Super Logic Programs [1]

Stefan Brass [a] Jürgen Dix [b] Teodor C. Przymusinski [c,2]

[a] *School of Information Sciences, University of Pittsburgh, PA 15260, USA*

[b] *University of Koblenz, Dept. of Computer Science, D-56075 Koblenz, Germany*

[c] *Dept. of Computer Science, University of California, Riverside, CA 92521, USA*

Abstract

Recent extensive research on *non-monotonic reasoning and logic programming* has clearly demonstrated that logic programs are sufficiently expressive to allow formalizations of many important problems in non-monotonic reasoning, as well as providing an important test bed for new such formalizations. However, standard logic programs suffer from some important limitations. Most importantly, they are unable to handle disjunctive information and they allow only one type of negation. Consequently, considerable interest and research effort has been recently given to the problem of finding a suitable *extension of the logic programming paradigm*.

We argue that in order to demonstrate that a class of programs can be justifiably called an *extension of logic programs* one should be able to show that their *syntax* resembles the syntax of logic programs but it applies to a significantly broader class of programs and that their *semantics* constitutes an intuitively natural extension of the semantics of normal logic programs. Moreover, their semantics should be closely related to other well-established non-monotonic formalisms and there should exist a reasonably simple procedural mechanism allowing us to compute it.

In this paper we propose a specific class of such logic programs which will be called *super logic programs*. We show that the class of super logic programs satisfies all of the above conditions, and, in addition, is sufficiently flexible to allow various application-dependent extensions and modifications. We also provide a brief description of a Prolog implementation of a *query-answering interpreter* for super programs.

Key words: Non-Monotonic Reasoning, Logics of Knowledge and Beliefs, Semantics of Logic Programs and Deductive Databases.

[1] An extended abstract of this paper appeared in the Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96), L. C. Aiello, J. Doyle and S. C. Shapiro (editors), 1996, pp. 529–541.

[2] Partially supported by the National Science Foundation grant #IRI-9313061.

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 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.

Recent extensive research on *non-monotonic reasoning and logic programming* has made significant contributions towards the better understanding of relations existing between logic programming and various other formalizations of non-monotonic reasoning, and, towards the eventual discovery of deeper underlying principles of non-monotonic reasoning as such. It has established quite clearly that while logic programs represent a relatively simple non-monotonic formalism, they are sufficiently expressive to allow formalizations of many important problems in non-monotonic reasoning, as well as providing fertile testing grounds for new such formalizations. Moreover, as shown by a number of papers (see, e.g., [34]), it is often possible to translate (compile) non-monotonic theories, which are originally *not* represented as logic programs, into logic programs. The advantages of such translation include simplicity and naturality of knowledge representation by means of logic programs as well as the existence of relatively efficient query answering mechanisms for logic programs.

However, standard logic programs suffer from some important limitations. Most importantly, they are unable to handle disjunctive information. 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:

- *Approximate information:* for instance, an age "around 30" can be 28, 29, 30, 31, or 32;
- *Legal rules:* the judge always has some freedom for his decision, otherwise he/she would not be needed; so laws cannot have unique models;
- *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;
- *Biological inheritance:* if the parents have blood groups A and 0, the child must also have one of these two blood groups (example from [42]);
- *Natural language understanding:* here there are many possibilities for ambiguity and they are represented most naturally by multiple intended models;

- *Reasoning about concurrent processes:* since we do not know the exact sequence in which certain operations are performed, again multiple models come into play;
- *Conflicts in multiple inheritance:* if we want to keep as much information as possible, we should assume disjunction of the inherited values, see [12].

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. Yet, somewhat surprisingly, not enough attention has been devoted to this issue in the area of non-monotonic knowledge representation. For example, it is well-known that default logic has a great deal of difficulty with the handling of disjunctive knowledge [55], although some attempts have been made in the literature to correct this problem [37]. Similar comments apply to autoepistemic logic.

Consequently, considerable interest and research effort [3] 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 [4].

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:

- 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 [36, 1];
- the proposed *semantics* of such programs constitutes an intuitively natural extension of one (or more) well-established semantics of normal logic programs;
- there exists a reasonably simple procedural mechanism allowing, at least in principle, to compute the semantics [5];

[3] It suffices just to mention several recent workshops on *Extensions of Logic Programming* specifically devoted to this subject ([24, 25, 28, 21]).

[4] The book by Minker et. al. [46] provides a detailed account of the extensive research effort in this area. See also [23, 50, 57, 59, 51].

[5] 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.

- 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.

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. We also provide a brief description of a Prolog implementation of a *query-answering interpreter* for the class of super-programs which is available via FTP and WWW [6] .

Since standard logic programs do not allow disjunctive information, they extend only a fragment of classical logic, called the Horn logic. On the other hand, super logic programs, augmented with the expressive power of true disjunctive reasoning, provide a powerful formalism that can be viewed, in some sense, as the simplest non-monotonic extension of the *full* classical logic. The additional expressive power of disjunctive logic programs [29, 31, 30] significantly simplifies *translation* of informal specifications, as well as specifications described in other non-monotonic formalisms, into logic programs. Consequently, it facilitates using logic programming as a language for knowledge representation and thus also as an *inference engine* for non-monotonic reasoning.

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 [56, 60]. 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 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 commonsense 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*, a rather simple and yet highly expressive class of non-monotonic theories introduced earlier by Przymusinski

[6] See `http://www-db.informatik.uni-hannover.de/software/static/`.

[56, 60]. Subsequently, in Section 3, we establish a new and powerful fixed-point characterization of the semantics of such knowledge bases, which provides the foundation for the remaining results obtained in the paper. In Section 4 we introduce the class of super logic programs as a special subclass of the class of all non-monotonic knowledge bases and we establish basic properties of super programs. In the following Sections 5 and 6 we describe two important characterizations, one of which is syntactic and the other model-theoretic, of the semantics of super-programs which, due to the restricted nature of super programs, are significantly simpler than those applicable to arbitrary non-monotonic knowledge bases. Finally, in Section 7 we briefly describe our implementation of a *query-answering interpreter* for super-programs which is based on the previously established model-theoretic characterization of their semantics. We conclude with some final remarks in Section 8. For the sake of clarity, most proofs are contained in the Appendix.

Throughout the paper, we restrict our attention to *propositional* programs. Since we can always consider a propositional instantiation of the program, from a semantic standpoint, this restriction does not limit the generality of the obtained results.

2 Non-Monotonic Knowledge Bases

We first define the notion of a non-monotonic knowledge base. Super logic programs, defined in Section 4, are special knowledge bases.

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*. 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 [56, 60])
By a non-monotonic knowledge base *we mean an arbitrary theory in the propositional language $\mathcal{L}_{Not}$, i.e., a (possibly infinite) set of clauses of the form:*

$$B_1 \wedge \ldots \wedge B_m \wedge Not\, G_1 \wedge \ldots \wedge Not\, G_k \rightarrow A_1 \vee \ldots \vee A_l \vee Not\, F_1 \vee \ldots \vee Not\, F_n$$

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 $l \neq 0$.*

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

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. On the other hand, *rational* knowledge bases are those theories that do not contain any default atoms $Not\, F_i$ in heads of their clauses. Observe that arbitrarily deep level of *nested default negations* is allowed in the language.

We assume the following two simple axiom schemata and one inference rule describing the arguably obvious properties of default atoms:

(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 refer to Remark 2.2 for a discussion of why, in general, we do not assume the distributive axiom for conjunction:

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

A (propositional) interpretation of $\mathcal{L}_{Not}$ is a mapping

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

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 2.2 (Derivable Formulae [56, 60])
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 $Cn_{Not}(T) \nvdash_{Not} false$.*

2.1 Equivalence to Belief Theories

The notion of a *non-monotonic knowledge base* defined above is completely equivalent to the notion of a *belief theory* in the *Autoepistemic Logic of Beliefs*, *AEB*, originally defined by Przymusinski in [56, 60].

Belief theories were defined using a propositional language $\mathcal{L}$ augmented with a *belief operator* $\mathcal{B}$ instead of the default negation operator *Not*. In order to *translate* a non-monotonic knowledge base into a belief theory all one needs to do is to replace everywhere $Not\,F$ by $\mathcal{B}\neg F$ thus giving the default negation $Not\,F$ the intended meaning of "$\neg F$ is believed":

$$Not\,F \ \equiv\ \mathcal{B}\neg F.$$

Conversely, in order to *translate* a belief theory into a non-monotonic knowledge base all one needs to do is to replace everywhere $\mathcal{B}F$ by $Not\,\neg F$ thus giving the belief atoms $\mathcal{B}F$ the intended meaning of "F can be assumed true by default":

$$\mathcal{B}F \ \equiv\ Not\,\neg F.$$

Readers interested in more details should consult Section A of the Appendix

where we recall the definition of belief theories in *AEB* and precisely establish the equivalence between non-monotonic knowledge bases and belief theories.

While the two notions are equivalent, the notion of a non-monotonic knowledge base seems better suited for the introduction of the class of super logic programs and thus throughout most of the paper we use the language of non-monotonic knowledge bases. However, for the sake of completeness, major results of this paper are also restated in the language of belief theories.

From the equivalence of the two logics (see Proposition A.2) it follows immediately that the following lemma, which will be needed in the sequel, holds.

Proposition 2.3 *For any knowledge base T and for any two formulae F and G of $\mathcal{L}_{Not}$:*

$$T \vdash_{Not} (Not\, F \rightarrow \neg Not\, \neg F)$$
$$T \vdash_{Not} (Not\, \neg(F \rightarrow G) \wedge Not\, \neg F) \rightarrow Not\, \neg G$$
$$If\ \ T \vdash_{Not} F\ \ then\ \ T \vdash_{Not} Not\, \neg F.$$

The first statement says that if a formula F is false by default then its negation $\neg F$ is not (i.e., F is not true by default). The second statement says that if the formulae $F \rightarrow G$ and F hold by default then we can also assume that the formula G holds by default. We recall again that the formula $Not\, \neg F$ has the intuitive reading of "F can be assumed true by default". The last inference rule says that if a formula F is derivable then it is true by default.

2.2 Intended Meaning of Default Negation

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 [49, 38] and [48]):

$Not\, F$ 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 2.4 (Minimal Models [56, 60])
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 [68]). 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., $T \models_{\min} F \equiv CIRC(T; At_{\mathcal{L}}) \models F$.

Example 2.5 Consider the following simple knowledge base T:

$$\rightarrow Car.$$
$$Car \wedge Not\,Broken \rightarrow Runs.$$

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):

$$M_1 = \{Not\,Broken,\ Car,\ Runs,\ \neg Broken\};$$
$$M_2 = \{\neg Not\,Broken,\ Car,\ \neg Runs,\ \neg Broken\}.$$

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

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 databases.

Definition 2.6 (Static Expansions of Knowledge Bases [56, 60, 59]) *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 database obtained by: (i) adding to T 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}.

Remark 2.1 When stated in the language of belief theories, the above fixed-point definition takes the form:

$$T^\diamond = Cn_{AEB}\Big(T \ \cup \{\mathcal{B}F : T^\diamond \models_{\min} F\}\Big).$$

Clearly, for any formula F of $\mathcal{L}_{Not}$ and for any static expansion $T^\diamond$ of T we have:

$$T^\diamond \models Not\, F \quad \text{if} \quad T^\diamond \models_{\min} \neg F, \tag{4}$$

thus a formula F is false by default in a static expansion $T^\diamond$ if its negation is minimally entailed by $T^\diamond$.

Consequently, the definition of static expansions formally *enforces* the intended meaning of default atoms described in the previous subsection. In general, the converse of (4) does not hold because default atoms may be forced in by the database itself. For example, $T = \{a \vee b,\ Not\, a\}$ has a consistent static expansion in which $Not\, a$ holds and yet $T \not\models_{\min} \neg a$. However, in *rational* knowledge bases, i.e., those in which default atoms do not occur positively, the converse is also true. Intuitively, rational databases are characterized by the fact that defaults $Not\, F$ are not explicitly asserted but instead are only implicitly *inferred* by virtue of minimal entailment of the underlying formulae F.

Theorem 2.7 (Semantics of Default Negation [56, 60]) *Let $T^\diamond$ be a static expansion of a* rational *knowledge base T. For any formula*

F of $\mathcal{L}_{Not}$ we have:

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

Remark 2.2 It is also not difficult to prove that for a rational database T and for any two formulae F and G:

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

This explains why, in general, we do not assume the distributive axiom for conjunction:

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

For example, given the fact that we can assume by default that we do not drink and drive (i.e., $T^\diamond \models Not\,(Drink(t) \wedge Drive(t))$ at the same time t, we do not necessarily want to conclude that that we can either assume by default that we don't drink (i.e., $T^\diamond \models Not\,(Drink(t))$ or assume by default that we don't drive (i.e., $T^\diamond \models Not\,(Drive(t))$, at time t. Expressed in the language of beliefs (see Section 2.1), this is equivalent to saying that a belief $\mathcal{B}(F \vee G)$ in a disjunction of formulae F and G does not necessarily imply the belief $\mathcal{B}F$ in F or the belief $\mathcal{B}G$ in G.

Note, however, that one direction of the above equivalence, 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.

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 [7] 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 $\mathcal{L}_{Not}$.

[7] Strictly speaking the operator is only monotone [56, 60] on the lattice of all theories of the form $Cn_{Not}(T \cup \{Not\, F : F \in \mathcal{L}_{Not}\})$. See also Lemma B.2.

Theorem 2.8 (Least Static Expansion [56, 60, 59])
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 unique fixed point $\overline{T} = T^\lambda = \Psi_T(T^\lambda)$, for some ordinal λ.

The above result allows us to establish the following useful characterization of the least static completion of a knowledge base.

Theorem 2.9 (Characterization of Least Static Expansions)
The least static expansion of a knowledge base T coincides with the smallest theory $\widehat{T}$ satisfying the conditions:

(1) $T \subseteq \widehat{T}$;
(2) $\widehat{T} = Cn_{Not}(\widehat{T})$;
(3) if $\widehat{T} \models_{\min} \neg F$ *then* $Not\, F \in \widehat{T}$.

Proof. Clearly the least static expansion $\overline{T}$ of T satisfies the above conditions 1–3 so $\widehat{T} \subseteq \overline{T}$. Moreover, from the conditions 1–3 it follows that $\Psi_T(\widehat{T}) \subseteq \widehat{T}$. Since the operator Ψ_T is monotonic on such theories (see [56, 60]), we conclude that $T^1 = \Psi_T(T) \subseteq \Psi_T(\widehat{T}) \subseteq \widehat{T}$, and, more generally, $T^\alpha \subseteq \widehat{T}$, for any ordinal α. We conclude therefore that $\overline{T} \subseteq \widehat{T}$. □

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 2.10 (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.*

Consequently, like the predicate completion semantics of a logic program P is completely determined by its Clark's completion $comp(P)$, the static semantics of a knowledge base T is fully determined by its static completion $\overline{T}$. It is easy to verify that a knowledge base T either has a *consistent* static completion $\overline{T}$ or it does *not* have any consistent static expansions at all. Moreover, it turns out that static completions of *affirmative* knowledge bases, introduced in Definition 2.1, are always consistent.

Theorem 2.11 (Consistency of Static Completions [56, 60])
Static completions $\overline{T}$ of affirmative knowledge bases T are consistent.

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 database discussed already in Example 2.5:

$$Car.$$
$$Car \wedge Not\,Broken \rightarrow Runs.$$

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 of this stratified program.

Example 2.13 Consider a slightly more complex knowledge base T:

$$Not\,Broken \rightarrow Runs.$$
$$Not\,Fixed \rightarrow Broken.$$

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

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

it follows that $Not\,Fixed \in T^1$ and therefore $Broken \in T^1$. Since,

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

it follows that $Not\,\neg Broken \in T^2$. From Proposition 2.3 we conclude that $\neg Not\,Broken \in T^2$ and therefore $T^2 \models_{\min} \neg Runs$. Accordingly, since:

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

we infer that $Not\,Runs \in T^3$. 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.

Remark 2.3 It follows from Theorem 3.8 that the fixed point is in fact already reached after the first iteration of the default negation operator Ψ_T. In other words, $\overline{T} = T^1$.

3 Static Completions vs. Clark's Completions

It is easy to see that if the knowledge base is *finite* then the construction of its static completion (or the least static expansion) will stop after *countably* many steps. However, a powerful and somewhat surprising result obtained in this section shows that static completions of finite knowledge bases T are in fact obtained by means of a *single iteration* of the default closure operator Ψ_T. This result eliminates the need for multiple iterations in the computation of static completions and allows us to replace the fixed-point definition of static completions by the equivalent explicit definition given by $\overline{T} = \Psi_T(T)$. By clarifying the notion of a static completion of a finite database it also provides the foundation for the remaining results obtained in the paper.

The second, closely related result establishes a very interesting and somewhat intriguing relationship between static completions $\overline{T}$ and Clark's completions $comp(T)$ of finite knowledge bases. It shows that the static completion $\overline{T}$ of a knowledge base T coincides with $Cn_{Not}\Big(T \cup \{Not\, \neg F : F \in comp(T)\}\Big)$, i.e., with the set of formulae derivable from the knowledge base T and the formulae $\{Not\, \neg F : F \in comp(T)\}$. Even though this result only applies to the so called residual databases (see below) we do not loose generality, as we will show that every finite knowledge base can be transformed into a finite residual knowledge base with equivalent minimal models. Since Clark's completion $comp(T)$ is easily computable, this result reduces reasoning under the static semantics to theorem-proving in the underlying modal logic (this can be done either by hand or by using an automated theorem prover).

The proof of these two powerful results is based on the idea of adding to a knowledge base T the set of formulae which ensure that models "seen" through the default negation operator *Not* are in fact minimal. As we will soon see, this task can be accomplished by a suitable generalization of Clark's completion but it only works for a restricted class of knowledge bases, namely those whose clauses do not have any objective premises. Such *residual databases* were previously introduced and investigated in the class of logic programs [16, 17, 26, 27, 10, 7]. Here we give a slightly more general definition.

Definition 3.1 (Residual Knowledge Bases [7])
By a residual knowledge base *we mean an arbitrary non-monotonic knowledge base whose clauses do not contain any objective (positive) premises, i.e., a (possibly infinite) set of arbitrary clauses of the form:*

$$Not\, G_1 \wedge \ldots \wedge Not\, G_k \rightarrow A_1 \vee \ldots \vee A_l \vee Not\, F_1 \vee \ldots \vee Not\, F_n$$

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

Whenever convenient, clauses of the form:

$$Not\, G_1 \wedge \ldots \wedge Not\, G_k \rightarrow Not\, F_1 \vee \ldots \vee Not\, F_n$$

i.e., clauses without any objective atoms in their heads, will be considered as clauses with a single objective atom $false$ in their head, i.e, clauses of the form:

$$Not\, G_1 \wedge \ldots \wedge Not\, G_k \rightarrow false \vee Not\, F_1 \vee \ldots \vee Not\, F_n.$$

Now we describe a natural extension of the notion of *Clark's completion* $comp(T)$. Clark's completion was initially introduced in [20] for the class of normal logic programs. Subsequently, its generalizations to disjunctive programs without positive premises were studied in [7]. Here we extend it to the class of all residual knowledge bases. As usual, the goal is to ensure that an objective atom A is true only if it really has to be true, i.e., if there is a rule with A in the head, in which the body is true and all the other head literals are false.

Definition 3.2 (Extended Clark's Completion)
Given a finite residual knowledge base T *and an objective atom* A *from* $\mathcal{L}$ *(including the falsity atom* $false$*) we define the* extended Clark's completion $comp(A, T)$ *of the atom* A *in* T *to be the formula:*

$$A \leftrightarrow \bigvee_{1 \leq m \leq s} (Not\, G_{1,m} \wedge \ldots \wedge Not\, G_{k_m,m} \wedge \neg A_{1,m} \wedge \ldots \wedge \neg A_{l_m,m} \wedge \neg Not\, F_{1,m} \wedge \ldots \wedge \neg Not\, F_{n_m,m}),$$

where

$$Not\, G_{1,m} \wedge \ldots \wedge Not\, G_{k_m,m} \rightarrow A \vee A_{1,m} \vee \ldots \vee A_{l_m,m} \vee Not\, F_{1,m} \vee \ldots \vee Not\, F_{n_m,m},$$

for $1 \leq m \leq s$*, are all clauses in* T *containing the atom* A *in their heads.*

By Clark's completion *comp*(T) *of a knowledge base* T *we mean the union of all completions* $comp(A, T)$ *of* A *in* T*, for all objective atoms* A *in the language* $\mathcal{L}$ *(including the falsity atom* $false$*).*

The following interesting result shows that if T is a residual knowledge base, then the models of Clark's completion *comp*(T) are precisely the minimal models of T itself. In other words, Clark's completion precisely describes the minimal model semantics of a residual knowledge base T.

Proposition 3.3 *Let* T *be a finite residual knowledge base. An interpretation* M *of the language* $\mathcal{L}_{Not}$ *is a minimal model of* T *if and only if* M *is a model of Clark's completion* $comp(T)$ *of* T*.*

Proof. Contained in the Appendix B.1. □

It is worth pointing out that one of the weaknesses of the original version of Clark's completion proposed in [20] was the fact that it was applied not just to residual but rather to *arbitrary* normal programs. As a result, Clark's original completion did *not* enforce minimal model semantics, e.g., a tautlogy like $p \leftarrow p$ made the completion axiom for p useless. Another problem with the original definition of Clark's completion was that it did not distinguish between logical negation $\neg p$ and default negation $Not\, p$. As a result, Clark's completion could be inconsistent when rules like $p \leftarrow Not\, p$ were present.

Remark 3.1 It is interesting to note that when T represents a disjunctive program and when we replace $Not\, C$ by $\neg C$, we in fact obtain stable models of T. We used this fact in [7] to compute the stable semantics of disjunctive programs.

Although the above result, as well as the definition of Clark's completion, applies only to the class of residual knowledge bases, it turns out that *every* finite knowledge base T can be transformed into a finite residual knowledge base T_{res} so that their sets of minimal models coincide.

Proposition 3.4 *Every finite knowledge base* T *can be transformed into a finite residual knowledge base* T_{res}*, called the* residuum *of* T*, so that an interpretation* M *of the language* $\mathcal{L}_{Not}$ *is a minimal model of* T *if and only if* M *is a minimal model of* T_{res}*.*

Proof. Contained in the Appendix B.2. □

In fact, just two elementary database transformations, namely, "unfolding" (or partial evaluation, GPPE [10, 61]) and "elimination of tautologies", are

sufficient to obtain such a residual program. For a detailed definition of these transformations the reader is referred to [7, 8, 10, 9]. Here we describe them very briefly.

The "elimination of tautologies" allows us to remove a rule if it contains the same atom in both, head and body. Such rules are always trivially satisfied and therefore useless. The transformation of "unfolding" - also called $GPPE$ – is defined as follows. Suppose that B is an objective atom and let $B \vee \mathcal{A}_i \leftarrow \mathcal{R}_i$, $i = 1, \ldots, n$, be all rules about B, i.e., rules containing B in their head. The application of $GPPE$ to the rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$, that contains B in its body, results in a new knowledge base obtained by deleting the rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$ and adding to the knowledge base the new rules $\mathcal{A} \vee \mathcal{A}_i \leftarrow \mathcal{R} \wedge \mathcal{R}_i$.

We now give a simple example. Whenever convenient, we use the traditional inverse notation for clauses. Consider the following knowledge base T:

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

The first two clauses already have the required form. In the last clause, we unfold the body literal q and obtain the residual program T_{res}:

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

One can easily see that this procedure represents a form of hyper-resolution.

As an immediate consequence of Propositions 3.3 and 3.4 we obtain the following important theorem which says that for any finite knowledge base T, Clark's completion of its residuum T_{res} precisely describes the minimal model semantics of T.

Theorem 3.5 *Let T be any finite knowledge base. An interpretation M of the language $\mathcal{L}_{Not}$ is a minimal model of T if and only if M is a model of Clark's completion $comp(T_{res})$ of the residuum T_{res} of T.*

Finally, let us denote by $\mathit{Not}\, \neg comp(T_{res})$ the set $\{\mathit{Not}\, \neg F : F \in comp(T_{res})\}$. Intuitively, augmenting a given knowledge base S with $\mathit{Not}\, \neg comp(T_{res})$ en-

sures that all formulae that belong to Clark's completion $comp(T_{res})$ are assumed to be true by default (in S). We will need the following lemma.

Lemma 3.6 *Let T be an arbitrary finite knowledge base and let*

$$\widehat{T} = Cn_{Not}\Big(T \cup Not\, \neg comp(T_{res})\Big).$$

If a formula F is false in all minimal models of $\widehat{T}$ then $Not\, F$ is already contained in $\widehat{T}$. In other words, $\Psi_T(\widehat{T}) \subseteq \widehat{T}$.

Proof. Contained in the Appendix B.3. □

Now we are ready to state the first of the two fundamental results obtained in this section.

Theorem 3.7 (Static Completion vs. Clark's Completion)
The static completion $\overline{T}$ of a finite knowledge base T can be equivalently defined as:

$$\overline{T} = Cn_{Not}\Big(T \cup Not\, \neg comp(T_{res})\Big).$$

Proof. Contained in the Appendix B.4. □

The above result states that the static completion $\overline{T}$ of T is obtained by augmenting T with the set $Not\, \neg comp(T_{res})$ thus ensuring that all formulae that belong to Clark's completion $comp(T_{res})$ of T_{res} are assumed to be true by default. It establishes an interesting and somewhat intriguing relationship between static completions and Clark's completions of finite knowledge bases. It also reduces reasoning under the static semantics (i.e., the computation of static completions $\overline{T}$) to the easily accomplished computation of Clark's completion $comp(T_{res})$ together with theorem-proving in the underlying modal logic (which can be done either by hand or using an automated theorem prover).

Remark 3.2 When stated in the language of beliefs, the above result takes the form:

$$\overline{T} = Cn_{AEB}\Big(T \cup \mathcal{B}(comp(T_{res}))\Big).$$

It states that the static completion $\overline{T}$ of T is obtained by augmenting T with beliefs $\mathcal{B}F$ in all formulae F which belong to Clark's completion of T_{res}.

As a consequence of the proof of the above result we immediately obtain the second of the two fundamental theorems obtained in this section, namely a powerful and somewhat surprising result stating that static completion $\overline{T}$ of an arbitrary finite knowledge base T is always obtained by a single iteration of the default closure operator $\Psi_T(T)$.

Theorem 3.8 (Explicit Characterization of Static Completions) *The static completion $\overline{T}$ of finite knowledge base T is always obtained by a single iteration of the default closure operator $\Psi_T(T)$. More precisely:*

$$\overline{T} = \Psi_T(T).$$

In other words,

$$\overline{T} = Cn_{Not}\Big(T \cup \{Not\, F : T \models_{min} \neg F\}\Big).$$

The above theorem states that the static completion $\overline{T}$ of T is obtained by augmenting T with negation by default $Not\, F$ of all formulae F which are false in all minimal models of the theory T itself. It eliminates therefore the need for multiple iterations in the computation of static completions and allows us to replace the fixed-point definition of static completions by the equivalent explicit definition. In addition to Theorems 2.8, 2.9 and 3.7, this is the fourth characterization of static completions presented in this paper.

Needless to say, the existence of an equivalent *non-fixed point* definition of static completions significantly simplifies this notion and the underlying theory. It also provides the foundation for the remaining results obtained in the paper.

Remark 3.3 When stated in the language of beliefs, the above result takes the form:

$$\overline{T} = Cn_{AEB}\Big(T \cup \{\mathcal{B}F : T \models_{min} F\}\Big).$$

It states that the static completion $\overline{T}$ of T is obtained by augmenting T with beliefs $\mathcal{B}F$ in all formulae F which are minimally entailed by the theory T itself.

Example 3.9 Consider the knowledge base T from Example 2.13:

$$Not\, Broken \rightarrow Runs.$$
$$Not\, Fixed \rightarrow Broken.$$

We already noted that $T \models_{min} \neg Fixed$ and thus

$$Not\, Fixed \in \overline{T} = Cn_{Not}\Big(T \cup \{Not\, F : T \models_{min} \neg F\}\Big).$$

This implies that $\overline{T}$ contains also $Broken$. By Proposition 2.3, $Not\, \neg Broken$ belongs to the static completion and so does $\neg Not\, Broken$. From the same Lemma it follows that $Not\, Not\, Broken$ belongs to the completion.

Moreover, it is easy to verify that $T \models_{min} Not\, Broken \leftrightarrow Runs$ (in fact, the formula $Not\, Broken \leftrightarrow Runs$ belongs to Clark's completion of T) and therefore $T \models_{min} \neg Not\, Broken \leftrightarrow \neg Runs$. It follows that $\overline{T}$ contains

$$Not\, \neg(\, \neg Not\, Broken \leftrightarrow \neg Runs\,),$$

which can also be written in the form

$$Not\, (\, (\neg Not\, Broken \wedge Runs) \vee (Not\, Broken \wedge \neg Runs)),$$

and thus the distributive axiom DA can be applied:

$$Not\, (\, \neg(\neg Not\, Broken \rightarrow \neg Runs)) \wedge Not\, (\, \neg(\neg Runs \rightarrow \neg Not\, Broken)).$$

Now we can apply the second formula of Proposition 2.3 to the first conjunct (we already know that $Not\, Not\, Broken$ belongs to the completion). Therefore $\overline{T}$ also contains $Not\, Runs$.

We again conclude that the static semantics of T derives $Not\, Fixed$, $Broken$ and $Not\, Runs$. Observe that while the above reasoning utilizes only one iteration of the default closure operator, it is slightly more complex than the one used in Example 2.13 where several iterations were performed.

Remark 3.4 Our work is closely related to the paper [68] where the authors define the *well-founded circumscriptive* semantics for disjunctive programs (without nested default negations). They introduced the concept of minimal model entailment with *fixed* default atoms and defined their semantics as a *single step application* of circumscription.

3.1 Adding Strong Negation to Knowledge Bases

As pointed out in [36, 2, 3, 1] in addition to default negation, $Not\, F$, nonmonotonic reasoning requires another type of negation $\neg F$ which is similar to

classical negation $\neg F$ but does not satisfy the law of the excluded middle (see [1] for more information).

Strong negation $\text{–}F$ can be easily added to knowledge databases by:

- Extending the original objective language $\mathcal{L}$ by adding to it new *objective* propositional symbols $\text{–}A$, called *strong negation atoms*, for all $A \in At_{\mathcal{L}}$; as a result we obtain a new objective language $\widehat{\mathcal{L}}$ and the new extended modal language $\widehat{\mathcal{L}}_{Not}$;
- Adding to the database T the following *strong negation clause*, for any strong negation atom $\text{–}A$ that appears in T:

  $$(S_A) \qquad A \wedge \text{–}A \rightarrow false,$$

  which says that A and $\text{–}A$ cannot be both true and thus ensures the intended meaning of $\text{–}A$ as "the opposite of A".

For example, a proposition A may describe the property of being *"qualified"* while the proposition $\text{–}A$ describes the property of being *"unqualified"*. The strong negation axiom states that a person cannot be *both* qualified and unqualified. We do not assume, however, that everybody is already known to be either qualified or unqualified.

Since the addition of strong negation clauses simply results in a new knowledge base, all the results stated so far apply as well to knowledge databases with strong negation.

4 Super Logic Programs

We now introduce the class of *super logic programs*, briefly called *super programs*, as a subclass of the class of all non-monotonic knowledge bases. This powerful class of databases includes all (monotonic) *propositional theories*, all *disjunctive logic programs* and all *extended logic programs* with strong negation (see Section 3.1).

Definition 4.1 (Super Logic Programs)
A Super Logic 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[8] *objective formulae.*

[8] I.e., a formula that can be represented as a disjunction of conjunctions of atoms.

Observe that since any formula $F \leftarrow G$ can be written as a finite conjunction of disjunctions of literals, we can always replace a super-clause by a finite set of clauses whose heads are disjunctions $A_1 \vee \ldots \vee A_k$ of atoms and whose positive premises are conjunctions $B_1 \wedge \ldots \wedge B_m$ of objective atoms. Finally, since the operator *Not* obeys the *distributive law for disjunction* (DA), the default atom $Not\, H$ 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. This allows us to establish the following useful fact.

Proposition 4.2 *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.

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:

- It allows constraints, i.e., headless rules. In particular it allows the addition of *strong* negation to such programs, as shown in Section 3.1, by assuming the *strong negation axioms* $\leftarrow A \wedge \neg A$, which themselves are program rules (rather than meta-level constraints);
- 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 $Not(man \wedge \neg human)$, which can be interpreted as stating that we can assume by default that every man is a human (note that $\neg human$ represents strong negation of $human$).

Although the class of super programs is quite broad, it is significantly less general than the class of all non-monotonic knowledge bases because (i) it *does not allow any nesting of default negation* and (ii) it consists only of *rational* knowledge bases. As a result, static completions of super programs admit much simpler and computationally more feasible characterizations. They allow the computation of their relevant parts without the need to involve arbitrarily deeply nested default negations, which are inherently *infinite* even for finite programs.

In the following two sections we provide two such powerful characterizations of static completions of super programs and one of them is later used to

implement a query-answering interpreter for super programs. First, however, we discuss general properties of super programs and give a simple example.

The class of super programs has a number of useful properties. Since super programs are rational knowledge bases, from Theorem 2.7 it immediately follows that for any formula F the static semantics (or completion) $\overline{P}$ of a super program P derives $Not\, F$ if and only if $\neg F$ is minimally entailed by $\overline{P}$.

Theorem 4.3 *Let P be any super logic program and let $\overline{P}$ be its static completion. For any formula F, $\overline{P} \models Not\, F \;\equiv\; \overline{P} \models_{min} \neg F$.*

The next result summarizes basic properties of super programs. It is an immediate consequence of Proposition A.2 and the results established in [56, 60].

Theorem 4.4 (Basic Properties of Super Programs)

- *Every affirmative super program has a consistent static completion. In particular, this applies to all disjunctive logic programs.*
- *For normal logic programs, static semantics coincides with the well-founded semantics [67]. More precisely, there is a one-to-one correspondence between consistent static expansions of a normal program and its partial stable models [58]. Under this correspondence, (total) stable models [35] correspond to those consistent static expansions which satisfy the axiom $Not\, A \vee Not\, \neg A$, for every objective atom A.*
- *For positive disjunctive logic programs, static semantics coincides with the minimal model semantics.*

Example 4.5 Consider a simple super-program P:

$$
\begin{aligned}
VisitEurope \vee VisitAustralia \leftarrow .& \\
Happy &\leftarrow VisitEurope \vee VisitAustralia. \\
Bankrupt &\leftarrow VisitEurope \wedge VisitAustralia. \\
Prudent &\leftarrow Not\,(VisitEurope \wedge VisitAustralia). \\
Disappointed &\leftarrow Not\,(VisitEurope \vee VisitAustralia).
\end{aligned}
$$

Given the usual exclusive interpretation of disjunctions we would most likely conclude that we will either visit Australia or Europe but not both and thus we expect the answer to the query $VisitEurope \vee VisitAustralia$ to be positive while the answer to the query $VisitEurope \wedge VisitAustralia$ to 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 $Disappointed$.

Observe that the query $Not\,(VisitEurope \wedge VisitAustralia)$ 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 $Not\, VisitEurope \wedge Not\, VisitAustralia$

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 $\neg VisitEurope \vee \neg VisitAustralia$ or $\neg(VisitEurope \wedge VisitAustralia)$ holds in all minimal models of the program P and therefore $Not\,(VisitEurope \wedge VisitAustralia)$ 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 $VisitEurope \vee VisitAustralia$ is true, we infer from Proposition 2.3 that $Not\,\neg(VisitEurope \vee VisitAustralia)$ and thus also $\neg Not\,(VisitEurope \vee VisitAustralia)$ holds. But then $\neg Disappointed$ is minimally entailed which implies that $Not\,Disappointed$ belongs to the completion.

4.1 Static Semantics vs. Disjunctive Stable Semantics

In this section we briefly relate the static semantics of super programs to the *disjunctive stable semantics*, the most popular semantics proposed earlier for disjunctive programs [58, 36]. For additional information and discussion involving other semantics of disjunctive logic programs and their relationship to static semantics, the reader is referred to [46, 59, 57, 11].

As stated in Theorem 4.4, static semantics extends the stable semantics of normal, non-disjunctive programs in the sense that stable models of a normal program P are a *special case* of static expansions of P. As a result, the full power of stable semantics can be captured by means of static expansions. However, even when restricted to normal programs, the stable semantics suffers from some important drawbacks compared to the static semantics. The most important of these drawbacks is the fact that stable semantics is not defined for some very natural and intuitive programs. To illustrate this point, let us consider the following program:

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

It appears that the first three rules describe rather obvious and natural mutual relationships between propositions $tired, work$ and $sleep$, without providing

sufficient information to determine their individual truth or falsity. On the other hand, regardless of the status of propositions $tired, work$ and $sleep$, the proposition $paid$ should be considered true and thus $angry$, by negation as failure, should be false.

This leads to the least static expansion $P^{\diamond}$ of the program P which implies $paid$ and $Not\ angry$ and, of course, contains also the three first program rules mutually relating $tired$, $work$ and $sleep$. As a result, if we later learn, e.g., that $work$ is actually true then we will immediately conclude that $sleep$ is false and $tired$ is true, but our beliefs about $paid$ and $angry$ will remain unchanged. It is worth noting, that Prolog would return the same answers and so would Fitting and Kunen's extension of Clark's semantics.

However, it is easy to see that the stable semantics, applied to this *normal* program is contradictory and thus undefined. As a result, we are unable to establish the well-defined facts about the truth of predicates $paid$ and $angry$ as well as the useful information about the mutual relationships existing between the predicates $tired$, $work$ and $sleep$.

Before comparing the behavior of the two semantics in broader classes of programs, one has to point out that the class of super programs is strictly broader than the class of disjunctive logic programs. Consequently, when comparing the two semantics in broader classes of programs, we will necessarily have to limit our discussion to the class of disjunctive logic programs.

The above example illustrated the fact that stable semantics may be contradictory even for quite simple and natural normal logic programs. However, as stated in Theorem 4.4, the static semantics of normal programs is equivalent to the *partial* stable semantics. As a result, partial stable semantics can be used instead of stable semantics to provide the meaning of programs that are contradictory under the standard stable semantics.

However, in case of disjunctive programs, the situation is considerably worse. While static semantics is well-defined and consistent for all disjunctive programs, and, as shown in the previous sections, has a very nice fixed-point characterizations, it is easy to produce simple and intuitive disjunctive programs for which not only the disjunctive stable semantics but also the disjunctive *partial* stable semantics is inconsistent.

To illustrate this point, it suffices to add to the previous example a simple disjunctive clause saying that we are either working or sleeping or are just tired:

$$work \vee sleep \vee tired.$$

The resulting *disjunctive program* still has a well defined static semantics in which one of the predicates *tired*, *work* or *sleep* holds true. On the other hand, both the stable and partial stable semantics of this program are contradictory.

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 ([8, 22]), and it is not *compositional* (or modular), a highly desirable property for software engineering and KR [14, 19, 65]. The static semantics has both of these nice structural properties.

5 Fixed Point Characterization of Static Completions

Due to Proposition 4.2, 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. In fact, default negation can be only applied to conjunctions of objective atoms. Note, however, that we allow also the "empty conjunction", so $Not\,(true)$ is contained in $\mathcal{L}^*_{Not}$. Note also that we can treat E essentially 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.

Clearly, 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 important 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 5.1 (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 unique fixed point $P^{n_0} = P^{n_0+1}$, 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 E. □

The above theorem represents a considerable simplification over the original characterization of static completions given in Theorem 2.8. First of all, instead of using the modal consequence operator Cn_{Not} the definition uses the *standard* propositional consequence operator Cn. 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 can no longer expect the fixed point to be reached in just one step, as in Theorem 3.8.

6 Model-Theoretic Characterization of Static Completions

In this section, we complement Theorem 5.1 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 briefly in the next section [9].

Definition 6.1 (Reduced Interpretations)

(1) Let $\mathcal{OBJ}$ be the set of all propositional valuations of the objective atoms $At_{\mathcal{L}}$.

[9] Throughout this section whenever we mention a sequence $p_1, \ldots, p_n$ we allow n to be 0 as well.

(2) Let $\mathcal{NOT}$ be the set of all propositional valuations of the default atoms $\{Not\,(p_1 \wedge \cdots \wedge p_n) \,:\, p_i \in At_{\mathcal{L}}\}$, which interpret $Not\,(true)$ (in case $n = 0$) as false.

(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 $\mathcal{I}$.

Since an 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 objective interpretation I'_{obj} 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 in this section 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 C). The worlds in such a Kripke structure are marked with objective interpretations, 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 objective models. So given a set $\mathcal{O}$ of minimal objective models, an interpretation I_{not} of the default atoms is possible iff there is some subset $\mathcal{O}' \subseteq \mathcal{O}$ (namely the objective 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 $I_{obj} \in \mathcal{O}'$. But the minimal objective models $\mathcal{O}$ also depend conversely 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 default interpretations, we also get less minimal objective models. This obviously results in a fixed-point computation, formalized by the following operators:

Definition 6.2 (Possible Interpretation of Default Atoms)
Let P be a super program.

(1) The operator $\Omega_P\colon 2^{\mathcal{NOT}} \to 2^{\mathcal{OBJ}}$ yields minimal objective models given possible interpretations of the default atoms. For every $\mathcal{N} \subseteq \mathcal{NOT}$, let

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

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

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

(3) The operator $\Pi_P\colon 2^{\mathcal{OBJ}} \to 2^{\mathcal{NOT}}$ yields possible interpretations of the default atoms, given minimal objective models. For every $\mathcal{O} \subseteq \mathcal{OBJ}$, let

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

(4) The operator $\Theta_P\colon 2^{\mathcal{NOT}} \to 2^{\mathcal{NOT}}$ 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 6.3 (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 D. □

Example 6.4 Let us consider the following logic program P:

$$p \vee q \leftarrow \mathit{Not}\ r.$$
$$q \leftarrow \mathit{Not}\ q.$$
$$r \leftarrow q.$$

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 superflous 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 possiblities 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 an objective 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. In the Appendix, Definition D.1, we define a Kripke structure which allows us conversely 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 C.6 in the Appendix).

7 Implementation of the Model-Theoretic Characterization

We implemented a *query-answering interpreter for the static semantics* in the class of super programs based on the above model-theoretic characterization [10] . Let us explain in a little more detail how our prototype implementation works. First, because of the large number of possible default atoms, it seems impossible to construct explicitly the set $\mathcal{NOT}$ of all valuations — it would have 2^{2^n} elements, where n is the number of objective atoms. However, it suffices to consider only those default atoms which occur explicitly in the program, other default atoms have no influence on the minimal models and we can always extend a valuation of these occurring default atoms consistently to a valuation of all default atoms.

Furthermore, it is possible to reduce the number of default atoms occurring in the program by evaluating the simple cases directly. For example, when it is possible to derive the disjunctive fact $\mathit{VisitEurope} \vee \mathit{VisitAustralia}$ (in the example it is given explicitely), we can delete any rule like

$$\mathit{Disappointed} \leftarrow \mathit{Not}\,(\mathit{VisitEurope}) \wedge \mathit{Not}\,(\mathit{VisitAustralia}),$$

because we know that its body can never be true (this is a special case of the "negative reduction" transformation used in [10]). In this example, deleting the rule also eliminates two default atoms.

Accordingly, the first step of our algorithm is to compute the residual program and to apply the reduction operators defined in [10]. This transformation preserves the minimal models, and often significantly reduces the number of

[10] See `http://www-db.informatik.uni-hannover.de/software/static/`

default atoms we have to consider (for instance, no default atoms remain if the input program is stratified and non-disjunctive).

Subsequently, we apply the definition of the operators Ω_P and Π_P from Section 6 quite literally. However, two points are worth mentioning. In order to compute the minimal models for the Ω_P-operator, we use the extension of Clark's completion. Given an interpretation of the default atoms occurring in the residual program, we evaluate the rule bodies, so we get a set of positive disjunctions $p_1 \vee \cdots \vee p_n$. We compute the completion by treating $p_1 \vee \cdots \vee p_n$ like $p_i \leftarrow \neg p_1 \wedge \cdots \wedge \neg p_{i-1} \wedge \neg p_{i+1} \wedge \cdots \wedge \neg p_n$. Then we use any model generation algorithm. Of course, other algorithms for generating minimal models can also be used, for instance the SATCHMO-like algorithm proposed in [15]. Other very efficient algorithms are described in [54, 53] and they can also be integrated into our framework.

Next, it would be very inefficient to consider all subsets $\mathcal{O}' \subseteq \mathcal{O}$, as required in the definition of the Π_P-operator. However, in order to check whether a given interpretation I_{not} of the default atoms is selected by Θ_P, only the maximal $\mathcal{O}'$ is of interest. We construct it as the set consisting of all $I_{obj} \in \mathcal{O}$ satisfying, for every default atom, the condition:

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

Obviously, the inclusion of other I_{obj} into $\mathcal{O}'$ would immediately destroy the required property. Next, we check wether the so constructed $\mathcal{O}'$ is non-empty. Finally, we check that for every default atom $\mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$ which is false in I_{not}, there is an $I_{obj} \in \mathcal{O}'$ with $I_{obj} \not\models \neg(p_1 \wedge \cdots \wedge p_n)$.

The program outputs the fixed point $\mathcal{N}^\diamond$ of Θ_P and all minimal reduced models, whose default part is contained in $\mathcal{N}^\diamond$. It then waits for queries. Queries to super programs are answered entirely on the basis of these minimal models.

8 Conclusion

In this concluding section we recall the main contributions of the work presented in this article. We then point to related work and finally we discuss the usefulness of our results by pointing to some specific problems where we expect our techniques to matter.

8.1 What did we achieve?

In the first part of the paper we proved two powerful characterizations of the static semantics of finite non-monotonic knowledge bases (or belief theories). The main and fundamental result shows that static completions of such knowledge bases can in fact be obtained by means of a *single iteration* of the default closure operator. This result eliminates the need for multiple iterations in the computation of static completions and allows us to replace the fixed-point definition of static completions by the equivalent explicit definition. The second, closely related result, establishes a very interesting and somewhat intriguing relationship between static completions and Clark's completions of non-monotonic knowledge bases (or belief theories). Since Clark's completions are easily computable, it effectively reduces reasoning under the static semantics to theorem-proving in the underlying modal logic. By clarifying the notion of a static completion of finite knowledge bases, the two powerful characterizations provide the foundation for the remaining results obtained in the paper.

In the second part of the paper, 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 additional 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 important 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 via FTP and WWW. We noted that while no such computational mechanism can be efficient, due to the inherent NP-completeness 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 [56, 60]. *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 4.4 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 [57]. 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 [59].

Even though most of the presented results, as well as the current implementation, are technically limited to finite propositional programs, they can be easily extended to range-restricted disjunctive Datalog programs, i.e., to programs in which every variable in the rule appears also in a positive body literal.

8.2 Related work

The first attempt to generalize the syntax of logic programs is due to Lloyd and Topor [43, 44, 45]. 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 logically equivalent transformations.

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 would have come up with a special case of the static semantics (restricted to formulae of the form $A \leftarrow W$).

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 [32, 33]). It is a knowledge representation system, which offers front-ends to several advanced KR formalisms. The kernel language, which extends dis-

junctive logic programming by true negation and integrity constraints, allows for representing complex knowledge base problems in a highly declarative fashion [33]. They also consider modular evaluation techniques along with linguistic extensions to deal also with quantitative information [18].

Recently, Lifschitz et. al. introduced nested expressions in the bodies of rules ([41]). This approach allows a larger class of formulae to appear in bodies of rules, but it is restricted to the semantics of answer sets, whereas in our framework, we do not need to restrict ourselves to any particular semantics.

In [39] nested rules are also allowed in rule heads. The authors show that they can capture the full second level of the polynomial hierarchy in terms of expressive power.

There are also successful approaches to generalize logic programming languages by using intuitionistic, linear, or higher order logics (e.g. [40, 52]). These extensions seem orthogonal to our treatment of negation in the context of arbitrary propositional formulas.

8.3 Why is it useful?

We already illustrated in the introduction the need for expressing disjunctive information and thus for extending normal logic programs to disjunctive programs. Besides the added expressivity (which goes up to the full second level of the polynomial hierarchy (already for positive programs) we emphasize the following reasons:

(1) Recent research has shown that even problems that can be formulated without disjunctions, sometimes can be naturally transformed into disjunctive programs. This can be done in such a way that solutions of the original problem correspond to minimal models of the disjunctive program. In the course of the DisLoP project ([5]) this has been demonstrated in a diagnosis setup. 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 ([6]).
(2) The view update problem in deductive and relational databases can also be transformed into a problem of computing minimal models of disjunctive programs ([4]).
(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 [62, 66] 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 [63, 64] a particular real-world problem is considered, namely, 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 [63, 64].

Why are we not satisfied with the disjunctive version of the stable semantics? We see the following reasons:

(1) Our proposed semantics can be seen as a more cautious version of a *disjunctive* well-founded semantics. In fact, recently this has been shown formally in [11, 9]: a suitable restriction of our framework to a certain sublanguage coincides with the D-WFS semantics of Brass and Dix [8, 10].

(2) As already mentioned at the end of Subsection 4.1, besides the inconsistency of the stable semantics, it is also neither *goal-oriented* nor *modular* [22, 13] (recently called *compositional* by Bry.). Our static semantics has these properties and thus methods for query optimization based on the dependency graph below that query can be applied and may reduce the overall computation.

(3) Moreover, even if one intends to use stable semantics, it may still make sense to use static semantics for *preprocessing*. This means that even inference engines designed for disjunctive stable semantics can utilize our results. This is exactly analogous to the non-disjunctive case. For example, the high performance of the stable model engine called "smodels" [54] is due to the fact that it first computes WFS and in the process significantly reduces the program. The following property explains the reasons: if M is a set of atoms such that $\mathrm{WFS}(P) \models M$, then $\mathrm{STABLE}(P + M) = \mathrm{STABLE}(P)$. Now the same property holds for the disjunctive case where WFS is replaced by D-WFS from Brass/Dix, which is an instance of the static semantics. Thus the static semantics is sound even for computing stable models.

(4) Finally, we can link our framework with tableau based procedures which seem to be very promising from the efficiency standpoint (they run in polynomial space). We refer to [11] for more information.

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 two anonymous reviewers for their detailed and very insightful comments.

References

[1] Jose Alferes, Luis M. Pereira, and Teodor C. Przymusinski. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20:107–142, 1998.

[2] Jose Julio Alferes and Luiz Moniz Pereira. On Logic Program Semantics with Two Kinds of Negation. In K. R. Apt, editor, *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, pages 574–589, Cambridge, Mass., November 1992. MIT Press.

[3] Jose Julio Alferes and Luiz Moniz Pereira, editors. *Reasoning with Logic Programming*, LNAI 1111, Berlin, 1996. Springer.

[4] Chandrabose Aravindan and Peter Baumgartner. A Rational and Efficient Algorithm for View Deletion in Databases. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the 1997 International Symposium*, pages 165–180. MIT Press, 1997.

[5] Chandrabose Aravindan, Jürgen Dix, and Ilkka Niemelä. Dislop: A research project on disjunctive logic programming. *AI Communications*, 10:151–165, 1997.

[6] Peter Baumgartner, Peter Fröhlich, Ulrich Furbach, and Wolfgang Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. In Martha Pollack, editor, *15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 460–465, Nagoya, 1997.

[7] Stefan Brass and Jürgen Dix. A General Approach to Bottom-Up Computation of Disjunctive Semantics. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, pages 127–155. Springer, Berlin, 1995.

[8] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997. (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.).

[9] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Well-founded Semantics: Confluent Calculi and Iterated GCWA. *Journal of Automated Reasoning*, 20(1):143–165, 1998. (Extended abstract appeared

in: Characterizing D-WFS: Confluence and Iterated GCWA. *Logics in Artificial Intelligence, JELIA '96*, pages 268–283, 1996. Springer, LNCS 1126.).

[10] Stefan Brass and Jürgen Dix. Semantics of Disjunctive Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, accepted for publication, 1998. (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.).

[11] Stefan Brass, Jürgen Dix, Ilkka Niemelä, and Teodor. C. Przymusinski. A Comparison of the Static and the Disjunctive Well-founded Semantics and its Implementation. In A. G. Cohn, L. K. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR '98)*, pages 74–85. San Francisco, CA, Morgan Kaufmann, May 1998.

[12] Stefan Brass and Udo W. Lipeck. Bottom-up query evaluation with partially ordered defaults. In Stefano Ceri, Katsumi Tanaka, and Shalom Tsur, editors, *Deductive and Object-Oriented Databases, Third Int. Conf., (DOOD'93)*, number 760 in LNCS, pages 253–266, Berlin, 1993. Springer.

[13] Gerhard Brewka and Jürgen Dix. Knowledge representation with extended logic programs. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, 2nd Edition, Volume 6, Methodologies*, chapter 6. Reidel Publ., 1998.

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

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

[16] François Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89)*, pages 34–50, 1989.

[17] François Bry. Negation in logic programming: A formalization in constructive logic. In Dimitris Karagiannis, editor, *Information Systems and Artificial Intelligence: Integration Aspects*, pages 30–46, Berlin, 1990. Springer.

[18] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Strong and Weak Constraints in Disjunctive Datalog. In *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR '97)*, pages 2–17, Dagstuhl, Germany, July 1997.

[19] M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming.

Journal of Logic Programming, 20:443–502, 1994.

[20] Keith L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data-Bases*, pages 293–322. Plenum, New York, 1978.

[21] J. Dix, L. Pereira, and T. Przymusinski, editors. *Logic Programming and Knowledge Representation*, LNAI, Berlin, 1998. Springer.

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

[23] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, pages 241–327. DeGruyter, 1995.

[24] Jürgen Dix, Luis M. Pereira, and Teodor C. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 927, Berlin, 1995. Springer Verlag. Proceedings of the Workshop at the Eleventh International Logic Programming Conference, ICLP'94, Santa Margherita Ligure, Italy, June 1994.

[25] Jürgen Dix, Luis M. Pereira, and Teodor C. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 1216, Berlin, 1997. Springer Verlag. Proceedings of the Workshop at the International Logic Programming Conference, JICSLP'96, Bonn, Germany, September 1996.

[26] P. M. Dung and K. Kanchansut. A fixpoint approach to declarative semantics of logic programs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of North American Conference Cleveland,Ohio, USA*. MIT Press, October 1989.

[27] P. M. Dung and K. Kanchansut. A natural semantics for logic programs with negation. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the 9th Conference on Foundations of Software Technology and Theoretical Computer Science*, Berlin, 1989. Springer.

[28] Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister, editors. *Extensions of Logic Programming*, LNAI 1050, Berlin, 1996. Springer.

[29] Thomas Eiter and Georg Gottlob. Complexity aspects of various semantics for disjunctive databases. In *Proc. of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'93)*, pages 158–167, 1993.

[30] Thomas Eiter and Georg Gottlob. Mächtigkeit von Logikprogrammierung über Datenbanken. *KI*, 3:32–39, 1996.

[31] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Adding disjunction to datalog. In *Proc. of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'94)*, pages 267–278, 1994.

[32] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A Deductive System for Nonmonotonic Reasoning. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Non-*

monotonic Reasoning (LPNMR '97), number 1265 in Lecture Notes in AI (LNAI), Berlin, 1997. Springer.

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

[34] M. Gelfond and V. Lifschitz. Compiling circumscriptive theories into logic programs. In *Proceedings of the Second Workshop on Non-monotonic Reasoning, Munich, July 1988*, pages 74–99, 1988.

[35] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Proceedings of the Fifth Logic Programming Symposium*, pages 1070–1080, Cambridge, Mass., 1988. Association for Logic Programming, MIT Press.

[36] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceedings of the Seventh International Logic Programming Conference, Jerusalem, Israel*, pages 579–597, Cambridge, Mass., 1990. Association for Logic Programming, MIT Press.

[37] M. Gelfond, V. Lifschitz, H. Przymusinska, and M. Truszczyński. Disjunctive defaults. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning, KR'91*, pages 230–237. Morgan Kaufmann, 1991.

[38] Michael Gelfond, Halina Przymusinska, and Teodor Przymusinski. On the Relationship between Circumscription and Negation as Failure. *Artificial Intelligence*, 38:75–94, 1989.

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

[40] Joshua S. Hodas and Dale Miller. Logic programming in a fragment of intuitionistic linear logic. *Journal of Information and Computation*, 110(2):327–365, 1994.

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

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

[43] John W. Lloyd and Rodney W. Topor. Making Prolog more expressive. *The Journal of Logic Programming*, 1:225–240, 1984.

[44] John W. Lloyd and Rodney W. Topor. A basis for deductive database systems. *The Journal of Logic Programming*, 2:93–109, 1985.

[45] John W. Lloyd and Rodney W. Topor. A basis for deductive database systems II. *The Journal of Logic Programming*, 3:55–67, 1986.

[46] Jorge Lobo, Jack Minker, and Arcot Rajasekar. *Foundations of Disjunc-*

tive Logic Programming. MIT-Press, 1992.

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

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

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

[50] Jack Minker. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue*, 17(2/3/4):95–126, 1993.

[51] Jack Minker. Logic and databases: A 20 year retrospective. In Dino Pedreschi and Carlo Zaniolo, editors, *Proceedings of the International Workshop on Logic in Databases (LID)*, LNCS 1154, pages 3–58. Springer, Berlin, 1996.

[52] Gopalan Nadathur and Dale Miller. Higher-order horn clauses. *Journal of the Association for Computing Machinery (JACM)*, 37(4):777–814, 1990.

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

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

[55] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[56] T. C. Przymusinski. 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*, pages 952–959, Los Altos, CA, 1994. Proceedings of the Conference of the American Association of Artificial Intelligence, Morgan Kaufmann.

[57] Teodor Przymusinski. Semantics of normal and disjunctive logic programs: A unifying framework. In J. Dix, L. Pereira, and T. Przymusinski, editors, *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*, pages 43–67. Springer, 1995.

[58] Teodor C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464, 1990.

[59] Teodor C. Przymusinski. Static semantics for normal and disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:323–357,

1995. Special Issue on Disjunctive Programs.

[60] Teodor C. Przymusinski. Autoepistemic logic of knowledge and beliefs. *Artificial Intelligence*, 95:115–154, 1998.

[61] Chiaki Sakama and Hirohisa Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In *Logic Program Synthesis and Transformation – Meta Programming in Logic*, LNCS 883, pages 170–182, Berlin, 1994. Springer.

[62] Dorothea Schäfer and Gerd Neugebauer. Opening the world to theorem provers. In J. Dix, U. Furbach, and A. Nerode, editors, *Logic Programming and Non-Monotonic Reasoning, Proceedings of the Fourth International Conference*, LNAI 1265, pages 410–419, Berlin, June 1997. Springer.

[63] Frieder Stolzenburg and Bernd Thomas. 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*, pages 269–282, London, 1996. Practical Application Company.

[64] Frieder Stolzenburg and Bernd Thomas. Analysing Rule Sets for the Calculation of Banking Fees by a Theorem Prover. In Wolfgang Bibel and Peter H. Schmitt, editors, *Automated Deduction — A Basis for Applications*. Kluwer Academic Publishers, 1998. to appear.

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

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

[67] Allen van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991. Preliminary abstract appeared in the Seventh ACM Symposium on Principles of Database Systems, March 1988, pp. 221–230.

[68] Jia-Huai You and Li-Yan Yuan. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning*, 10:143–160, 1993.

A Equivalence of Knowledge Bases and Belief Theories

The notion of a *non-monotonic knowledge base* defined in this paper is equivalent to the notion of a *belief theory* in the *Autoepistemic Logic of Beliefs*, *AEB*, originally defined by Przymusinski in [56, 60]. Belief theories were defined using a propositional language $\mathcal{L}$ augmented with a *belief operator* $\mathcal{B}$ instead of the default negation operator *Not* . In order to *translate* a non-monotonic knowledge base into a belief theory all one needs to do is to replace everywhere *Not* F by $\mathcal{B}\neg F$. Conversely, in order to *translate* a belief theory into a non-monotonic knowledge base all one needs to do is to replace everywhere $\mathcal{B}F$ by *Not* $\neg F$.

As a result, *Not* F can be simply viewed as a shorthand for $\mathcal{B}\neg F$ and, vice versa, $\mathcal{B}F$ can be viewed as a shorthand for *Not* $\neg F$. However, in the definition of non-monotonic knowledge bases, in addition to using the above substitution, we also used different axioms and derivation rules from those used in the Logic *AEB*. Consequently, it is not entirely obvious that the two logics are equivalent. Below, we recall the definition of belief theories in *AEB* and formally establish the equivalence of the two logics.

Consider a fixed propositional language $\mathcal{L}$ with standard connectives ($\vee$, $\wedge$, $\rightarrow$, $\leftarrow$, $\leftrightarrow$, $\neg$) and the propositional letters *true* and *false*. Extend the language $\mathcal{L}$ to a propositional modal language $\mathcal{L}_{AEB}$ by augmenting it with a modal operator $\mathcal{B}$, called the *belief* operator. The atomic formulae of the form $\mathcal{B}F$, where F is an arbitrary formula of $\mathcal{L}_{AEB}$, are called *belief atoms*. The formulae of the original language $\mathcal{L}$ are called *objective*. Any propositional theory in the modal language $\mathcal{L}_{AEB}$ is called a *belief theory*.

Assume the following two simple axiom schemata and the necessitation inference rule describing the arguably obvious properties of belief atoms:

(D) Consistency Axiom:

$$\neg\mathcal{B}false \tag{A.1}$$

(K) Normality Axiom:
For any formulae F and G:

$$\mathcal{B}(F \rightarrow G) \ \rightarrow \ (\mathcal{B}F \rightarrow \mathcal{B}G) \tag{A.2}$$

(N) Necessitation Inference Rule:
For any formulae F:

$$\frac{F}{\mathcal{B}F} \tag{A.3}$$

The first axiom states that falsity is *not* believed. The second axiom states that if we believe that a formula F implies a formula G and if we believe that F is true then we believe that G is true as well. The necessitation inference rule states that if a formula F has been proven to be true then F is also believed to be true.

Definition A.1 (Formulae Derivable from a Belief Theory)
For any belief theory T, we denote by $Cn_{AEB}(T)$ the smallest set of formulae of the language $\mathcal{L}_{AEB}$ which contains the theory T, all (substitution instances of) the axioms (D) and (K) and is closed under both standard propositional consequence and the necessitation rule (N).

We say that a formula F is derivable *from theory T in the logic AEB if F belongs to $Cn_{AEB}(T)$. We denote this fact by $T \vdash_{AEB} F$. A belief theory T is* consistent *if the theory $Cn_{AEB}(T)$ is consistent.*

In the presence of the axiom (K), the axiom (D) is equivalent to the axiom $\mathcal{B}F \rightarrow \neg\mathcal{B}\neg F$, stating that if we believe in a formula F then we do *not* believe in $\neg F$ (see [60] for details). For readers familiar with modal logics it should be clear by now that we are, in effect, considering here a *normal* modal logic with one modality $\mathcal{B}$ which satisfies the consistency "no dead ends") axiom (D) [47].

In order to *translate* a non-monotonic knowledge base into a belief theory all we need to do is to replace everywhere *Not* F by $\mathcal{B}\neg F$ thus giving the default negation *Not* F the intended meaning of "$\neg F$ is believed":

$$Not\, F \;\equiv\; \mathcal{B}\neg F.$$

Clearly, upon such a translation, the axioms (CA) and (DA) and the rule (IR) take the form:

(CA') Consistency Axiom:

$$\mathcal{B}true \;\text{ and }\; \neg\mathcal{B}false \tag{A.4}$$

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

$$\mathcal{B}(F \wedge G) \leftrightarrow \mathcal{B}F \wedge \mathcal{B}G \tag{A.5}$$

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

$$\frac{F \leftrightarrow G}{\mathcal{B}F \leftrightarrow \mathcal{B}G} \tag{A.6}$$

The consistency axiom (CA') states that *true* is believed while $false$ is not believed. The second axiom (DA') states that beliefs $\mathcal{B}$ are distributive with respect to conjunctions. The invariance inference rule (IR') states that if two formulae are known to be equivalent then so are beliefs in these formulae. In other words, the meaning of $\mathcal{B}F$ does not depend on the specific form of the formula F.

In order to show that the notion of a non-monotonic knowledge base is equivalent to the notion of a belief theory it suffices to show that the axioms (D) and (K) and the necessitation rule (IR) are equivalent to the axioms (CA') and (DA') and the inference rule (IR').

Proposition A.2 (Knowledge Bases vs. Belief Theories)
The axioms (D) and (K) and the necessitation rule (N) are equivalent to the axioms (CA') and (DA') and the inference rule (IR').

More precisely, a theory in the language $\mathcal{L}_{AEB}$ contains all (substitution instances of) the axioms (D) and (K) and is closed under both standard propositional consequence and the necessitation rule (N) if and only if it contains all (substitution instances of) the axioms (CA') and (DA') and is closed under both standard propositional consequence and the inference rule (IR').

Proof. "⇒:" Suppose that a theory T contains all (substitution instances of) the axioms (D) and (K) and is closed under both standard propositional consequence and the necessitation rule (N). Since *true* belongs to T, from the necessitation rule (N) it follows that so does $\mathcal{B}true$ and thus the axiom (CA') holds.

To show that the axiom (DA') holds we first show that for any formulae F and G:

$$\mathcal{B}(F \wedge G) \rightarrow \mathcal{B}F \wedge \mathcal{B}G$$

belongs to T. Clearly $(F \wedge G) \rightarrow F \in T$ and therefore, by the Necessitation Rule (N), $\mathcal{B}((F \wedge G) \rightarrow F) \in T$. From the Normality Axiom (K) we infer that $\mathcal{B}(F \wedge G) \rightarrow \mathcal{B}F \in T$. Similarly, $\mathcal{B}(F \wedge G) \rightarrow \mathcal{B}G \in T$. It follows that $\mathcal{B}(F \wedge G) \rightarrow \mathcal{B}F \wedge \mathcal{B}G \in T$.

To show that the axiom (DA') holds it now suffices to prove that for any formulae F and G:

$$\mathcal{B}F \wedge \mathcal{B}G \rightarrow \mathcal{B}(F \wedge G)$$

belongs to T. Clearly, $F \rightarrow (G \rightarrow F \wedge G) \in T$ and therefore, by the Necessitation Rule (N), $\mathcal{B}(F \rightarrow (G \rightarrow F \wedge G)) \in T$. From the Normality Axiom

(K) we infer that $\mathcal{B}F \rightarrow \mathcal{B}(G \rightarrow F \wedge G) \in T$. Applying the Normality Axiom (K) again we conclude that $\mathcal{B}F \rightarrow (\mathcal{B}G \rightarrow \mathcal{B}(F \wedge G)) \in T$. This shows that $\mathcal{B}F \wedge \mathcal{B}G \rightarrow \mathcal{B}(F \wedge G) \in T$.

To demonstrate that the inference rule (IR') holds suppose that $F \leftrightarrow G$ belongs to T. By necessitation, also $\mathcal{B}(F \leftrightarrow G)$ belongs to T. From the Normality axiom it follows immediately that so does $\mathcal{B}F \leftrightarrow \mathcal{B}G$.

"$\Leftarrow$:" Suppose that a theory T contains all (substitution instances of) the axioms (CA') and (DA') and is closed under both standard propositional consequence and the inference rule (IR'). Clearly, the axiom (C) also holds. Since $(F \rightarrow G) \wedge F \leftrightarrow F \wedge G$ belongs to T, from the rule (IR') it follows that $\mathcal{B}((F \rightarrow G) \wedge F) \leftrightarrow \mathcal{B}(F \wedge G)$ belongs to T. From the axiom (DA') we conclude that also $\mathcal{B}(F \rightarrow G) \wedge \mathcal{B}(F) \leftrightarrow \mathcal{B}F \wedge \mathcal{B}G$ belongs to T. This proves that $\mathcal{B}((F \rightarrow G)) \wedge \mathcal{B}(F) \rightarrow \mathcal{B}(G)$ belongs to T and thus the Normality axiom (N) holds.

It remains to show that T is closed under necessitation. If F belongs to T then so does $F \leftrightarrow \mathit{true}$. By the rule (IR'), the formula $\mathcal{B}F \leftrightarrow \mathcal{B}\mathit{true}$ belongs to T which implies that $\mathcal{B}F$ also belongs to T. □

Conversely, in order to *translate* a belief theory into a non-monotonic knowledge base all we need to do is to replace everywhere $\mathcal{B}F$ by $\mathit{Not}\,\neg F$ thus giving the belief atoms $\mathcal{B}F$ the meaning of "formula F can be assumed true by default".

B Proofs of Theorems from Section 3

B.1 Proof of Proposition 3.3

"$\Rightarrow$:" Suppose $\mathcal{I}$ is not a model of $\mathit{comp}(T)$, so there is a completion axiom $\mathit{comp}(A, T)$ of an atom A in T:

$$A \leftrightarrow \bigvee_{1 \leq m \leq s} (\mathit{Not}\, G_{1,m} \wedge \ldots \wedge \mathit{Not}\, G_{k_m,m} \wedge \neg A_{1,m} \wedge \ldots \wedge \neg A_{l_m,m} \wedge \neg \mathit{Not}\, F_{1,m} \wedge \ldots \wedge \neg \mathit{Not}\, F_{n_m,m}),$$

which is not satisfied in $\mathcal{I}$. If the direction $\leftarrow$ is not satisfied then $\mathcal{I}$ obviously is not even a model of T. So assume that this direction is satisfied for all completion axioms, i.e. $\mathcal{I}$ is a model of T. If the direction $\rightarrow$ is not satisfied

then $\mathcal{I} \models A$ (consequently, $A \neq false$) but for every rule

$$Not\, G_{1,m} \wedge \ldots \wedge Not\, G_{k_m,m} \\ \rightarrow A \vee A_{1,m} \vee \ldots \vee A_{l_m,m} \vee Not\, F_{1,m} \vee \ldots \vee Not\, F_{n_m,m},$$

with A in the head, one of the $A_{i,j}$ is true or one of the $Not\, F_{i,j}$ is true or one of the $Not\, G_{i,j}$ is false. But then the interpretation $\mathcal{I}'$, which differs from $\mathcal{I}$ only in $\mathcal{I}' \not\models A$, is also a model of T and therefore $\mathcal{I}$ is not minimal.

"$\Leftarrow$:" Suppose conversely that $\mathcal{I}$ is not a minimal model of T. If $\mathcal{I}$ is not even a model, i.e. does not satisfy some rule

$$Not\, G_{1,m} \wedge \ldots \wedge Not\, G_{k_m,m} \\ \rightarrow A \vee A_{1,m} \vee \ldots \vee A_{l_m,m} \vee Not\, F_{1,m} \vee \ldots \vee Not\, F_{n_m,m},$$

then obviously the direction $\leftarrow$ of the completion axioms for every A_i is also violated. So let us assume that $\mathcal{I}$ is a model, but not minimal, so there is a smaller model $\mathcal{I}'$. We can choose $\mathcal{I}'$ in such a way that it is minimal. Let A be an objective atom with $\mathcal{I} \models A$ and $\mathcal{I}' \not\models A$ (consequently, $A \neq false$). Let us consider the completion axiom $comp(A, T)$ for A:

$$A \leftrightarrow \bigvee_{1 \leq m \leq s} (Not\, G_{1,m} \wedge \ldots \wedge Not\, G_{k_m,m} \wedge \neg A_{1,m} \wedge \ldots \wedge \neg A_{l_m,m} \\ \wedge \neg Not\, F_{1,m} \wedge \ldots \wedge \neg Not\, F_{n_m,m}),$$

By the contraposition of the already proven opposite direction, the minimal model $\mathcal{I}'$ is a model of $comp(T)$, so it satisfies the completion axiom. Since $\mathcal{I}' \not\models A$, for every i there is a j such that $\mathcal{I}' \models A_{i,j}$ or $\mathcal{I}' \models Not\, F_{i,j}$ or $\mathcal{I}' \not\models Not\, G_{i,j}$. But since $\mathcal{I}'$ is smaller than $\mathcal{I}$, $\mathcal{I}' \models A_{i,j} \implies \mathcal{I} \models A_{i,j}$ and $\mathcal{I}' \models Not\, F_{i,j} \iff \mathcal{I} \models Not\, F_{i,j}$ and $\mathcal{I}' \not\models Not\, G_{i,j} \iff \mathcal{I} \not\models Not\, G_{i,j}$. Thus, the right hand side is also false in $\mathcal{I}$, but A is true in $\mathcal{I}$, and therefore we conclude that $\mathcal{I} \not\models comp(T)$. □

B.2 Proof of Proposition 3.4

We are supposed to prove that every finite knowledge base T can be transformed into a finite residual knowledge base T_{res}, called the *residuum* of T, so that an interpretation M is a minimal model of T if and only if M is a minimal model of T_{res}.

We will show that every knowledge base T can be transformed into a residual knowledge base T_{res} by applying just two database transformations, namely, "elimination of tautologies" and "unfolding". For a detailed definition of these transformations the reader is referred to [7, 8, 10, 9]. Here we describe them very briefly.

The "elimination of tautologies" allows us to remove an atom if it appears both in the head and in the body of a given clause. The transformation of "unfolding" - also called $GPPE$ – is defined as follows. Suppose that B is an objective atom and let $B \vee \mathcal{A}_i \leftarrow \mathcal{R}_i$, $i = 1, \ldots, n$, be all rules about B, i.e., rules containing B in their head. The application of $GPPE$ to the rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$, that contains B in its body, results in a new knowledge base obtained by deleting the rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$ and adding to the knowledge base the new rules $\mathcal{A} \vee \mathcal{A}_i \leftarrow \mathcal{R} \wedge \mathcal{R}_i$.

It is easy to see that each application of $GPPE$, perhaps combined with tautology elimination, removes a given objective atom B from the body of one clause without introducing any new clauses containing B in their bodies. This means that a finite number of such transformations will lead to a knowledge base that does not contain B's in bodies of its clauses. Applying this procedure to all objective atoms appearing in a (finite) knowledge base will therefore produce a residual knowledge base.

Accordingly, in order to prove Proposition 3.4, it suffices to establish the following lemma.

Lemma B.1 *Let T_1 and T_2 be two knowledge bases such that T_2 results from T_1 by an application of GPPE or deletion of tautologies. Then T_1 and T_2 have the same minimal models.*

The deletion of tautologies is an equivalence transformation, so it does not change the set of models, and thus also not the set of minimal models. Now suppose that we apply GPPE to a rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$, where B is an objective atom, and suppose that $B \vee \mathcal{A}_i \leftarrow \mathcal{R}_i$, $i = 1, \ldots, n$, are all the rules about B, i.e., rules containing B in their heads.

- Since the new rules $\mathcal{A} \vee \mathcal{A}_i \leftarrow \mathcal{R} \wedge \mathcal{R}_i$, resulting from the application of $GPPE$, are logical consequences of the rules in T_1, every model of T_1 is also a model of T_2.
- Let $\mathcal{I}$ be a minimal model of T_2. We first show that it is a model of T_1 (not necessarily minimal). Suppose this is not the case, so the rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$ is violated in $\mathcal{I}$ (this is the only rule in $T_1 \setminus T_2$). This means that $\mathcal{I} \models B$, $\mathcal{I} \models \mathcal{R}$, and $\mathcal{I} \not\models \mathcal{A}$. Since $\mathcal{I}$ is a minimal model of T_2, the interpretation $\mathcal{I}'$ with $\mathcal{I}' \not\models B$ (and otherwise equal to $\mathcal{I}$) cannot be a model of T_2. Since objective atoms do not occur negated in rule bodies, this means that there

is a rule with B in the head, i.e. one of the rules $B \vee \mathcal{A}_i \leftarrow \mathcal{R}_i$ [11], such that $\mathcal{I}' \models \mathcal{R}_i$ and $\mathcal{I}' \not\models \mathcal{A}_i$. Since $\mathcal{I}' \models \mathcal{R}_i$, the atom B cannot be contained in $\mathcal{R}_i$, and since we exclude multiple occurrences in the head, it is also not contained in $\mathcal{A}_i$. Thus, $\mathcal{I} \models \mathcal{R}_i$ and $\mathcal{I} \not\models \mathcal{A}_i$. But now we infer that the new rule $\mathcal{A} \vee \mathcal{A}_i \leftarrow \mathcal{R} \wedge \mathcal{R}_i$ contained in T_2 is violated by $\mathcal{I}$, which is a contradiction.

We now have to show that the sets of minimal models are equal:

- Suppose that $\mathcal{I}$ is a minimal model of T_1. By the first part of this proof, we know that it is a model of T_2. If it were not minimal, there would have to be a smaller minimal model $\mathcal{I}'$ of T_2. But by the second part of this proof it would also have to be a model of T_1, contradicting the assumed minimality of $\mathcal{I}$ as a model of T_1.
- Suppose that $\mathcal{I}$ is a minimal model of T_2. Then it is a model of T_1. If there were a smaller model $\mathcal{I}'$ of T_1, it would also have to be a model of T_2, again leading to a contradiction. □

B.3 Proof of Lemma 3.6

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

Lemma B.2 *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 $\mathcal{I}$ 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 $\mathcal{I}$. □

[11] It is not a new rule $\mathcal{A} \vee \mathcal{A}_i \leftarrow \mathcal{R} \wedge \mathcal{R}_i$, because otherwise $\mathcal{A}$ would contain B, and the unfolded rule $\mathcal{A} \leftarrow B \wedge \mathcal{R}$ would be a tautology and could never be violated.

Now we continue the proof of of Lemma 3.6.

Lemma 3.6 *Let T be an arbitrary finite knowledge base and let*

$$\widehat{T} = Cn_{Not}\Big(T \cup Not\,\neg comp(T_{res})\Big).$$

If a formula F is false in all minimal models of $\widehat{T}$ then $Not\,F$ is already contained in $\widehat{T}$. In other words, $\Psi_T(\widehat{T}) \subseteq \widehat{T}$.

Let T_{Not} be the subset of $\hat{T}$ which consists entirely of default negation atoms, i.e. $Not\,\big(\neg comp(T_{res})\big)$ together with the axioms (CA), all instances of the axiom (DA) and all formulae forced in by the derivation rule (IR). Then the models of $\hat{T}$ coincide with the models of $T \cup T_{Not}$ because the required closure under logical consequences does not change the set of models.

By Lemma B.2, minimal models of $T \cup T_{Not}$ are precisely those minimal models of T which satisfy T_{Not}. Suppose that $\neg F$ holds in all minimal models of $\widehat{T}$. By Proposition 3.3, the minimal models of T coincide with the models of $comp(T_{res})$ and therefore $\neg F$ holds in all models of $comp(T_{res}) \cup T_{Not}$, i.e. it is a propositional consequence of $comp(T_{res}) \cup T_{Not}$. By the compactness of propositional logic, there is a finite subset $\{F_1, \ldots, F_n\} \subseteq comp(T_{res}) \cup T_{Not}$ such that $\{F_1, \ldots, F_n\} \models \neg F$. But then $F_1 \wedge \cdots \wedge F_n \rightarrow \neg F$ is a tautology and therefore, by Proposition 2.3, $Not\,(\neg F_1) \wedge \cdots \wedge Not\,(\neg F_n) \rightarrow Not\,(F)$ is contained in $\hat{T}$.

If F_i is from $comp(T_{res})$, then $Not\,(\neg F_i)$ is obviously contained in $\hat{T}$ and otherwise we get $Not\,(\neg F_i)$ by necessitation (see Proposition 2.3). Thus, by the closure under classical consequences, we conclude $Not\,(F) \in \hat{T}$. □

B.4 Proof of Theorem 3.7

Let $\widehat{T} = Cn_{Not}\Big(T \cup Not\,\neg comp(T_{res})\Big)$. We first show that $\widehat{T} = \Psi_T(T)$, i.e.

$$\widehat{T} = Cn_{Not}\Big(T \cup \{Not\,F : T \models_{\min} \neg F\}\Big).$$

- "$\subseteq$": Let $Not\,(\neg G) \in Not\,\neg comp(T_{res})$, i.e. $G \in comp(T_{res})$. By Theorem 3.5, $T \models_{\min} G$, and thus of course $T \models_{\min} \neg(\neg G)$, so $Not\,(\neg G)$ is contained in the right hand side.
- "$\supseteq$": Let $T \models_{\min} \neg F$. By Theorem 3.5, $comp(T_{res}) \models \neg F$. Then there are $G_1, \ldots, G_n \in comp(T_{res})$ such that $G_1 \rightarrow (G_2 \rightarrow (\ldots(G_n \rightarrow \neg F)\ldots))$ is a propositional tautology. Thus $Not\,\neg(G_1 \rightarrow (G_2 \rightarrow (\ldots(G_n \rightarrow \neg F)\ldots)))$ is contained in $\widehat{T}$ (see Proposition 2.3). Since $Not\,(\neg G_i) \in \widehat{T}$ by construction, we get $Not\,\neg F \in \widehat{T}$ (again by Proposition 2.3).

From Lemma 3.6 it follows that $\Psi_T(\widehat{T}) \subseteq \widehat{T}$. Since $\widehat{T} = \Psi_T(T)$ we conclude that:

$$\Psi_T\Big(\Psi_T(T)\Big) \subseteq \Psi_T(T)$$

which means that $\Psi_T\Big(\Psi_T(T)\Big) = \Psi_T(T)$ and therefore $\Psi_T(\widehat{T}) = \widehat{T}$. This proves that $\widehat{T}$ is a fixed point of the default closure operator Ψ_T and thus is a static expansion of T. Since $\widehat{T} = \Psi_T(T)$ we conclude that $\widehat{T}$ is the least such fixed point and thus coincides with the static completion $\overline{T}$ of T. □

C Kripke Models of Static Expansions

Before we prove the modeltheoretic characterization (Theorem 6.3), 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 6.3, 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 [47]. Since our axioms entail the normality axiom, it suffices to consider normal Kripke structures:

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

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

Definition C.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

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

- *If F is a default negation atom $Not\,(G)$, then*

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

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 $At_{\mathcal{L}} \cup \left\{ Not\,(F) \ : \ F \in \mathcal{L}_{Not} \right\}$: 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 C.3 (Kripke Structures Yield Static Expansions)
Let T be an arbitrary belief theory and $\mathcal{K} = (W, R, V)$ be a Kripke structure satisfying

- *For every $w \in W$, there is a $w' \in W$ with $R(w, w')$ (consistency).*
- *For every $w \in W$, $(\mathcal{K}, w) \models T$, i.e. the interpretation $\mathcal{I} = \mathcal{K}(w)$ is a model of T.*
- *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 $T^{\diamond} := \{F \in \mathcal{L}_{Not} \ : \ \textit{for every } w \in W\textit{: } w \models F\}$ is a static expansion of T.

Proof.

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

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 C.4 *If a Kripke structure $\mathcal{K} = (W, R, V)$ satisfies the conditions of Theorem C.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 C.5 Let us consider the knowledge base of Example 2.13:

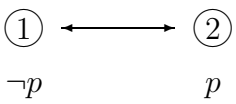
$$Not\,Broken \rightarrow Runs.$$
$$Not\,Fixed \rightarrow Broken.$$

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 $(\mathcal{K}, w) \models Not\,Fixed$, $(\mathcal{K}, w) \not\models Not\,Broken$, and $(K, w) \models Not\,Runs$.

It is, however, also possible to add a world w' with a non-minimal valuation $I'_{obj} = \{Fixed,\ Broken,\ Runs\}$. Of course, 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.

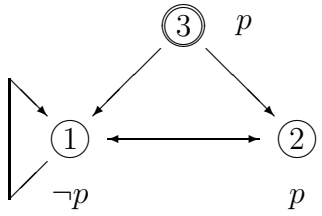
Example C.6 Let us consider the theory $P := \{p \leftarrow Not\,(p)\}$ corresponding to a well-known logic program. We claimed in Section 6 that the least static expansion $\overline{P}$ of this program has infinitely many different models. But let us

first look a Kripke structure which generates only two models:



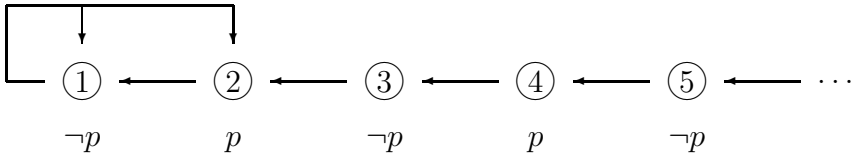
I.e. the set of worlds W is $\{1, 2\}$, the visibility relation R is $\Big\{(1, 2),\ (2, 1)\Big\}$ (each world can see the other one, but not itself), and p is false in world 1 and true in world 2. In world 1, $Not\,(p)$ is false, and in world 2, it is true.

But there are also quite different Kripke models. The construction in Section 6 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 C.3 and Corollary C.4), so the theory

$\{p \leftarrow \mathit{Not}\,(p)\}$ really has infinitely many full models.

D Proof of Modeltheoretic Characterization (Theorem 6.3)

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 6:

Definition D.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:

- *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 $I_{not} \in \mathcal{N}^\diamond$.*
- *$R(I, I')$, i.e. I sees I' iff I' is a minimal model of P and for all $p_1, \ldots, p_n \in At_\mathcal{L}$:*

$$I \models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n) \implies I' \models \neg p_1 \vee \cdots \vee \neg p_n.$$

- *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.

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

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

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

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

Proof.

- Let $I_{not} \models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$. 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, $\mathcal{K}(I) \models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$.
- Let $I_{not} \not\models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$. 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 $I_{not} \models \mathit{Not}\,(q_1 \wedge \cdots \wedge q_m)$, 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 $\mathcal{K}(I) \not\models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$. □

Lemma D.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 C.3 are satisfied:

- 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 $p_1, \ldots, p_n \in At_{\mathcal{L}}$:

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

  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.
- The reduced interpretations $I \in W$ satisfy the program P. By Lemma D.2, I and $\mathcal{I} = \mathcal{K}(I)$ agree on the atoms occurring in P. Thus, also $\mathcal{I}$ is a model of P.
- 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 D.2 in order to conclude that I' and $\mathcal{I}'$, and thus I' and $\hat{I}'$ agree on the default atoms).

Now Theorem C.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 D.4 $\mathit{Not}\,(\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G) \vdash_{Not} \mathit{Not}\,(F_1) \wedge \cdots \wedge \mathit{Not}\,(F_m) \rightarrow \mathit{Not}\,(G)$.

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

$$\mathit{Not}\big(F_1 \vee \cdots \vee F_m \vee (\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G)\big) \leftrightarrow \mathit{Not}\,(F_1 \vee \cdots \vee F_m \vee G)$$

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

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

This implies propositionally:

$$\mathit{Not}\,(F_1) \wedge \cdots \wedge \mathit{Not}\,(F_m) \wedge \mathit{Not}\,(\neg F_1 \wedge \cdots \wedge \neg F_m \wedge G) \rightarrow \mathit{Not}\,(G).$$

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

Lemma D.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

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

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

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

This implies that $\mathcal{O}'$ is non-empty because for $n = 0$ the empty conjunction is logically true. Because of the consistency axiom, $I_{not} \not\models \mathit{Not}\,(\mathit{true})$. 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 $\mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$ 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 $\mathit{Not}\,(q_{i,1} \wedge \cdots \wedge q_{i,n_i})$, $i = 1, \ldots, m$, 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

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

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

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

Thus, this formula is also not contained in $\overline{P}$. But the static semantics requires that $\mathit{Not}\,(F) \in \overline{P}$ if $\overline{P} \models_{\min} \neg F$. So if $\mathit{Not}\,(F) \notin \overline{P}$, 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 6.3 follows directly from Lemma D.3 and Lemma D.5:

Theorem 6.3 (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.

(1) 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 minimal objective objective model $I_{obj} \in \Omega_P(\mathcal{N}_2)$ is based on a default 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 $\mathcal{O}_1$.

(2) Part 2 means that for every $I_{not} \in \mathcal{N}OT$:

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

The direction $\Longrightarrow$ is Lemma D.5, and the direction $\Longleftarrow$ follows from Lemma D.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 $I \in W$.

(3) 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 D.3 and the direction $\Longrightarrow$ follows trivially from Lemma D.5 and $P \subseteq \overline{P}$. □

Remark D.1 Note that we have used the finiteness of P only in the proof to Lemma D.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 $\mathit{Not}\,(q_i)$ true and $\mathit{Not}\,(p)$ 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.

E Proof of Fixpoint Characterization (Theorem 5.1)

We prove Theorem 5.1 by using the model-theoretic characterization. More specifically, we show that formulas $\mathit{Not}\,(E_1) \wedge \cdots \wedge \mathit{Not}\,(E_m) \rightarrow \mathit{Not}\,(E_0)$ contained in P^n characterize exactly the default interpretations remaining after n applications of Θ_P. But first we need the monotonicity of the sequence P^n:

Lemma E.1 *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 B.2 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 E.2 (Regular Model)
Let a super program P be given. A default interpretation I_{not} is called regular wrt P iff

- *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 $I_{not} \models Not\,(q_1 \wedge \cdots \wedge q_m)$.*
- *$I_{not} \models Not\,(p_1 \wedge \cdots \wedge p_n)$ if some $p_i \in At_{\mathcal{L}}$ does not occur in P.*

Lemma E.3 *All $I_{not} \in \mathcal{N}_i$, $i \geq 1$ are regular.*

Proof.

- 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, $i \geq 1$.
- 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 E.1 it is contained also in every P^n, $n \geq 1$. □

Lemma E.4 *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 $\mathcal{N}_0 = \mathcal{N}OT$ and $P^0 = P$ and we anyway consider only models of P.

- " $\Longleftarrow$ ": 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 B.2 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}.
- " $\Longrightarrow$ ": Let $I_{not} \models P^{n+1}_{Not}$. By Lemma E.1 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

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

  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 E.5 $P^{n_0} \subseteq \overline{P}|\mathcal{L}^*_{Not}$.

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 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 B.2 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 D.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 5.1. First, a fixpoint is reached after a finite number of iterations, because by Lemma E.3 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 E.1).

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 E.4 we get $I_{not} \in \mathcal{N}^\diamond$. Now the already proven Theorem 6.3 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 E.5.

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 E.1 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 5.1 might hold for infinite programs. This question is topic of our future research.