b`. This means that the disjunction of the corresponding instances of the query follows from the static semantics of the program.

The implementation can also print the fixed point $\mathcal{N}^\diamond$ of $\Theta_\mathcal{F}$, i.e. all static interpretations of the default negation literals. Normally, only the truth values of the critical negations are shown, but one can add further default negation literals if one wants to see their truth values. This is done by adding a command of the form

```
! not p(a).
```

6. RELATED WORK

6.1 Extended Logic Programs by Lloyd and Topor

The first attempt to generalize the syntax of logic programs is due to Lloyd and Topor [Lloyd and Topor 1984; 1985; 1986]. They allow arbitrary formulae in bodies of clauses, which is especially important in order to map database queries and view definitions into logic programming rules. However, the head of a rule must still be a single atom. Furthermore, their semantics differentiates between an atom occurring in the head and its negation occurring in the body, whereas our semantics is invariant under logical equivalences.

Lloyd and Topor use Clark's completion for defining the meaning of default negation, whereas the static semantics extends the well-founded semantics. We have not yet treated quantifiers in the formulae of knowledge bases, but all other normalization transformations of Lloyd and Topor work as well for our semantics. This means that if Lloyd and Topor had used the WFS instead of Clark's completion, they could have come up with a special case of the static semantics (restricted to formulae of the form $A \leftarrow W$).

6.2 Disjunctive Stable Semantics

The disjunctive stable semantics (answer sets) is the most popular semantics proposed earlier for disjunctive programs [Przymusinski 1990; Gelfond and Lifschitz 1991]. Its main problem is that it is inconsistent for some very intuitive programs, such as:

$$\begin{aligned} work &\leftarrow not\, tired. \\ sleep &\leftarrow not\, work. \\ tired &\leftarrow not\, sleep. \\ angry &\leftarrow not\, paid, work. \\ paid &\leftarrow . \end{aligned}$$

Here we should at least be able to conclude *paid* and *not angry*, and the static semantics gives us just that. In general, the static semantics is always consistent for disjunctive logic programs.

In addition to the fact that stable semantics is contradictory for some very simple and natural programs, it also suffers from a number of *structural* problems. In

particular, the stable semantics is not *relevant*, i.e., answering a query does not depend only on the call graph below that query ([Brass and Dix 1997; Dix 1995a]), and it is not *compositional* (or modular), a highly desirable property for software engineering and KR [Bry 1996; Bugliesi et al. 1994; Teusink and Etalle 1996]. The static semantics has both of these nice structural properties. Thus methods for query optimization based on the dependency graph can be applied and may reduce the overall computation.

Needless to say, we do not claim that the static semantics should replace the disjunctive stable semantics. Like it is the case with normal logic programs, there are some application domains for which the disjunctive stable semantics seems to be better suited to represent their intended meaning [Cholewiński et al. 1995]. However, we do claim that the static semantics is a very natural and well-behaved extension of the well-founded semantics to disjunctive programs and beyond. Because of the importance of the well-founded semantics for normal programs, it is of great importance to find the its proper extension(s) to more general theories.

6.3 Well-Founded Circumscriptive Semantics

Our work is also related to the approach presented in [You and Yuan 1993] where the authors defined the *well-founded circumscriptive* semantics for disjunctive programs. They introduced the concept of minimal model entailment with fixed interpretation of the default atoms and defined the semantics of T as the minimal models of T which satisfy the limit of the following sequence: $W^0 = \emptyset$ and

$$W^{n+1} := W^n \cup \{\neg not\, p \,:\, T \cup W^n \models p\} \cup \{not\, p \,:\, T \cup W^n \models_{\min} \neg p\}$$

(this is a translation into our own notation). In the definition of the static fixpoint operator Ψ_T (see Proposition 2.10), negations of default negations are not directly assumed. But in contrast to the well-founded circumscriptive semantics which only assumes the default negation of propositions, the static semantics assumes the default negation of arbitrary propositions. With Proposition 2.4 it follows that if $T \cup \{not\, F \;:\; T^n \models_{\min} \neg F\}$ implies F, then also $\neg not\, F$ is contained in $T^{n+1} = \Psi_T(T^n)$. With this, it is easy to see that $W^n \subseteq T^n$. So the default negation part of the well-founded circumscriptive semantics is weaker than that of the static semantics. The reason is that the static semantics assumes negations of arbitrary formulae, not only of atoms. E.g. in Theorem 3.1 it is really needed that also implications between default negation atoms are assumed. For instance, in Example 4.4, the static semantics first derives $not\, q \leftrightarrow not\, r$ and then $not\, p$ follows. However, in the well-founded circumscriptive semantics the limit of the sequence W^n is the empty set: Nothing becomes known about the default negation literals.

Another difference between the two approaches is that the well-founded circumscriptive semantics permits only minimal models of the objective atoms. The static semantics considers minimal models when deciding which default negations to assume, but if one does not use default negation, one gets standard propositional logic. For instance, let $P = \{p\}$. If also q belongs the language, the well-founded circumscriptive semantics implies $\neg q$ and $not\, q$, whereas the static semantics implies $not\, q$, but not $\neg q$. Of course, both semantics imply p and $\neg not\, p$.

6.4 Disjunctive Logic Programming Systems

In addition to the DisLoP project in Koblenz, there is a similar project on disjunctive logic programming at the Theoretical University of Vienna called `dlv` (see [Eiter et al. 1997; 1998]). While DisLoP concentrated on a disjunctive extension of the well-founded semantics (D-WFS), `dlv` computes stable models (answer sets) both for the disjuctive and the non-disjunctive case. It is a knowledge representation system, which offers front-ends to several advanced KR formalisms. The kernel language, which extends disjunctive logic programming by true negation and integrity constraints, allows for representing complex knowledge base problems in a highly declarative fashion [Eiter et al. 1998]. The project also incorporates modular evaluation techniques along with linguistic extensions to deal with quantitative information [Buccafurri et al. 1997].

6.5 Nested Rules

Recently, Lifschitz et. al. introduced nested expressions in the heads and bodies of rules ([Lifschitz et al. 1998]). Also nested negation as failure is supported, which is excluded in super logic programs. Their semantics is based on answer sets (also called stable models), whereas in our framework, we build upon the well-founded semantics.

In [Greco et al. 1998] nested rules are also allowed in rule heads. The authors show that, in terms of expressive power, they can capture the full second level of the polynomial hierarchy (which is also true for other approaches).

6.6 Other Logics

There are also successful approaches to generalize logic programming languages by using intuitionistic, linear, or higher order logics (e.g. [Hodas and Miller 1994; Nadathur and Miller 1990]). These extensions seem somewhat orthogonal to our treatment of negation in the context of arbitrary propositional formulas.

7. CONCLUSION

We introduced the class of super programs as a subclass of the class of all non-monotonic knowledge bases. We showed that this class of programs properly extends the classes of disjunctive logic programs, logic programs with strong (or "classical") negation and arbitrary propositional theories. We demonstrated that the semantics of super programs constitutes an intuitively natural extension of the semantics of normal logic programs. When restricted to normal logic programs, it coincides with the well-founded semantics, and, more generally, it naturally corresponds to the class of all partial stable models of a normal program.

Subsequently, we established two characterizations of the static semantics of finite super programs, one of which is syntactic and the other model-theoretic, which turned out to lead to procedural mechanisms allowing its computation. Due to the restricted nature of super programs, these characterizations are significantly simpler than those applicable to arbitrary non-monotonic knowledge bases.

We used one of these characterizations as a basis for the implementation of a *query-answering interpreter* for super programs which is available on the WWW. We noted that while no such computational mechanism can be efficient, due to the

inherent NP-hardness of the problem of computing answers to just positive disjunctive programs, they can become efficient when restricted to specific subclasses of programs and queries. Moreover, further research may produce more efficient *approximation methods*.

The class of non-monotonic knowledge bases, and, in particular, the class of super programs, constitutes a special case of a much more expressive non-monotonic formalism called the *Autoepistemic Logic of Knowledge and Beliefs*, *AELB*, introduced earlier in [Przymusinski 1994; 1998]. *AELB* isomorphically includes the well-known non-monotonic formalisms of Moore's *Autoepistemic Logic* and McCarthy's *Circumscription*. Via this embedding, the semantics of super programs is clearly linked to other well-established non-monotonic formalisms.

The proposed semantic framework for super programs is sufficiently flexible to allow various application-dependent extensions and modifications. We have already seen in Theorem 2.16 that by assuming an additional axiom we can produce the stable semantics instead of the well-founded semantics. By adding the distributive axiom for conjunction we can obtain a semantics that extends the *disjunctive stationary semantics* of logic programs introduced in [Przymusinski 1995a]. Many other modifications and extensions are possible including variations of the notion of a minimal model resulting in *inclusive*, instead of *exclusive*, interpretation of disjunctions [Przymusinski 1995b].

APPENDIX

A. KRIPKE MODELS OF STATIC EXPANSIONS

Before we prove the model-theoretic characterization (Theorem 4.3), we prove here a Theorem which easily allows us to construct models of the least static expansion from Kripke structures. It is used as a lemma in the proof to Theorem 4.3, but it is of its own interest. Although we later need only super programs and reduced models, we allow in this section arbitrary belief theories and consider full models.

In order to be precise and self-contained, let us briefly repeat the definition of Kripke structures [Marek and Truszczyński 1993]. Since our axioms entail the normality axiom, it suffices to consider normal Kripke structures:

Definition A.1 *(Kripke Structure).* A (normal) Kripke structure is a triple $\mathcal{K} = (W, R, V)$ consisting of

(1) a non-empty set W, the elements $w \in W$ are called worlds,
(2) a relation $R \subseteq W \times W$, the "visibility relation" (if $R(w, w')$ we say that world w sees world w'), and
(3) a mapping $V\colon W \rightarrow \mathcal{O}BJ$, which assigns to every world w a valuation $I_{obj} = V(w)$ of the objective atoms $At_{\mathcal{L}}$.

Definition A.2 *(Truth of Formulas in Worlds).* The validity of a formula F in a world w given Kripke structure $\mathcal{K} = (W, R, V)$ is defined by

(1) If F is an objective atom (proposition) $p \in At_{\mathcal{L}}$, then $(\mathcal{K}, w) \models F \ :\Longleftrightarrow\ V(w) \models p$.
(2) If F is a negation $\neg G$, then $(\mathcal{K}, w) \models F \ :\Longleftrightarrow\ (\mathcal{K}, w) \not\models G$.

(3) If F is a disjunction $G_1 \vee G_2$, then

$$(\mathcal{K}, w) \models F \ :\Longleftrightarrow\ (\mathcal{K}, w) \models G_1 \text{ or } (\mathcal{K}, w) \models G_2$$

(and further propositional connectives as usual).

(4) If F is a default negation atom $not\,(G)$, then

$$(\mathcal{K}, w) \models F \ :\Longleftrightarrow\ \text{for all } w' \in W \text{ with } R(w, w'){:}\ (\mathcal{K}, w') \not\models G.$$

Now given such a Kripke structure $\mathcal{K}$, we get from every world w a propositional interpretation $\mathcal{I} = \mathcal{K}(w)$ of $\mathcal{L}_{not}$, i.e. a valuation of $At_{\mathcal{L}} \cup \{not\,(F) : F \in \mathcal{L}_{not}\}$: We simply make an objective or belief atom A true in $\mathcal{I}$ iff $(\mathcal{K}, w) \models A$. Since the propositional connectives are defined in a Kripke structure like in standard propositional logic, we obviously have $\mathcal{I} \models F \iff (\mathcal{K}, w) \models F$ for all $F \in \mathcal{L}_{not}$.

Theorem A.3 (Kripke Structures Yield Static Expansions). *Let T be an arbitrary belief theory and $\mathcal{K} = (W, R, V)$ be a Kripke structure satisfying*

(1) For every $w \in W$, there is a $w' \in W$ with $R(w, w')$ (consistency).

(2) For every $w \in W$, $(\mathcal{K}, w) \models T$, i.e. the interpretation $\mathcal{I} = \mathcal{K}(w)$ is a model of T.

(3) For every $w, w' \in W$ with $R(w, w')$, the interpretation $\mathcal{I} = \mathcal{K}(w')$ is a minimal model of T ("only minimal models are seen").

Then $T^{\diamond} := \{F \in \mathcal{L}_{not} : \text{for every } w \in W{:}\ (\mathcal{K}, w) \models F\}$ is a static expansion of T.

Proof. We have to show that $T^{\diamond} = Cn_{not}\left(T \cup \{not\,F : T^{\diamond} \models_{\min} \neg F\}\right)$. The fact that $T \subset T^{\diamond}$ follows immediately from the second assumption. First we prove that $T^{\diamond}$ is closed under Cn_{not}:

(1) Consistency Axioms: Let any $w \in W$ be given. The first part $(K, w) \models not\,(false)$ is trivial, because for any $w' \in W$ we have $(\mathcal{K}, w') \not\models false$. Second, we have to show $(\mathcal{K}, w) \models \neg not\,(true)$. By the first requirement of the theorem, there is $w' \in W$ with $R(w, w')$. Now $(\mathcal{K}, w') \models true$, therefore $(\mathcal{K}, w) \not\models not\,(true)$, i.e. $(\mathcal{K}, w) \models \neg not\,(true)$.

(2) Distributive Axiom: We have to show that for all formulas $F, G \in \mathcal{L}_{not}$ and all $w \in W$ that the distributive axiom holds in w: $(\mathcal{K}, w) \models not\,(F \vee G) \leftrightarrow Not(F) \wedge not\,(G)$. This follows simply from applying the definitions: $(\mathcal{K}, w) \models not\,(F \vee G) \iff$ for all $w' \in W$ with $R(w, w')$: $(\mathcal{K}, w') \not\models F \vee G \iff$ for all $w' \in W$ with $R(w, w')$: $(\mathcal{K}, w') \not\models F$ and $(\mathcal{K}, w') \not\models G \iff (\mathcal{K}, w) \models not\,(F)$ and $(K, w) \models not\,(G), \iff (\mathcal{K}, w) \models not\,(F) \wedge not\,(G)$.

(3) Invariance Inference Rule: Let $F \leftrightarrow G \in T^{\diamond}$, i.e. for every $w \in W$ we have $(\mathcal{K}, w) \models F \iff (\mathcal{K}, w) \models G$. But then also $(\mathcal{K}, w) \models not\,(F \leftrightarrow G)$ holds for every $w \in W$ since $(K, w') \not\models F \iff (\mathcal{K}, w') \not\models G$ holds for every $w' \in W$ with $R(w, w')$.

(4) Closure under propositional consequences: Let $F \in \mathcal{L}_{not}$ a formula which is a propositional consequence of $F_1, \ldots, F_n \in T^{\diamond}$. Now $F_i \in T^{\diamond}$ means that $(\mathcal{K}, w) \models F_i$ for every $w \in W$. But the formulas valid in one world are closed under propositional consequences, since the meaning of the propositional connectives is defined as in the standard case. So we get $(\mathcal{K}, w) \models F$, and thus $F \in T^{\diamond}$.

Now suppose that $T^\diamond \models_{\min} \neg F$, i.e. F is false in all minimal models of $T^\diamond$. We have to show that $not\,(F) \in T^\diamond$, i.e. $(\mathcal{K}, w) \models not\,(F)$ for every $w \in W$. So we have to show $(\mathcal{K}, w') \not\models F$ for every $w' \in W$ with $R(w, w')$.

Let such a w' be given, and let $\mathcal{I} = \mathcal{K}(w')$. By the last condition of the theorem, we know that $\mathcal{I}$ is a minimal model of T. By construction, it is also a model of $T^\diamond$, and since $T \subseteq T^\diamond$, there can be no smaller model. Thus, $\mathcal{I}$ is a minimal model of $T^\diamond$ and therefore satisfies $\neg F$, i.e. $(\mathcal{K}, w') \not\models F$.

Thus we have shown that $Cn_{not}\big(T \cup \{not\,F : T^\diamond \models_{\min} \neg F\}\big) \subseteq T^\diamond$. The converse follows from assumption (3) and (4) of the preceding definition. □

This theorem gives us a simple way to construct models of the least static expansion $\overline{T}$: Obviously, for every $w \in W$, the interpretation $\mathcal{I} = \mathcal{K}(w)$ is a model of $T^\diamond$. But the least static expansion is a subset of every other static expansion, i.e. $\overline{T} \subseteq T^\diamond$, and therefore we have $\mathcal{I} \models \overline{T}$.

Corollary A.4. *If a Kripke structure $\mathcal{K} = (W, R, V)$ satisfies the conditions of Theorem A.3, then for every $w \in W$, the interpretation $\mathcal{I} = \mathcal{K}(w)$ is a model of the least static expansion $\overline{T}$.*

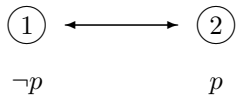
Example A.5. Let us consider the knowledge base of Example 2.13:

$$\begin{array}{ll} not\,broken & \rightarrow\ runs. \\ not\,fixed & \rightarrow\ broken. \end{array}$$

Here, we can construct a Kripke model $\mathcal{K}$ with only one world w with the valuation $I_{obj} = \{\neg fixed,\ broken,\ \neg runs\}$. We let this world "see" itself, i.e. $R := \big\{(w, w)\big\}$. Then *not fixed* and *not runs* are true in $(\mathcal{K}, w)$, but *not broken* is false.

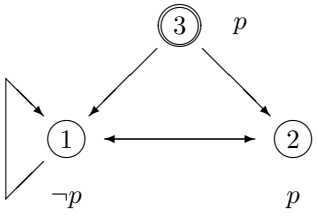
It is, however, also possible to add a world w' with a non-minimal valuation $I'_{obj} = \{fixed,\ broken,\ runs\}$. Both worlds can only see w, i.e. $R := \big\{(w, w),\ (w', w)\big\}$, because all seen worlds must be minimal models. This example shows that the static semantics does not imply $\neg fixed$, but it of course implies *not fixed*. So the non-monotonic negation is cleanly separated from the classical negation.

Example A.6. Let us consider the theory $P := \{p \leftarrow not\,(p)\}$ corresponding to a well-known logic program. We claimed in Section 4 that the least static expansion $\overline{P}$ of this program has infinitely many different models. But let us first look a Kripke structure which generates only two models:



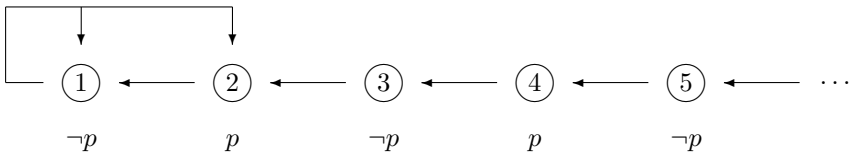
I.e. the set of worlds W is $\{1, 2\}$, the visibility relation R is $\big\{(1, 2),\ (2, 1)\big\}$ (each world can see the other one, but not itself), and p is false in world 1 and true in world 2. In world 1, $not\,(p)$ is false, and in world 2, it is true.

But there are also quite different Kripke models. The construction in Section 4 will yield:



Note that in world 3, the default atom $not\,(p)$ is false, so the interpretation of this world is non-minimal (there is no need to make p true). Thus, there can be no incoming edges.

Let us now finally present the Kripke structure which yields infinitely many models:



Now we of course have to explain that different worlds really yield different interpretations. The trick is that world 1 is the only world in which neither $not\,(p)$ nor $not\,(\neg p)$ are true. Now other worlds can be identified with the number of nested beliefs necessary to get to world 1. So the formula

$$\mathcal{B}^{n-1}\big(\neg not\,(p) \wedge \neg not\,(\neg p)\big) \wedge \mathcal{B}^{n-2}\big(not\,(p) \vee not\,(\neg p)\big)$$

is true in world $n \geq 2$, but in no other world. Each world gives rise to one full model (as explained in Theorem A.3 and Corollary A.4), so the theory $\{p \leftarrow not\,(p)\}$ really has infinitely many full models.

B. PROOF OF THE MODEL-THEORETIC CHARACTERIZATION (THEOREM 4.3)

We will first prove that a reduced model $I = I_{obj} \cup I_{not}$ with $I \models P$ and $I_{not} \in \mathcal{N}^{\diamond}$ can be extended to a full model $\mathcal{I}$ of the least static expansion $\overline{P}$. Of course, this proof is based on the Kripke structure already mentioned in Section 4:

Definition B.1 *(Standard Kripke Model).* Let $P \subseteq \mathcal{L}^{*}_{not}$ be a super program. We call the Kripke structure $\mathcal{K} = (W, R, V)$ defined as follows the "standard Kripke model" of P. Let $\mathcal{N}^{\diamond}$ be the greatest fixpoint of Θ_P. Then:

(1) The set of worlds W are the reduced interpretations $I = I_{obj} \cup I_{not}$ we are interested in, i.e. satisfying $I \models P$ and $I_{not} \in \mathcal{N}^{\diamond}$.

(2) $R(I, I')$, i.e. I sees I' iff I' is a minimal model of P and for all $p_1, \ldots, p_n \in At_{\mathcal{L}}$:

$$I \models not\,(p_1 \wedge \cdots \wedge p_n) \implies I' \models \neg p_1 \vee \cdots \vee \neg p_n.$$

(3) The valuation $V(I)$ of a reduced interpretation $I = I_{obj} \cup I_{not}$ is the objective part I_{obj}.

As before, we denote by $\mathcal{K}(I)$ the full interpretation satisfying the default atoms true in world I, i.e.

$$\mathcal{K}(I) \models not\,(F) \iff \text{for all } I' \in W \text{ with } R(I,I')\text{: } (\mathcal{K}, I') \models \neg F.$$

Now the reduced interpretation $I = I_{obj} \cup I_{not}$ assigns a truth value to the atoms $not\,(p_1 \wedge \cdots \wedge p_n)$ and the Kripke structure also assigns a truth value to them. But the construction guarantees that the truth values always agree:

LEMMA B.2. *For $I = (I_{obj} \cup I_{not}) \in W$ and all $p_1, \ldots, p_n \in At_{\mathcal{L}}$:*

$$\mathcal{K}(I) \models not\,(p_1 \wedge \cdots \wedge p_n) \iff I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n).$$

PROOF. First we show the direction $\Longleftarrow$: Let $I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n)$. By the construction, $I' \models \neg p_1 \vee \cdots \vee \neg p_n$ holds for all worlds I' seen by I, i.e. satisfying $R(I, I')$. Thus, $\mathcal{K}(I) \models not\,(p_1 \wedge \cdots \wedge p_n)$.

Now we prove $\Longrightarrow$ by contraposition: Let $I_{not} \not\models not\,(p_1 \wedge \cdots \wedge p_n)$. Since $\mathcal{N}^\diamond$ is a fixpoint of Θ_P, we have $\mathcal{N}^\diamond = \Pi_P\big(\Omega_P(\mathcal{N}^\diamond)\big)$. By the definition of Π_P, there is a non-empty $\mathcal{O}' \subseteq \Omega_P(\mathcal{N}^\diamond)$ such that the default atoms true in I_{not} are the intersection of the corresponding true negations in $\mathcal{O}'$. Thus, there is $I'_{obj} \in \mathcal{O}'$ with $I'_{obj} \models p_1 \wedge \cdots \wedge p_n$. Furthermore, for all $q_1, \ldots, q_m$ with $I_{not} \models not\,(q_1 \wedge \cdots \wedge q_m)$, we have $I'_{obj} \models \neg q_1 \vee \cdots \vee \neg q_m$. Now by the definition of Ω_P, there is a $I'_{not} \in \mathcal{N}^\diamond$ such that $I' = I'_{obj} \cup I'_{not}$ is a minimal model of P. It follows that $R(I, I')$ holds, i.e. I sees a world with valuation I'_{obj}, in which $\neg p_1 \vee \cdots \vee \neg p_n$ is false. Thus $\mathcal{K}(I) \not\models not\,(p_1 \wedge \cdots \wedge p_n)$. □

LEMMA B.3. *For all reduced interpretations $I \in W$, the full interpretation $\mathcal{I} = \mathcal{K}(I)$ is a model of the least static expansion $\overline{P}$.*

PROOF. We show that the conditions of Theorem A.3 are satisfied:

(1) We first have to show that every world $I = I_{obj} \cup I_{not}$ sees at least one world I'. This follows from $I_{not} \in \mathcal{N}^\diamond$, i.e. $I_{not} \in \Pi_P\big(\Omega_P(\mathcal{N}^\diamond)\big)$. By the definition of Π_P there is a non-empty subset $\mathcal{O} \subseteq \Omega_P(\mathcal{N}^\diamond)$ such that every $I'_{obj} \in \mathcal{O}$ satisfies for all $p_1, \ldots, p_n \in At_{\mathcal{L}}$:

$$I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n) \implies I'_{obj} \models \neg p_1 \vee \cdots \vee \neg p_n.$$

Now by the definition of Ω_P, there is a $I'_{not} \in \mathcal{N}^\diamond$ such that $I' = I'_{obj} \cup I'_{not}$ is a minimal model of P. But then $R(I, I')$ holds.

(2) The reduced interpretations $I \in W$ satisfy the program P. By Lemma B.2, I and $\mathcal{I} = \mathcal{K}(I)$ agree on the atoms occurring in P. Thus, also $\mathcal{I}$ is a model of P.

(3) For every $(I, I') \in R$, the reduced interpretation I' is a minimal model of P. If the full interpretation $\mathcal{I}' = \mathcal{K}(I')$ were not a minimal model of P, i.e. there were a smaller model $\hat{\mathcal{I}}'$ of P, then its reduct $\hat{I}'$ would be a smaller model than I' (using again Lemma B.2 in order to conclude that I' and $\mathcal{I}'$, and thus I' and $\hat{I}'$ agree on the default atoms).

Now Theorem A.3 allows us to conclude that the formulas true in all worlds of W are a static expansion of P. Of course, every $\mathcal{I} = \mathcal{K}(I)$ is a model of this static expansion. But the least static expansion $\overline{P}$ is a subset, so $\mathcal{I}$ is also a model of $\overline{P}$. □

LEMMA B.4. $not\,(\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G) \vdash_{not} not\,(F_1) \wedge \cdots \wedge not\,(F_m) \rightarrow not\,(G)$.

PROOF. This is a simple exercise in applying the axioms of $\mathcal{L}_{not}$: First, we get

$$not\big(F_1 \vee \cdots \vee F_m \vee (\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G)\big) \leftrightarrow not\,(F_1 \vee \cdots \vee F_m \vee G)$$

by the invariance inference rule. Then we apply on both sides the distributive axiom:

$$\begin{aligned} & not\,(F_1) \wedge \cdots \wedge not\,(F_m) \wedge not\,(\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G) \\ \leftrightarrow\ & not\,(F_1) \wedge \cdots \wedge not\,(F_m) \wedge not\,(G). \end{aligned}$$

This implies propositionally:

$$not\,(F_1) \wedge \cdots \wedge not\,(F_m) \wedge not\,(\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G) \rightarrow not\,(G).$$

Now we insert our precondition and get the required formula. □

LEMMA B.5. *Let P be finite, $\mathcal{I}$ be a full model of the least static expansion $\overline{P}$, and let $I = I_{obj} \cup I_{not}$ be its reduct. Then $I_{not} \in \mathcal{N}^{\diamond}$.*

PROOF. We show by induction on k that the default part I_{not} of a model $\mathcal{I}$ of the least static expansion $\overline{P}$ is contained in $\Theta_P^k(\mathcal{N}OT)$. For $k = 0$ this is trivial, since $\mathcal{N}OT$ is the complete set of default interpretations.

Let us assume that $I_{not} \in \Theta_P^k(\mathcal{N}OT)$. We have to show that I_{not} is not "filtered out" by one further application of Θ_P. Let $\mathcal{N} := \Theta_P^k(\mathcal{N}OT)$, $\mathcal{O} := \Omega_P(\mathcal{N})$, and

$$\begin{aligned} \mathcal{O}' := \big\{ I_{obj} \in \mathcal{O} \ :\ & \text{for every default atom } not\,(p_1 \wedge \cdots \wedge p_n): \\ & \quad \text{if } I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n), \\ & \quad \text{then } I_{obj} \models \neg p_1 \vee \cdots \vee \neg p_n \big\}. \end{aligned}$$

We will now show that $\mathcal{O}'$ has the properties required in the definition of Π_P, namely we will show

$$I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n) \iff \text{for all } I'_{obj} \in \mathcal{O}' \colon\ I'_{obj} \models \neg p_1 \vee \cdots \vee \neg p_n.$$

This implies that $\mathcal{O}'$ is non-empty because for $n = 0$ the empty conjunction is logically true. Because of the consistency axiom, $I_{not} \not\models not\,(true)$. But then there must be at least one $I'_{obj} \in \mathcal{O}'$, because otherwise the "for all" on the right hand side would be trivially true. Also the direction $\Longrightarrow$ follows trivially from the construction.

Now we have to show that for every default atom $not\,(p_1 \wedge \cdots \wedge p_n)$ which is false in I_{not}, there is $I'_{obj} \in \mathcal{O}'$ with $I'_{obj} \not\models \neg p_1 \vee \cdots \vee \neg p_n$. Let $not\,(q_{i,1} \wedge \cdots \wedge q_{i,n_i})$, $i = 1, \ldots, m$, be all default atoms true in I_{not} (containing only the finitely many objective propositions occurring in P). Since $\mathcal{I} \models \overline{P}$, the formula

$$\Big(\bigwedge_{i=1}^{m} not\,(q_{i,1} \wedge \cdots \wedge q_{i,n_i})\Big) \ \rightarrow\ not\,(p_1 \wedge \cdots \wedge p_n)$$

cannot be contained in $\overline{P}$ (it is violated by I_{not} and thus by $\mathcal{I}$). But Lemma B.4 shows that the above formula would follow from

$$not\left(\Big(\bigwedge_{i=1}^{m}(\neg q_{i,1} \vee \cdots \vee \neg q_{i,n_i})\Big) \wedge (p_1 \wedge \cdots \wedge p_n)\right).$$

Thus, this formula is also not contained in $\overline{P}$. But the static semantics requires that $not\,(F) \in \overline{P}$ if $\overline{P} \models_{\min} \neg F$. So if $not\,(F) \notin \overline{P}$, there must be a minimal

model of $\overline{P}$ violating $\neg F$, i.e. satisfying F. In our case this means that there is a minimal model $\mathcal{I}'$ of $\overline{P}$ with $\mathcal{I}' \models \neg q_{i,1} \vee \cdots \vee \neg q_{i,n_i}$ for $i = 1, \ldots, m$ and $\mathcal{I}' \models (p_1 \wedge \cdots \wedge p_n)$, i.e. $\mathcal{I}' \not\models \neg p_1 \vee \cdots \vee \neg p_n$. But by our inductive hypothesis, the default part I'_{not} of $\mathcal{I}'$ is contained in $\mathcal{N}$, and thus the objective part I'_{obj} is in $\mathcal{O}$. Since $I'_{obj} \models \neg q_{i,1} \vee \cdots \vee \neg q_{i,n_i}$, it is contained in $\mathcal{O}'$. Thus, I'_{obj} is the required element. □

Now Theorem 4.3 follows directly from Lemma B.3 and Lemma B.5:

THEOREM 4.3 (MODEL-THEORETIC CHARACTERIZATION). *Let P be a finite super program:*

(1) *The operator Θ_P is monotone (in the lattice of subsets of $\mathcal{NOT}$ with the order $\supseteq$) and thus its iteration beginning from the set $\mathcal{NOT}$ (all interpretations of the default atoms, the bottom element of this lattice) has a fixed point $\mathcal{N}^\diamond$.*

(2) *$\mathcal{N}^\diamond$ consists exactly of all $I_{not} \in \mathcal{NOT}$ that are default parts of a full model $\mathcal{I}$ of $\overline{P}$.*

(3) *A reduced interpretation $I_{obj} \cup I_{not}$ is a reduct of a full model $\mathcal{I}$ of $\overline{P}$ iff $I_{not} \in \mathcal{N}^\diamond$ and $I_{obj} \cup I_{not} \models P$.*

PROOF. The monotonicity of Θ_P is obvious: Let $\mathcal{N}_1 \supseteq \mathcal{N}_2$ (since we are working with inverse set inclusion, this means that $\mathcal{N}_1$ is below $\mathcal{N}_2$ in the lattice). Then we get $\Omega_P(\mathcal{N}_1) \supseteq \Omega_P(\mathcal{N}_2)$: any $I_{obj} \in \Omega_P(\mathcal{N}_2)$ is based on a interpretation $I_{not} \in \mathcal{N}_2$, and since $\mathcal{N}_2 \subseteq \mathcal{N}_1$, we also have $I_{obj} \in \Omega_P(\mathcal{N}_1)$. Now let $\mathcal{O}_1 := \Omega_P(\mathcal{N}_1)$ and $\mathcal{O}_2 := \Omega_P(\mathcal{N}_2)$. From $\mathcal{O}_1 \supseteq \mathcal{O}_2$ it also follows that $\Theta_P(\mathcal{N}_1) = \Pi_P(\mathcal{O}_1) \supseteq \Pi_P(\mathcal{O}_2) = \Theta_P(\mathcal{N}_2)$, since any $\mathcal{O}' \subseteq \mathcal{O}_2$ used to construct $I_{not} \in \Pi_P(\mathcal{O}_2)$ is also a subset of $\mathcal{O}_1$.

Part 2 means that for every $I_{not} \in \mathcal{NOT}$:

$$\text{There is a full model } \mathcal{I} \text{ of } \overline{P} \text{ with belief part } I_{not} \iff I_{not} \in \mathcal{N}^\diamond.$$

The direction $\Longrightarrow$ is Lemma B.5, and the direction $\Longleftarrow$ follows from Lemma B.3 and the fact that any default interpretation I_{not} can be extended to a reduced model $I = I_{obj} \cup I_{not}$ of P (by making all objective atoms $At_{\mathcal{L}}$ true in I_{obj}). But then $I \in W$.

Part 3 requires that for every reduced interpretation $I = I_{obj} \cup I_{not}$:

$$\begin{aligned}\text{There is a full model } \mathcal{I} \text{ of } \overline{P} \text{ with reduct } I \\ \iff \quad I_{not} \in \mathcal{N}^\diamond \text{ and } I_{obj} \cup I_{not} \models P.\end{aligned}$$

Here the direction $\Longleftarrow$ is Lemma B.3 and the direction $\Longrightarrow$ follows trivially from Lemma B.5 and $P \subseteq \overline{P}$. □

Remark B.6. Note that we have used the finiteness of P only in the proof to Lemma B.5. So even in the infinite case, our model-theoretic construction yields models of the least static expansion, but it does not necessarily yield all reduced models. One example, where a difference might occur, is $P = \{q_i \vee r_i : i \in \mathbb{N}\} \cup \{r_i \vee p : i \in \mathbb{N}\}$. Our model-theoretic construction excludes an interpretation which makes all $not\,(q_i)$ true and $not\,(p)$ false. It seems plausible that $\overline{P}$ allows such models, because it would need an "infinite implication" to exclude them. But this question needs further research.

C. PROOF OF FIXPOINT CHARACTERIZATION (THEOREM 3.1)

We prove Theorem 3.1 by using the model-theoretic characterization. More specifically, we show that formulas $not\,(E_1) \wedge \cdots \wedge not\,(E_m) \rightarrow not\,(E_0)$ contained in P^n characterize exactly the default interpretations remaining after n applications of Θ_P.

First we prove the following lemma which characterizes minimal models of special knowledge bases.

LEMMA C.1. *Let T be any knowledge base and T_{not} be a set of formulae which contain only default atoms. Minimal models of $T \cup T_{not}$ are precisely those minimal models of T which satisfy T_{not}.*

PROOF. Let $\mathcal{I}$ be a minimal model of $T \cup T_{not}$. Of course, $\mathcal{I}$ is a model of T and of T_{not}. Now suppose that there is a smaller model $\mathcal{I}'$ of T. Since $\mathcal{I}$ and $\mathcal{I}'$ do not differ in the interpretation of default atoms, $\mathcal{I}'$ is also a model of T_{not}, and thus a model of $T \cup T_{not}$. But this contradicts the assumed minimality of $\mathcal{I}$.

Let now $\mathcal{I}$ be a minimal model of T which also satisfies T_{not}. Clearly, $\mathcal{I}$ is a model of $T \cup T_{not}$. Since $\mathcal{I}'$ is also a model of T, the existence of a smaller model $\mathcal{I}'$ would contradict the minimality of $\mathcal{I}$. □

Next, we need the monotonicity of the sequence P^n:

LEMMA C.2. *For every $n \in \mathbb{N}$: $P^n \subseteq P^{n+1}$.*

PROOF. The propositional consequence operator Cn is monotonic and has no influence on the minimal models, so it suffices to show that the sequence $\hat{P}_0 := P$,

$$\hat{P}^{n+1} := P \cup \{ not\, E_1 \wedge \cdots \wedge not\, E_m \rightarrow not\, E_0 \; : \; \hat{P}^n \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow \neg E_0 \}$$

increases monotonically.

The proof is by induction on n. The case $n = 0$ is trivial. For larger n, the inductive hypothesis gives us $\hat{P}^n \supseteq \hat{P}^{n-1}$ and all formulas in $\hat{P}^n \setminus \hat{P}^{n-1}$ contain only default atoms. So Lemma C.1 yields that the minimal models of $\hat{P}^n$ are a subset of the minimal models of $\hat{P}^{n-1}$, and therefore $\hat{P}^{n-1} \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow E_0$ implies $\hat{P}^n \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow E_0$, i.e. $\hat{P}^{n+1} \supseteq \hat{P}^n$. □

Next, we have the small problem that we must in principle look at infinitely many default negation atoms, although we have required our program to be finite. For instance, $not\,(p)$, $not\,(p \wedge p)$, and so on are different default atoms, and in general, default interpretations could assign different truth values to them. However, already P^1 excludes this:

Definition C.3 *(Regular Model).* Let a super program P be given. A default interpretation I_{not} is called regular wrt P iff

(1) If $I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n)$ and $\{p_1, \ldots, p_n\} \subseteq \{q_1, \ldots, q_m\}$, then $I_{not} \models not\,(q_1 \wedge \cdots \wedge q_m)$.
(2) $I_{not} \models not\,(p_1 \wedge \cdots \wedge p_n)$ if some $p_i \in At_{\mathcal{L}}$ does not occur in P.

LEMMA C.4. *All $I_{not} \in \mathcal{N}_i$, $i \geq 1$ are regular.*

PROOF. (1) Every interpretation I satisfies $\neg(p_1 \wedge \cdots \wedge p_n) \rightarrow \neg(q_1 \wedge \cdots \wedge q_m)$, if $\{p_1, \ldots, p_n\} \subseteq \{q_1, \ldots, q_m\}$. So $not\,(p_1 \wedge \cdots \wedge p_n) \rightarrow not\,(q_1 \wedge \cdots \wedge q_m)$ is contained in all P^i, $i \geq 1$.

(2) Every minimal model I of P satisfies $\neg(p_1 \wedge \cdots \wedge p_n)$ if some p_i does not occur in P (because $I \models \neg p_i$). But then $not\,(p_1 \wedge \cdots \wedge p_n) \in P^1$, and by Lemma C.2 it is contained also in every P^n, $n \geq 1$.

□

LEMMA C.5. *Let P^n_{not} be the set of formulas of the form*

$$not\,(E_1) \wedge \cdots \wedge not\,(E_m) \rightarrow not\,(E_0)$$

contained in P^n. Furthermore, let $\mathcal{N}_0 := \mathcal{NOT}$ and $\mathcal{N}_{n+1} := \Theta_P(\mathcal{N}_n)$. Then for every $I = I_{obj} \cup I_{not}$ with $I \models P$: $I_{not} \models P^n_{not} \iff I_{not} \in \mathcal{N}_n$.

PROOF. The proof is by induction on n. The case $n = 0$ is trivial, since $\mathcal{N}_0 = \mathcal{NOT}$ and $P^0 = P$ and we anyway consider only models of P.

(1) " $\Longleftarrow$ ": Let $I_{not} \in \mathcal{N}_{n+1} = \Theta_P(\mathcal{N}_n)$. We have to show that $I_{not} \models P^{n+1}_{not}$. Suppose that this were not the case, i.e. I_{not} violates a formula $not\,E_1 \wedge \cdots \wedge not\,E_m \rightarrow not\,E_0$ contained in P^{n+1}_{not}. This means that $I_{not} \models not\,E_i$ for $i = 1, \ldots, m$, but $I_{not} \not\models not\,E_0$. By the definition of Θ_P, there must be an objective model $I'_{obj} \in \Omega_P(\mathcal{N}_n)$ (contained in the non-empty $\mathcal{O}'$) such that $I'_{obj} \models \neg E_i$ for $i = 1, \ldots, m$ and $I'_{obj} \not\models \neg E_0$. By the definition of Ω_P, there must be a default interpretation $I'_{not} \in \mathcal{N}_n$ such that $I' = I'_{obj} \cup I'_{not}$ is a minimal model of P. Then the inductive hypothesis gives us $I'_{not} \models P^n_{not}$, and Lemma C.1 allows us to conclude that I' is a minimal model of $P \cup P^n_{not}$ and thus of P^n. But this means that $P^n \not\models_{\min} not\,E_1 \wedge \cdots \wedge not\,E_m \rightarrow not\,E_0$. Since I' is a model of P, the critical formula cannot be contained in P (if E_0 is the empty conjunction and P is not affirmative, this would be syntactically possible). And finally, it cannot be introduced by the Cn-operator, since I' is a model of its preconditions. Thus, it is impossible that $not\,E_1 \wedge \cdots \wedge not\,E_m \rightarrow not\,E_0$ is contained in P^{n+1}_{not}.

(2) " $\Longrightarrow$ ": Let $I_{not} \models P^{n+1}_{not}$. By Lemma C.2 we have $P^n_{not} \subseteq P^{n+1}_{not}$, so $I_{not} \models P^n_{not}$, and the inductive hypothesis gives us $I_{not} \in \mathcal{N}_n$. We have to show that I_{not} is not "filtered out" by one further application of the Θ_P-operator. Let

$$\mathcal{O}' := \{I'_{obj} \in \Omega_P(\mathcal{N}_n) \,:\, I'_{obj} \models \neg E \text{ for all } E \text{ with } I_{not} \models not\,E\}.$$

We have to show that for every E_0 with $I_{not} \not\models not\,E_0$ there is an $I'_{obj} \in \mathcal{O}'$ with $I'_{obj} \not\models \neg E_0$. This especially implies that $\mathcal{O}'$ is non-empty, since $I_{not} \not\models not\,(true)$, which is obviously contained in P^{n+1}_{not}.

Now suppose that this were not the case, i.e. there were an E_0 such that there is no $I'_{obj} \in \mathcal{O}'$ with $I'_{obj} \not\models \neg E_0$.

Let $not\,(E_i)$, $i = 1, \ldots, m$, be all belief atoms which are true in I_{not} and satisfy the following conditions (in order to make the set finite): First, the conjunctions E_i contain only propositions $p \in At_{\mathcal{L}}$ occurring in P (which was required to be finite), and second, each E_i contains each proposition at most once.

We now prove that $not\, E_1 \wedge \cdots \wedge not\, E_m \rightarrow not\, E_0$ is contained in P^{n+1}_{not}, which contradicts $I_{not} \models P^{n+1}_{not}$. Let $I' = I'_{obj} \cup I'_{not}$ be any minimal model of P^n. We have to show that it satisfies $\neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow \neg E_0$. The induction hypothesis gives us $I'_{not} \in \mathcal{N}_n$, and thus $I'_{obj} \in \Omega_P(\mathcal{N}_n)$. Suppose that $I'_{obj} \not\models \neg E_i$ for $i = 1, \ldots, m$, since otherwise the formula is trivially satisfied. Since I' is a minimal model and I_{not} is regular, this means that $I'_{obj} \in \mathcal{O}'$ (the validity of $\neg E_1, \ldots, \neg E_m$ implies the validity of all other $\neg E$ considered in the construction of $\mathcal{O}'$). But we have assumed that no element of $\mathcal{O}'$ violates $\neg E_0$, thus $I'_{obj} \models \neg E_0$.

□

Lemma C.6. $P^{n_0} \subseteq \overline{P}|\mathcal{L}^*_{not}$.

Proof. We show $P^n_{not} \subseteq \overline{P}$ by induction on n (this implies the above statement since $\overline{P}$ is closed under consequences and $P^{n_0} \subseteq \mathcal{L}^*_{not}$).

For $n = 0$ this is trivial since $P \subseteq \overline{P}$. Now suppose that $P^n \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow \neg E_0$. By Theorem 3.8 in [Brass et al. 1999] and the definition of Cn_{not}, $\overline{P}$ differs from P only by the addition of formulas containing only default atoms plus propositional consequences. By the inductive hypothesis, $P^n_{not} \subseteq \overline{P}$. Now Lemma C.1 gives us that all minimal models of $\overline{P}$ are also minimal models of P^n, and therefore $\overline{P} \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \rightarrow \neg E_0$, i.e. $\overline{P} \models_{\min} \neg E_1 \wedge \cdots \wedge \neg E_m \wedge E_0$. This means that $not\,(\neg E_1 \wedge \cdots \wedge \neg E_m \wedge E_0) \in \overline{P}$, and by Lemma B.4 we get that $not\, E_1 \wedge \cdots \wedge not\, E_m \rightarrow not\, E_0$ is contained in $\overline{P}$. □

Now we can complete the proof of Theorem 3.1. First, a fixpoint is reached after a finite number of iterations, because by Lemma C.4 we know that there are only a finite number of "really different" default negation atoms, so after the first iteration (which ensures the regularity) it suffices to consider a finite number of implications $not\, E_1 \wedge \cdots \wedge not\, E_m \rightarrow not\, E_0$, and the sets P^n are monotonically increasing (Lemma C.2).

So we have $P^{n_0}_{not} = P^{n_0+1}_{not}$ and thus $\mathcal{N}_{n_0} = \mathcal{N}_{n_0+1} = \mathcal{N}^\diamond$. Now for any reduced interpretation $I = I_{obj} \cup I_{not}$: if $I \models P^{n_0}$, then $I \models P$ and $I \models P^{n_0}_{not}$, and by Lemma C.5 we get $I_{not} \in \mathcal{N}^\diamond$. Now the already proven Theorem 4.3 implies that I is a reduct of a full model of the least static expansion $\overline{P}$. But this implies $I \models \overline{P}|\mathcal{L}^*_{not}$. The other direction $I \models \overline{P}|\mathcal{L}^*_{not} \implies I \models P^{n_0}$ follows from Lemma C.6.

From the equivalence of P^{n_0} and $\overline{P}|\mathcal{L}^*_{not}$ we get $P^{n_0} = \overline{P}|\mathcal{L}^*_{not}$, since both sets are closed under propositional consequences: For instance, let $F \in P^{n_0}$ and suppose that $F \notin \overline{P}|\mathcal{L}^*_{not}$. Since $\overline{P}|\mathcal{L}^*_{not}$ is closed under propositional consequences, there must be a reduced interpretation I with $I \models \overline{P}|\mathcal{L}^*_{not}$, but $I \not\models F$. This is impossible since we already know that every model of $\overline{P}|\mathcal{L}^*_{not}$ is also a model of P^{n_0}.

Remark C.7. The finiteness of P was used in the proof for $I_{not} \models P^n_{not} \implies I_{not} \in \mathcal{N}_n$. This was to be expected, since we strongly conjecture that for infinite programs, the model-theoretic construction does not yield all models of the static completion. However, this does not give us any hint whether Theorem 3.1 might hold for infinite programs. This question is topic of our future research.

D. PROOFS OF PROPERTIES USED IN THE IMPLEMENTATION

THEOREM 5.3. *Let P be a super logic program, $\mathcal{F}_0 := \emptyset$, and $\mathcal{F}_{i+1} := H_P(\mathcal{F}_i)$, and n be a natural number such that $\mathcal{F}_{n+1} = \mathcal{F}_n$. Then the following holds for all Herbrand interpretations I: I is a minimal model of the ground instantiation P^* of P if and only if I is a minimal model of $\mathcal{F}_n$.*

PROOF. First we show that a (minimal) model of one of P^* and $\mathcal{F}_n$ is also a model of the other:

(1) The conditional facts in $H_P(\mathcal{F}_i)$ are logical consequences of $P^* \cup \mathcal{F}_i$. By induction on i, it follows that $\mathcal{F}_i$ is implied by P^*. Therefore, every model of P^* is also a model of $\mathcal{F}_n$.

(2) Next, we prove that every minimal model of $\mathcal{F}_n$ is also a model of P^*: Suppose that this would not be the case, i.e. I is a minimal model of $\mathcal{F}_n$, but it violates a rule

$$A_1 \vee \ldots \vee A_k \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_l$$

in P^*. This means that $B_1, \ldots, B_m$ are true in I. Since I is a minimal model of $\mathcal{F}_l$, $\mathcal{F}_l$ contains for $i = 1, \ldots, m$ a conditional fact $\mathcal{A}_i \leftarrow \mathcal{C}_i$ that is violated in the interpretation $I \setminus \{B_i\}$ (i.e. the interpretation that agrees with I except that B_i is false in it). It follows that $B_i \in \mathcal{A}_i$, that $\mathcal{A}_i \setminus \{B_i\}$ is false in I, and that $\mathcal{C}_i$ is true in I (since the default negation literals are treated like new propositions, making B_i false does not change any of the default negation literals). Now consider the conditional fact that is derived from the rule instance and the $\mathcal{A}_i \leftarrow \mathcal{C}_i$:

$$\{A_1\sigma, \ldots, A_k\sigma\} \cup (\mathcal{A}_1 \setminus \{B_1\sigma\}) \cup \ldots \cup (\mathcal{A}_m \setminus \{B_m\sigma\}) \leftarrow \{not\, C_1\sigma, \ldots, not\, C_l\sigma\} \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_m$$

Since $\mathcal{F}_n$ is a fixpoint of H_P, this fact is also contained in $\mathcal{F}_n$. But this is a contradiction, since it is violated in I.

Now let I be a minimal model of P^*. (1) shows that it is a model of $\mathcal{F}_n$. If it were not minimal, there would be a smaller model I_1 of $\mathcal{F}_n$. Then also a minimal model I_0 of $\mathcal{F}_n$ must exist that is still smaller than (or equal to) I_1. But by (2) above, I_0 is also a model of P^* which contradicts the assumed minimality of I.

Let conversely I be a minimal model of $\mathcal{F}_n$. By (2), it is a model of P^*. If it were not minimal, there would be a smaller model I_0 of P^*, which is by (1) also a model of $\mathcal{F}_n$, which again contradicts the assumed minimality of I. □

LEMMA D.1. *Let T_1 be a nonmonotonic knowledge base with $T_1 \models_{\min} F$. Let T_2 be a set of default negation atoms, i.e. formulas of the form not G with an arbitrary formula G. Then $Cn_{not}(T_1 \cup T_2) \models_{\min} F$.*

PROOF. If this were not the case, there were a minimal model I of $Cn_{not}(T_1 \cup T_2)$ which does not satisfy $\neg F$. Because of $T_1 \models_{\min} \neg F$ and $T_1 \subseteq Cn_{not}(T_1 \cup T_2)$, there must be a model I_0 of T_1 that is smaller than I, but not a model of $Cn_{not}(T_1 \cup T_2)$. Since I_0 is smaller than I, it assignes the same truth values to the default negation literals. But then it satisfies T_2, i.e. it is a model of $T_1 \cup T_2$. Thus, a violated formula must be one that is added by Cn_{not}. Take the first such formula. It

cannot be a propositional consequence, because propositional consequences are by definition satisfied in all models that satisfy the preconditions (and until this first formula, it satisfied all preconditions). But it can also not be added by (CA), (DA), or (IR), because all these formulas consist only of default negation literals, which are interpreted the same in both models. Therefore, we can conclude $I_0 \models Cn_{not}(T_1, \ldots, T_2)$. But that contradicts the assumed minimality of I. □

THEOREM 5.6. *(1) Let T be a knowledge base with $T \models_{\min} \neg F$. Then T and $T \cup \{not\, F\}$ have the same static expansions.*

(2) Let T_1 and T_2 be knowledge bases with $Cn_{not}(T_1) = Cn_{not}(T_2)$. Then T_1 and T_2 have the same static expansions.

PROOF. (1) Let $T^\diamond$ be a static expansion of T. By the lemma above, $T^\diamond \models_{\min} \neg F$. But then it easily follows that $T^\diamond$ is also a static expansion of $T \cup \{not\, F\}$: It has to satisfy

$$T^\diamond = Cn_{not}\big(T \cup \{not\, F\} \cup \{not\, G : T^\diamond \models_{\min} \neg G\}\big).$$

Since $T^\diamond \models_{\min} \neg F$, the formula $not\, F$ is anyway contained in the preconditions of Cn_{not}, so the union with $\{not\, F\}$ changes nothing.
Assume conversely that $T^\diamond$ is a static expansion of $T \cup \{not\, F\}$, i.e.

$$T^\diamond = Cn_{not}\big(T \cup \{not\, F\} \cup \{not\, G : T^\diamond \models_{\min} \neg G\}\big).$$

Again by the lemma above we get $T^\diamond \models_{\min} \neg F$. But this means that the preconditions are not changed when we do not add $\{not\, F\}$ explicitly to the preconditions:

$$T^\diamond = Cn_{not}\big(T \cup \{not\, G : T^\diamond \models_{\min} \neg G\}\big).$$

(2) This follows with the following sequence of equations:

$$\begin{aligned} T^\diamond &= Cn_{not}\big(T_1 \cup \{not\, F : T^\diamond \models_{\min} \neg F\}\big) \\ &= Cn_{not}\Big(Cn_{not}(T_1) \cup \{not\, F : T^\diamond \models_{\min} \neg F\}\Big) \\ &= Cn_{not}\Big(Cn_{not}(T_2) \cup \{not\, F : T^\diamond \models_{\min} \neg F\}\Big) \\ &= Cn_{not}\big(T_2 \cup \{not\, F : T^\diamond \models_{\min} \neg F\}\big). \end{aligned}$$

□

THEOREM 5.8. *Let $\mathcal{D}$ be a set of disjunctions of objective atoms, which does not contain the empty disjunction false, and which does not contain two disjunctions $\mathcal{A}$ and $\mathcal{A}'$, such that $\mathcal{A} \subset \mathcal{A}'$ (i.e. $\mathcal{A}'$ is non-minimal). Let I be a partial interpretation such that atoms interpreted as false do not appear in $\mathcal{D}$ and atoms interpreted as true appear as facts in $\mathcal{D}$. Let p be an atom that appears in the proper disjunction*

$$p_1 \vee \cdots \vee p_{i-1} \vee p \vee p_{i+1} \vee \cdots \vee p_n$$

(and possibly more such disjunctions). Then

(1) There is a minimal model of $\mathcal{D}$ that extends I and interprets p as false.

(2) There is a minimal model of $\mathcal{D}$ that extends I and interprets p as true and $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ as false.

PROOF. (1) Consider the interpretation I' that extends I by interpreting p as false and all remaining atoms as true. Suppose that it were not a model of $\mathcal{D}$, i.e. it would violate some disjunction in $\mathcal{D}$. Since all atoms that I interprets as false do not appear in $\mathcal{D}$, and all the remaining atoms except p are interpreted as true, the violated disjunction can only be p. But this is a contradiction, since then the proper disjunction $p_1 \vee \cdots \vee p_{i-1} \vee p \vee p_{i+1} \vee \cdots \vee p_n$ would not be minimal. Thus, I' is a model of $\mathcal{D}$. Then there is also a minimal model I_0 of $\mathcal{D}$ that is less than or equal to I'. The atoms interpreted as false in I as well as p must be false in I_0, since it is less or equal to I'. The atoms interpreted as true in I appear as facts in $\mathcal{D}$, so they must be true in I_0.

(2) Consider the interpretation I' that extends I by interpreting p as true, the remaining atoms of the disjunction $(p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n)$ as false, and all other atoms as true. Suppose that it were not a model of $\mathcal{D}$, i.e. it would violate some disjunction in $\mathcal{D}$. Since all atoms that I interprets as false do not appear in $\mathcal{D}$, and all the remaining atoms except $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ are interpreted as true, the atoms in the violated disjunction can only be a subset of $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}$. But this contradicts the assumed minimality of the disjunction $p_1 \vee \cdots \vee p_{i-1} \vee p \vee p_{i+1} \vee \cdots \vee p_n$. Thus, I' is a model of $\mathcal{D}$. Then there is also a minimal model I_0 of $\mathcal{D}$ that is less than or equal to I'. This means that the atoms interpreted as false in I as well as $p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n$ must be false in I_0. All atoms interpreted as true in I must be true in I_0, since they appear as facts in $\mathcal{D}$. Finally, also p must be true in I_0, since otherwise it would violate the disjunction $p_1 \vee \cdots \vee p_{i-1} \vee p \vee p_{i+1} \vee \cdots \vee p_n$.

□

ACKNOWLEDGMENTS

The authors would like to express their deep appreciation to Luis Moniz Pereira and Jose Alferes for their helpful comments. The authors are also greatly indebted to the anonymous reviewers for their detailed and very insightful comments which have lead to a significant improvement of this article.

REFERENCES

ALFERES, J., PEREIRA, L. M., AND PRZYMUSINSKI, T. C. 1998. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning 20*, 107–142.

APT, K. R., BLAIR, H. A., AND WALKER, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases*, J. Minker, Ed. Morgan Kaufmann, Chapter 2, 89–148.

ARAVINDAN, C. AND BAUMGARTNER, P. 1997. A Rational and Efficient Algorithm for View Deletion in Databases. In *Logic Programming: Proceedings of the 1997 International Symposium*, J. Maluszynski, Ed. MIT Press, 165–180.

ARAVINDAN, C., DIX, J., AND NIEMELÄ, I. 1997. Dislop: A research project on disjunctive logic programming. *AI Communications 10*, 151–165.

BAUMGARTNER, P., FRÖHLICH, P., FURBACH, U., AND NEJDL, W. 1997. Semantically Guided Theorem Proving for Diagnosis Applications. In *15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, M. Pollack, Ed. Morgan Kaufmann, Nagoya, 460–465.

BRASS, S. 1996. Bottom-up query evaluation in extended deductive databases. Habilitation Thesis, University of Hannover. `http://www-db.informatik.uni-hannover.de/~sb/habil.html`.

Brass, S. and Dix, J. 1995. A general approach to bottom-up computation of disjunctive semantics. In *Nonmonotonic Extensions of Logic Programming*, J. Dix, L. M. Pereira, and T. C. Przymusinski, Eds. Number 927 in LNAI. Springer, 127–155.

Brass, S. and Dix, J. 1997. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming 32(3)*, 207–228. (Extended abstract appeared in: Characterizations of the Stable Semantics by Partial Evaluation *LPNMR, Proceedings of the Third International Conference, Kentucky*, pages 85–98, 1995. LNCS 928, Springer.).

Brass, S. and Dix, J. 1999. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming 38,* 3, 167–213. (Extended abstract appeared in: Disjunctive Semantics Based upon Partial and Bottom-Up Evaluation, *Proceedings of the 12-th International Logic Programming Conference, Tokyo*, pages 199–213, 1995. MIT Press.).

Brass, S., Dix, J., and Przymusinski, T. C. 1999. Computation of the semantics of autoepistemic belief theories. *Artificial Intelligence 112,* 1-2, 233–250.

Brass, S. and Lipeck, U. W. 1993. Bottom-up query evaluation with partially ordered defaults. In *Deductive and Object-Oriented Databases, Third Int. Conf., (DOOD'93)*, S. Ceri, K. Tanaka, and S. Tsur, Eds. Number 760 in LNCS. Springer, Berlin, 253–266.

Bry, F. 1989. Logic programming as constructivism: A formalization and its application to databases. In *Proceedings of the Symposium on Principles of Database Systems*. ACM SIGACT-SIGMOD, 34–50.

Bry, F. 1990. Negation in logic programming: A formalization in constructive logic. In *Information Systems and Artificial Intelligence: Integration Aspects*, D. Karagiannis, Ed. Springer, Berlin, 30–46.

Bry, F. 1996. A compositional semantics for logic programs and deductive databases. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, M. Maher, Ed. The MIT Press, Bonn, Germany, 453–467.

Bry, F. and Yahya, A. 1996. Minimal model generation with positive unit hyper-resolution tableaux. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*. Springer-Verlag, Terrasini, Italy, 143–159.

Buccafurri, F., Leone, N., and Rullo, P. 1997. Strong and Weak Constraints in Disjunctive Datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)*. Dagstuhl, Germany, 2–17.

Bugliesi, M., Lamma, E., and Mello, P. 1994. Modularity in logic programming. *Journal of Logic Programming 20*, 443–502.

Cholewiński, P., Marek, V., Mikitiuk, A., and Truszczyński, M. 1995. Experimenting with nonmonotonic reasoning. In *Proceedings of the 12th International Conference on Logic Programming*. Tokyo, 267–281.

Dix, J. 1995a. A Classification-Theory of Semantics of Normal Logic Programs: II. Weak Properties. *Fundamenta Informaticae XXII(3)*, 257–288.

Dix, J. 1995b. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, A. Fuhrmann and H. Rott, Eds. DeGruyter, 241–327.

Dix, J., Pereira, L., and Przymusinski, T., Eds. 1998. *Logic Programming and Knowledge Representation*. LNAI. Springer, Berlin.

Dix, J., Pereira, L. M., and Przymusinski, T. C., Eds. 1995. *Non-Monotonic Extensions of Logic Programming*. LNAI 927. Springer Verlag, Berlin. Proceedings of the Workshop at the Eleventh International Logic Programming Conference, ICLP'94, Santa Margherita Ligure, Italy, June 1994.

Dix, J., Pereira, L. M., and Przymusinski, T. C., Eds. 1997. *Non-Monotonic Extensions of Logic Programming*. LNAI 1216. Springer Verlag, Berlin. Proceedings of the Workshop at the International Logic Programming Conference, JICSLP'96, Bonn, Germany, September 1996.

Dung, P. M. and Kanchanasut, K. 1989. A Fixpoint Approach to Declarative Semantics of Logic Programs. In *Proceedings of the North American Conference on Logic Programming*, E. L. Lusk and R. A. Overbeek, Eds. MIT Press, Cambridge, Mass.

Dyckhoff, R., Herre, H., and Schroeder-Heister, P., Eds. 1996. *Extensions of Logic Programming*. LNAI 1050. Springer, Berlin.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. 1997. A Deductive System for Nonmonotonic Reasoning. In *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '97)*, J. Dix, U. Furbach, and A. Nerode, Eds. Number 1265 in Lecture Notes in AI (LNAI). Springer, Berlin.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., and Scarcello, F. 1998. The KR System `dlv`: Progress Report, Comparisons and Benchmarks. In *Proceedings Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*. Forthcoming.

Gelfond, M. and Lifschitz, V. 1988. The Stable Model Semantics for Logic Programming. In *5th Conference on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing 9*, 365–387. (Extended abstract appeared in: Logic Programs with Classical Negation. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 579-597, 1990. MIT Press.).

Gelfond, M., Przymusinska, H., and Przymusinski, T. 1989. On the Relationship between Circumscription and Negation as Failure. *Artificial Intelligence 38*, 75–94.

Greco, S., Leone, N., and Scarcello, F. 1998. Disjunctive datalog with nested rules. In *Logic Programming and Knowledge Representation*, J. Dix, L. Pereira, and T. Przymusinski, Eds. LNAI. Springer, Berlin.

Hodas, J. S. and Miller, D. 1994. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation 110,* 2, 327–365.

Lifschitz, V., Tang, L., and Turner, H. 1998. Nested expressions in Logic Programs. In *Proceedings of the LP-Track of the NMR-WS, preceding KR '98, Trento, Italy*, J. Dix and J. Lobo, Eds. University of Koblenz, TR 3/98, 10–20.

Lipski, Jr., W. 1979. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems 4*, 262–296.

Lloyd, J. W. and Topor, R. W. 1984. Making Prolog more expressive. *The Journal of Logic Programming 1*, 225–240.

Lloyd, J. W. and Topor, R. W. 1985. A basis for deductive database systems. *The Journal of Logic Programming 2*, 93–109.

Lloyd, J. W. and Topor, R. W. 1986. A basis for deductive database systems II. *The Journal of Logic Programming 3*, 55–67.

Lobo, J., Minker, J., and Rajasekar, A. 1992. *Foundations of Disjunctive Logic Programming*. MIT-Press.

Marek, W. and Truszczyński, M. 1993. *Nonmonotonic Logics; Context-Dependent Reasoning*, 1st ed. Springer, Berlin.

McCarthy, J. 1980. Circumscription: A Form of Nonmonotonic Reasoning. *Artificial Intelligence 13*, 27–39.

Minker, J. 1982. On indefinite databases and the closed world assumption. In *Proceedings of the 6th Conference on Automated Deduction, New York*. Springer, Berlin, 292–308.

Minker, J. 1993. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue 17,* 2/3/4, 95–126.

Minker, J. 1996. Logic and databases: A 20 year retrospective. In *Proceedings of the International Workshop on Logic in Databases (LID)*, D. Pedreschi and C. Zaniolo, Eds. LNCS 1154. Springer, Berlin, 3–58.

Nadathur, G. and Miller, D. 1990. Higher-order horn clauses. *Journal of the Association for Computing Machinery (JACM) 37,* 4, 777–814.

Niemelä, I. 1996. A tableau calculus for minimal model reasoning. In *Proceedings of the Fifth Workshop on Theorem Proving with Analytic Tableaux and Related Methods*, P. Miglioli, U. Moscato, D. Mundici, and M. Ornaghi, Eds. LNAI 1071, Springer-Verlag, Terrasini, Italy, 278–294.

Niemelä, I. and Simons, P. 1996. Efficient Implementation of the Well-founded and Stable Model Semantics. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, M. Maher, Ed. The MIT Press, Bonn, Germany, 289–303.

Przymusinski, T. 1995a. Semantics of normal and disjunctive logic programs: A unifying framework. In *Proceedings of the Workshop on Non-Monotonic Extensions of Logic Programming at the Eleventh International Logic Programming Conference, ICLP'94, Santa Margherita Ligure, Italy, June 1994*, J. Dix, L. Pereira, and T. Przymusinski, Eds. Springer, 43–67.

Przymusinski, T. C. 1990. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae 13,* 4, 445–464.

Przymusinski, T. C. 1994. A knowledge representation framework based on autoepistemic logic of minimal beliefs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94, Seattle, Washington, August 1994*. Proceedings of the Conference of the American Association of Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 952–959.

Przymusinski, T. C. 1995b. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence 14*, 323–357. Special Issue on Disjunctive Programs.

Przymusinski, T. C. 1998. Autoepistemic logic of knowledge and beliefs. *Artificial Intelligence 95*, 115–154.

Schäfer, D. and Neugebauer, G. 1997. Opening the world to theorem provers. In *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Fourth International Conference*, J. Dix, U. Furbach, and A. Nerode, Eds. LNAI 1265. Springer, Berlin, 410–419.

Seipel, D. 1994. An efficient computation of the extended generalized closed-world assumption by support-for-negation sets. In *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'94)*. LNAI 822. Springer, 245–259.

Stolzenburg, F. and Thomas, B. 1996. Analysing rule sets for the calculation of banking fees by a theorem prover with constraints. In *Proceedings of the 2nd International Conference on Practical Application of Constraint Technology*. Practical Application Company, London, 269–282.

Stolzenburg, F. and Thomas, B. 1998. Analysing Rule Sets for the Calculation of Banking Fees by a Theorem Prover. In *Automated Deduction — A Basis for Applications, Volume III*, W. Bibel and P. H. Schmitt, Eds. Kluwer Academic Publishers, 243–264.

Teusink, F. and Etalle, S. 1996. A compositional semantics for normal open logic programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, M. Maher, Ed. The MIT Press, Bonn, Germany, 468–482.

Thomas, B. 1998. Intelligent web querying with logic programming. In *Proceedings of the Workshop on Inference Systems in Knowledge-based Systems, preceding the national german AI conference KI '98, Bremen, Germany*, J. Dix and S. Hölldobler, Eds. University of Koblenz, TR 10/98.

Van Gelder, A., Ross, K. A., and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM 38*, 620–650.

You, J.-H. and Yuan, L.-Y. 1993. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning 10*, 143–160.

Received October 2000; revised March 2002; accepted May 2002