Negation by Default

Teodor Przymusinski
Department of Computer Science
University of California
Riverside, CA 92521, USA
(teodor@cs.ucr.edu)

UCR-CS-93-5

Abstract

In this paper, we propose a new *semantic framework* for disjunctive logic programs by introducing the so called *static expansions* which extend to disjunctive programs the notions of stable, well-founded and stationary (or partial stable) models while at the same time generalizing the minimal model semantics of positive disjunctive programs. For every disjunctive program P, we define first order extensions of P, called static expansions of P. Any static expansion of P provides the meaning or semantics for P by determining the set of *true sentences* about P. We show that among all static expansions of a disjunctive program there is always the *least* static expansion and we call it the *static completion* $\overline{P}$ of P. The static completion $\overline{P}$ is constructively defined as a fixed point of a natural *minimal model* operator and can be *iteratively* computed. The semantics defined by the static completion $\overline{P}$ is called the *static semantics* of P. The static semantics always coincides with the set of sentences that are true in *all* static expansions of P.

The class of static expansions represents a semantic framework which differs significantly from the other semantics proposed recently for disjunctive programs and databases. It is also defined for a much broader class of programs.

Keywords: Normal and Disjunctive Logic Programs, Deductive Databases, Semantics , Artificial Intelligence.

1 Introduction

During the last couple of years a significant body of knowledge has been accumulated providing us with a better understanding of *semantic issues* in logic programming and the theory of deductive databases. In particular, the class of *perfect models* [ABW88, VG89, Prz88] was shown to provide a suitable semantics for stratified logic programs. Subsequently, two competing, but closely related [Prz91d, Prz91a], extensions of the class of perfect models to normal, *non-disjunctive* logic programs were introduced and extensively investigated. One of them is the class of *well-founded models* [VGRS90] and the other is the class of *stable models* [GL88]. In [Prz90, Prz91c], another extension of the class of perfect models, namely the class of *partial stable models*, later renamed *stationary models* [Prz91b], was introduced[1], for arbitrary normal programs. The class of stationary models includes both the class of stable models and the class of well-founded models. Moreover, every normal program has the *least* stationary model which coincides with its well-founded model.

The problem of extending these results and defining a suitable semantics for the class of *disjunctive logic programs* and deductive databases turned out to be a difficult one, as evidenced by a large number of papers [Ros89, BLM90, BLM89, Prz91c, GL90, Prz91b, Dix91] and the recent book [LMR92] devoted to this issue.

In this paper, we propose a new *semantic framework* for disjunctive logic programs by introducing the so called static expansions of a disjunctive logic program which extend to disjunctive programs the notions of stable, well-founded and stationary models while at the same time generalizing the minimal model semantics of positive disjunctive programs. For every disjunctive program P, we define first order extensions of P, called *static expansions* of P. Any static expansion of P provides the meaning or semantics for P by determining the set of *true sentences* about P. We show that among all static expansions of a disjunctive program there is always the *least* static expansion and we call it the *static completion* $\overline{P}$ of P. The semantics defined by the static completion $\overline{P}$ is called the *static semantics* of P. The static semantics always coincides with the set of sentences that are true in *all* static expansions of P.

The static completion $\overline{P}$ is constructively defined as a fixed point of a natural *minimal model* operator and can be *iteratively* constructed by starting from $P^0 = P$, i.e., from the program P itself, and then at every successor step adding to the previous iteration P^n those sentences which are true in *all minimal models* of P^n, resulting in the next iteration P^{n+1}. The construction continues until no new sentences can be added, i.e., until $P^{m+1} = P^m$, at which point the static completion, $\overline{P}$, is obtained.

Readers familiar with autoepistemic logic will notice that the definition of static expansions is closely related to *autoepistemic logic* and they are referred to [Prz91a, Prz93] for

[1] Partial stable models were also called *3-valued stable models.*

more information on the nature of this relationship.

The class of static expansions represents a semantic framework which differs significantly from the other semantics proposed recently for disjunctive programs and databases and is defined for a much broader class of programs. Since for normal programs static expansions correspond to stationary (or partial stable) models, they provide a natural characterization of this important class of models.

The paper is organized as follows:

- In Section 2 we *extend the propositional language* to allow us to formally *express* the so called "negation by default" $not\, C$ which appears in premises of program clauses.
- In Section 3 we define *the default formalism* providing the desired meaning for the negation by default.
- In Section 4 we *develop a formal definition* of static expansions and static semantics based on this formalism.
- In Sections 5 and 6 we give examples and investigate properties of static expansions and static semantics in the class of normal and disjunctive programs, respectively.
- In Section 7 we discuss natural extensions and modifications of the proposed semantic framework.
- Section 8 contains concluding remarks, including a discussion of the relationship between static semantics and the other semantics proposed for normal and disjunctive logic programs.

2 Propositional Language

A *disjunctive logic program* (or a disjunctive deductive database) P is a set of *informal* clauses of the form

$$A_1 \vee \ldots \vee A_l \leftarrow B_1 \wedge \ldots \wedge B_m \wedge not\, C_1 \wedge \ldots \wedge not\, C_n \qquad (1)$$

where $l \geq 1$; $m,\ n \geq 0$ and A_i, B_i and C_i's are atomic formulae. If $l = 1$, for all clauses, then the program is called *normal* or *non-disjunctive*. As usual, we assume (see [PP90]) that the program P has been already *instantiated* and thus all of its clauses (possibly infinitely many) are propositional. This assumption allows us to restrict our considerations to a fixed *propositional language* $\mathcal{L}$. In particular, if the original (uninstantiated) program is *function-free* then the resulting objective language $\mathcal{L}$ is finite.

Clauses (1) are informal because the negation symbol $not\,C$ does not denote the *classical negation* $\neg C$ of C but rather the so called *negation by default*, whose intended meaning is roughly:

$$not\,C \quad \equiv \quad negation of\,C\ can be assumed by default.$$

Before making an attempt to properly formalize negation by default $not\,C$ we must first be able to *express* it in our language. After all, propositional logic includes only one connective for negation, namely "$\neg$", and this connective represents *classical negation*. One possibility to achieve this goal would be to add a new *negation connective* "*not*" to the language; another possibility would be to view $not\,C$ as a modal operator in *modal logic*. Both approaches, however, would take us out of the realm of classical propositional logic.

Instead, we use a different approach (see [Lif86]) which allows us to stay entirely within the bounds of propositional logic. Namely, we extend the original propositional language $\mathcal{L}$ by augmenting it with additional *propositional symbols* $\mathcal{D}_F$, with the intended meaning that *"F is true by default"*. The extended language $\mathcal{L}^*$ is obtained therefore by adding a new propositional symbol $\mathcal{D}_F$ for every propositional formula F in the original language $\mathcal{L}$:

$$\mathcal{L}^* = \mathcal{L} \ \cup \ \{\mathcal{D}_F : F \in \mathcal{L}\}.$$

We call the new propositional symbols $\mathcal{D}_F$ *default propositions*. Formulae that do *not* contain any default propositions are called *objective formulae*[2].

We assume the following two simple axiom schemes describing the properties of default propositions.

The Consistency Axioms: For any tautologically false objective formula F:

$$\neg\mathcal{D}_F \ . \tag{2}$$

The Distribution Axioms: For any objective formulae F and G:

$$\mathcal{D}_{F\wedge G} \quad \equiv \quad \mathcal{D}_F \wedge \mathcal{D}_G \ . \tag{3}$$

The first axiom schema states that tautologically false formulae *cannot* be true by default. The second axiom schema states that the conjunction $F \wedge G$ of formulae F and G is true by default if and only if both F and G are true by default. From now on, we assume that all theories *include* the axioms (2) and (3) and are *closed* under the usual propositional consequence. In other words, when we mention a theory T we in fact have in mind the theory:

$$T' = Con\{T \cup (2) \cup (3)\}.$$

[2]It is easy to extend the objective language so that it allows also *nested default formulae*. Since they are not needed for our purposes, we do not discuss this issue in this paper.

We can now *formally* express negation by default $not\, F$ by means of the default proposition $\mathcal{D}_{\neg F}$:

$$not\, F \stackrel{def}{\equiv} \mathcal{D}_{\neg F}$$

thus giving it the intended meaning of *"negation of F is true by default"*, or, equivalently, *"F is false by default"*. This allows us to replace the informal program clauses (1) by the formulae:

$$A_1 \vee \ldots \vee A_l \subset B_1 \wedge \ldots \wedge B_m \wedge \mathcal{D}_{\neg C_1} \wedge \ldots \wedge \mathcal{D}_{\neg C_n} \tag{4}$$

of classical *propositional logic* in which $l > 0$, $m, n \geq 0$, A_i's, B_i's and C_i's are *objective* propositional symbols, $\mathcal{D}_{\neg C_i}$'s are default propositions and $\subset$ represents *standard material implication*. In other words, the introduction of default propositions $\mathcal{D}_F$ allows us to *formally* talk about two different types of negation:

- Classical negation $\neg F$,

  and

- Negation by default $\mathcal{D}_{\neg F}$.

Observe that the distributive axiom (3) implies that $\mathcal{D}_{\neg C_1} \wedge \ldots \wedge \mathcal{D}_{\neg C_n}$ is equivalent to $\mathcal{D}_{\neg C_1 \wedge \ldots \wedge \neg C_n}$, which suggests the following, much more general, definition of disjunctive logic programs.

Definition 2.1 (Generalized Logic Programs) *By a* generalized logic program *we mean a (possibly infinite) set of arbitrary propositional clauses in the extended language $\mathcal{L}^*$, namely, formulae of the form:*

$$A_1 \vee \ldots \vee A_m \vee \mathcal{D}_{F_1} \vee \ldots \vee \mathcal{D}_{F_n} \subset B_1 \wedge \ldots \wedge B_k \wedge \mathcal{D}_{G_1} \wedge \ldots \wedge \mathcal{D}_{G_l} \tag{5}$$

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 objective formulae. In particular, programs of the form:

$$A_1 \vee \ldots \vee A_m \subset B_1 \wedge \ldots \wedge B_k \wedge \mathcal{D}_{G_1} \wedge \ldots \wedge \mathcal{D}_{G_l} \tag{6}$$

where $m > 0$, for all clauses, will be called non-negative.

Clearly, disjunctive logic programs, defined in (4), are a special case of *non-negative* generalized logic programs. The class of generalized logic programs is much broader since it permits empty heads and allows default consequents $\mathcal{D}_F$ and more general default premises $\mathcal{D}_G$. For example, we can have a program clause of the form:

$$\neg A \vee \neg B \subset C \wedge \mathcal{D}_{\neg A \vee \neg B}$$

which says that if C is true and if $\neg A \vee \neg B$ can be assumed true by default then $\neg A \vee \neg B$ is true. It is logically equivalent to:

$$false \subset A \wedge B \wedge C \wedge \mathcal{D}_{\neg A \vee \neg B}\ .$$

It is important to point out that the negation symbol "$\neg$" represents here the *true classical negation* and *not* the so called *explicit negation* in the sense of Gelfond and Lifschitz [GL90]. Since the implication symbol represents also classical *material implication* we can freely move literals from one side of the implication to the other. In Section 7 we show how one can further extend this class of programs to allow the use of explicit negation.

In the sequel, unless stated otherwise, by a logic program we mean a generalized logic program defined in (5). We close this section with the following easy lemma.

Lemma 2.1 *The implication:*

$$\mathcal{D}_F \supset \neg\mathcal{D}_{\neg F} \tag{7}$$

is a logical consequence of the axioms (2) and (3).

Proof. To prove (7) it suffices to observe that:

$$\mathcal{D}_F \supset \neg\mathcal{D}_{\neg F} \;\equiv\; \neg\mathcal{D}_F \vee \neg\mathcal{D}_{\neg F} \;\overset{(3)}{\equiv}\; \neg\mathcal{D}_{F\wedge\neg F} \;\overset{(2)}{\equiv}\; true. \;\Box$$

The formula (7) states that if F is true by default then F cannot be false by default. Observe that the reverse implication $\neg\mathcal{D}_F \supset \mathcal{D}_{\neg F}$, namely that F is false by default if F is not true by default, is generally *not* true.

3 Default Formalism

We now have to select a *default formalism* which will be used in the next section to *formally* define the meaning of default propositions $\mathcal{D}_F$. We choose Minker's *Generalized Closed World Assumption GCWA* (see [Min82, GPP89]) or McCarthy's *Circumscription* [McC80] which says that a formula F is true *by default* if and only if F is true in all minimal models of the theory, i.e., if and only if F is *minimally entailed* by the theory. In other words, we *intend* to base the meaning of default propositions $\mathcal{D}_F$ on the principle of predicate minimization:

$$\mathcal{D}_F \;\equiv\; F \; is true in all minimal models.$$

In particular, the intended meaning of negation by default $\mathcal{D}_{\neg F}$ is to be given by:

$$not\, F \overset{def}{\equiv} \mathcal{D}_{\neg F} \;\equiv\; F \; is false in all minimal models.$$

We now give a formal definition of minimal models and minimal entailment.

Definition 3.1 (Minimal Models) *By a* minimal model *of a propositional theory $\mathcal{E}$ in the extended propositional language $\mathcal{L}^*$ we mean a model M of $\mathcal{E}$ with the property that there is* no *smaller model N of $\mathcal{E}$ which coincides with M on* default *propositions. If an objective formula F is true in all minimal models of $\mathcal{E}$ then we write:*

$$\mathcal{E} \;\models_{\min}\; F$$

and say that F *is* minimally entailed *by* $\mathcal{E}$.

For readers familiar with *circumscription*, this means that we are considering predicate circumscription $CIRC(\mathcal{E};\mathcal{O})$ of the theory $\mathcal{E}$ in which objective propositions $\mathcal{O}$ are minimized while default propositions are fixed[3]. In other words, minimal models are obtained by *minimizing* objective propositions and assigning *arbitrary* truth values to default propositions $\mathcal{D}_F$.

The reason that we treat objective and default propositions differently is that the truth values of default propositions $\mathcal{D}_F$ intuitively represent an arbitrary *possible world* or *scenario* and thus are not minimized, while objective propositions represent *objective knowledge* which is subject to minimization based on $GCWA$.

Throughout the paper *models* are represented as (consistent) *sets of literals*. An atom A is *true* in a model M if and only if A belongs to M. An atom A is *false* in a model M if and only if $\neg A$ belongs to M. A model M is *total* if for every atom A either A or $\neg A$ belongs to M. Otherwise, the model is called *partial*. Unless stated otherwise, all models are assumed to be total models. Clearly a (total) model M is *smaller* than a (total) model N if and only if it contains less positive literals (atoms). When describing models we usually list *only* those of their members that are *relevant* for our considerations, typically those whose predicate symbols appear in the program that we are currently discussing.

Example 3.1 *Consider the following normal program* P*:*

$$Fly \quad \leftarrow \quad not\ Ab$$

or, after translation:

$$Fly \quad \subset \quad \mathcal{D}_{\neg Ab}$$

where Ab *stands for* "Abnormal". *In order to find minimal models of* P*, we need to assign an* arbitrary *truth value to the default proposition* $\mathcal{D}_{\neg Ab}$ *and then* minimize *the objective propositions* Ab *and* Fly*. We easily see that* P *has the following two classes of minimal models:*

$$\begin{aligned} M_1 &= \{Fly, \neg Ab,\ \mathcal{D}_{\neg Ab}\} \\ M_2 &= \{\neg Fly, \neg Ab,\ \neg\mathcal{D}_{\neg Ab}\}. \end{aligned}$$

Since the proposition Ab *is false in all minimal models of* P*, while* Fly *is false only in some of them, we have:*

$$P \models_{min} \neg Ab \ \ but \ \ P \not\models_{min} \neg Fly \ \ and \ \ P \not\models_{min} Fly.$$

While the choice of predicate minimization as a default formalism is quite natural and seems to correspond closely to the intuitive meaning of negation in logic programs and

[3]The author is grateful to L.Yuan for pointing out the need to use *prioritized* rather than standard circumscription [YYar].

deductive databases, within the same general framework other non-monotonic default formalisms can be used, leading, in general, to different semantics. We discuss such possible modifications in Section 7.

4 Static Expansions and Static Completions

We now define static expansions and static completions of an arbitrary generalized logic program P. The definition is based on the idea of building an extension $\widehat{P}$ of the program P obtained by augmenting P with precisely those default atoms $\mathcal{D}_F$ for which F is true in all minimal models of $\widehat{P}$. Consequently, the definition of static expansions precisely *enforces* the intended meaning of default propositions $\mathcal{D}_F$ as specified in the previous section.

Definition 4.1 (Static Expansions) *A* static expansion *of a program P is any propositional theory $\widehat{P}$ which satisfies the following fixed point condition:*

$$\widehat{P} \;=\; P \;\cup\; \{\, \mathcal{D}_F : \quad \widehat{P} \;\models_{min}\; F \,\}, \tag{8}$$

where F is an arbitrary objective *formula.*

We recall that all theories are assumed to be *closed* under the usual propositional consequence. Although the definition of static expansions is similar to the definition of Moore's *stable autoepistemic expansions* [Moo85], it uses satisfaction by default $\mathcal{D}_F$ instead of logical satisfaction $\mathcal{L}_F$ and it does *not* specify when a default proposition $\mathcal{D}_F$ is *false* in the expansion. As a result, properties of static expansions are very different from the properties of stable autoepistemic expansions. We also note that, in general, we do *not* assume that static expansions are consistent theories.

It turns out that *every* generalized logic program P has the *least* (in the sense of inclusion) static expansion $\overline{P}$ which is called the *static completion* of P. Static completion $\overline{P}$ has a *constructive* and *iterative* definition as the *least fixed point* of a monotonic operator Ψ_P defined below. This property of static expansions is in sharp contrast with the properties of stable autoepistemic expansions which typically do *not* have least expansions and do *not* admit constructive definitions.

Definition 4.2 (Default Closure Operator) *For any program P define the* default closure *operator Ψ_P by the formula:*

$$\Psi_P(\mathcal{E}) = P \cup \{\mathcal{D}_F : \mathcal{E} \;\models_{min}\; F\} \tag{9}$$

where $\mathcal{E}$ is an arbitrary theory.

We begin with the following easy observation.

Proposition 4.1 *A theory $\mathcal{E}$ is a static expansion of the program P if and only if $\mathcal{E}$ is a fixed point of the operator Ψ_P, i.e., if $\mathcal{E} = \Psi_P(\mathcal{E})$.*

Proof. Theory $\mathcal{E}$ is a fixed point of the operator Ψ_P if $\mathcal{E} = \Psi_P(\mathcal{E}) = P \cup \{\mathcal{D}_F : \mathcal{E} \models_{min} F\}$ which happens if and only if $\mathcal{E}$ is a static expansion of P. □

Consequently, in order to show that every generalized logic program has the least static expansion we need to prove that the operator Ψ_P has the least fixed point. We first establish the *monotonicity* of the operator Ψ_P.

Proposition 4.2 *The operator Ψ_P is monotonic. More precisely, suppose that the theories $\mathcal{E}$ and $\mathcal{E}'$ are extensions of P obtained by adding some default propositions $\mathcal{D}_F$ to P.*

$$If\ \mathcal{E} \subseteq \mathcal{E}'\ then \Psi_P(\mathcal{E}) \subseteq \Psi_P(\mathcal{E}')\ .$$

Proof. Suppose that:

$$\mathcal{E} = P \cup \{\mathcal{D}_{F_s} :\ s \in S\},\ \ \mathcal{E}' = P \cup \{\mathcal{D}_{F_s} :\ s \in S'\}\ and\ S \subseteq S'.$$

We have to show that $\Psi_P(\mathcal{E}) \subseteq \Psi_P(\mathcal{E}')$. It suffices to show that if $\mathcal{E} \models_{min} F$ then $\mathcal{E}' \models_{min} F$. Suppose that $\mathcal{E} \models_{min} F$ and let M be an arbitrary minimal model of $\mathcal{E}'$. Since $\mathcal{E} \subseteq \mathcal{E}'$ and since $\mathcal{E}$ and $\mathcal{E}'$ differ only on the set of default atoms and minimal models do *not* minimize default atoms, M is also a minimal model of $\mathcal{E}$. We conclude that $M \models F$ and therefore $\mathcal{E}' \models_{min} F$. Consequently, $\mathcal{D}_F \in \Psi_P(\mathcal{E}')$ which completes the proof. □

¿From Propositions 4.1 and 4.2 and the well-known Theorem of Tarski, ensuring the existence of least fixed points of monotonic operators, we easily obtain:

Theorem 4.1 (Least Static Expansion) *Every generalized logic program P has the* least *static expansion, namely, the least fixed point $\overline{P}$ of the monotonic default closure operator Ψ_P.*

The least static expansion $\overline{P}$ of a program P can be constructively computed as follows. Let:

$$P^0 = P$$

and suppose that P^α has already been defined for any ordinal number $\alpha < \beta$. If $\beta = \alpha + 1$ is a successor ordinal then define:

$$P^\beta = \Psi_P(P^\alpha) = P \cup \{\mathcal{D}_F : P^\alpha \models_{min} F\},$$

or, equivalently (because the sequence $\{P^\alpha\}$ is monotonically increasing):

$$P^\beta = \Psi_P(P^\alpha) = P^\alpha \cup \{\mathcal{D}_F : P^\alpha \models_{min} F\},$$

where F denotes any objective formula. Else, if β is a limit ordinal, define:

$$P^\beta = \bigcup_{\alpha<\beta} P^\alpha.$$

The sequence $\{P^\alpha\}$ *is monotonically increasing and thus has a unique fixed point* $\overline{P} = P^\lambda = \Psi_P(P^\lambda) = P^{\lambda+1}$, *for some ordinal* λ. *Moreover, if the objective language* $\mathcal{L}$ *is finite (in particular, if the original, uninstantiated program* P *is function-free) then the fixed point* $\overline{P}$ *is reached after* finitely *many steps* λ.

Proof. By Proposition 4.2 the operator Ψ_P is monotonic. By the well-known Theorem of Tarski it must therefore have the least fixed point $\overline{P}$ constructed by consecutive iterations of the operator. From Proposition 4.1 we infer that the least fixed point $\overline{P}$ of the operator Ψ_P is the least static expansion of P.

If the objective language $\mathcal{L}$ is finite then clearly the construction has to stop after finitely many steps. In particular, if the original (uninstantiated) program P does not contain functional symbols then the resulting objective propositional language $\mathcal{L}$ is finite. □

Definition 4.3 (Static completion) *The least static expansion* $\overline{P}$ *of a program* P *is called the* static completion *of* P.

Since the static completion of a program P is obtained by augmenting P with the *least possible* number of default propositions $\mathcal{D}_F$, it can be viewed as the *most skeptical* among static expansions (cf. [HTT87]). Like Clark's *predicate completion* $COMP(P)$ of a program P, the static completion $\overline{P}$ of P describes the *semantics* of P by defining the set of *true sentences* about P.

Definition 4.4 (Static Semantics) *By the* static semantics *of a logic program* P *we mean the semantics consisting of those sentences that are true in the static completion* $\overline{P}$ *of* P.

More generally, any class $\mathcal{K}$ of static expansions of a program P naturally defines the corresponding semantics consisting of those sentences that are true in *all* expansions belonging to class $\mathcal{K}$. Observe, however, that since the static completion of a program P is the *least* (in the sense of inclusion) static expansion of P, it contains those and only those formulae which are true in *all* static expansions of P.

One of the important strengths of static semantics is the fact that it can be computed by means of the constructive iterative minimal model procedure given in Theorem 4.1. As a result, static semantics not only has an elegant fixed point characterization but it can simply be viewed as the *iterated minimal model semantics*. The last fact is important from the procedural point of view. Namely, once a suitable procedure is devised to compute the *minimal model* semantics, it can then be iteratively applied to compute the static semantics.

The following theorem significantly extends Theorem 4.1 and provides a *complete characterization* of all static expansions of a generalized logic program.

Theorem 4.2 (Characterization Theorem) *A theory $\widehat{P}$ is a static expansion of a generalized logic program P if and only if*

(i) $\widehat{P}$ is the static completion $\overline{P \cup \{\mathcal{D}_{F_s} : s \in S\}}$ of a program $P \cup \{\mathcal{D}_{F_s} : s \in S\}$, and,

(ii) $\widehat{P}$ satisfies the condition that $\widehat{P} \models_{min} F_s$, for every $s \in S$.

In particular, the least static expansion $\widehat{P}$ is obtained when the set $\{\mathcal{D}_{F_s} : s \in S\}$ is empty and thus $\widehat{P} = \overline{P}$.

Proof. ($\Longrightarrow$) Suppose that $\widehat{P}$ is a static expansion of P. Then:

$$\widehat{P} = P \cup \{\mathcal{D}_F : \widehat{P} \models_{min} F\}.$$

Since $\widehat{P}$ obviously satisfies the condition that $\widehat{P} \models \mathcal{D}_F$ if $\widehat{P} \models_{min} F$, we infer that $\widehat{P} = \overline{\widehat{P}} = \overline{P \cup \{\mathcal{D}_F : \widehat{P} \models_{min} F\}}$, which shows that $\widehat{P}$ satisfies the condition of the Theorem.

($\Longleftarrow$) Suppose now that $\widehat{P}$ is the static completion $\overline{P \cup \{\mathcal{D}_{F_s} : s \in S\}}$ of a program $P \cup \{\mathcal{D}_{F_s} : s \in S\}$ and satisfies the condition that $\widehat{P} \models_{min} F_s$, for every $s \in S$. Since $\widehat{P}$ is the (least) static expansion of the program $P \cup \{\mathcal{D}_{F_s} : s \in S\}$ we have:

$$\widehat{P} = P \cup \{\mathcal{D}_{F_s} : s \in S\} \cup \{\mathcal{D}_F : \widehat{P} \models_{min} F\}.$$

Since $\widehat{P} \models_{min} F_s$, for every $s \in S$ we obtain:

$$\widehat{P} = P \cup \{\mathcal{D}_F : \widehat{P} \models_{min} F\},$$

which shows that $\widehat{P}$ is a static expansion of P. The last part of the claim follows immediately from Theorem 4.1. □

According to the above theorem, in order to find a static expansion $\widehat{P}$ of a program P one needs to:

- Select a set $\{\mathcal{D}_{F_s} : s \in S\}$ of default propositions.
- Construct the static completion $\widehat{P} = \overline{P \cup \{\mathcal{D}_{F_s} : s \in S\}}$ of the augmented program $P \cup \{\mathcal{D}_{F_s} : s \in S\}$ using the iterative fixed point definition from Theorem 4.1.
- Show that $\widehat{P} \models_{min} F_s$, for every $s \in S$.

It is the first part, namely, the selection (guessing) of the set of default propositions $\{\mathcal{D}_{F_s} : s \in S\}$, that is most difficult. However, one can always choose the *empty* set to obtain the least static expansion, or, equivalently, the static completion $\overline{P}$ of the program P.

Theorem 4.1 also immediately implies:

Corollary 4.3 (Greatest Static Expansion) *Every generalized logic program has the* greatest static expansion *which is always inconsistent.*

Proof. Let $\widehat{P} = \overline{P \cup \{\mathcal{D}_{A \wedge \neg A}\}}$, where A is an arbitrary atom. Clearly $\widehat{P}$ is inconsistent because $\neg\mathcal{D}_{A\wedge\neg A}$ also belongs to $\widehat{P}$ by the Consistency Axioms. Thus $\widehat{P} \models A \wedge \neg A$, which, by Theorem 4.1, implies that $\widehat{P}$ is a stationary expansion.

Example 4.1 *Consider the normal logic program P discussed in Example 3.1:*

$$Fly \quad \leftarrow \quad not\ Ab$$

or, after translation:

$$Fly \quad \subset \quad \mathcal{D}_{\neg Ab}.$$

Its static completion is given by (see Convention 4.1 below):

$$\overline{P} = P \cup \{\mathcal{D}_{\neg Ab},\ \mathcal{D}_{Fly}\}.$$

Indeed, we showed in Example 3.1 that $P \models_{\min} \neg Ab$ and therefore:

$$P_1 = \Psi_P(P) = P \cup \{\mathcal{D}_{\neg Ab}\}.$$

Since, $P \cup \{\mathcal{D}_{\neg Ab}\} \models Fly$ we obtain:

$$P_2 = \Psi_P(P_1) = P \cup \{\mathcal{D}_{\neg Ab},\ \mathcal{D}_{Fly}\}.$$

We easily verify that P_2 is a fixed point, i.e., that $P_2 = \Psi_P(P_2)$ and therefore $\overline{P} = P_2$ is the static completion of P.

It is easy to check, using Theorem 4.2, that P does not have any other (consistent) static expansions. For example, to show that $\widehat{P} = \overline{P \cup \{\mathcal{D}_{Ab}\}}$ is not a static expansion of P it suffices to note that $\widehat{P} \not\models_{\min} Ab$.

Observe that there is a natural correspondence, between the unique static expansion $\overline{P}$ of P and the unique perfect *(*well-founded*, or* stable*) model $M = \{\neg Ab, Fly\}$ of P in which Fly is true and Ab is false.*

Convention 4.1 *Here, and in the rest of the paper, we list only the* relevant *elements $\mathcal{D}_F$ of static expansions $\widehat{P}$ and program iterations P^α. For example, the static completion $\overline{P} = P \cup \{\mathcal{D}_{\neg Ab},\ \mathcal{D}_{Fly}\}$ of the program P discussed in the previous example also contains default propositions $\mathcal{D}_{Fly\vee\neg Fly}$ or $\mathcal{D}_{Fly\vee\neg Ab}$, because the formulae $Fly \vee \neg Fly$ and $Fly \vee \neg Ab$ are logical consequences of $\neg Ab$ and Fly and thus if the latter formulae hold in all minimal models then so do the former ones. However, since default propositions $\mathcal{D}_{Fly\vee\neg Fly}$ or $\mathcal{D}_{Fly\vee\neg Ab}$ do not appear in clauses of the program P, they can be viewed as irrelevant and thus skipped. To make this convention absolutely precise we will from now on assume that whenever we say that $P \cup \{\mathcal{D}_{F_s} : s \in S\}$ is a static expansion $\widehat{P}$ or a program iteration P^α then, in fact, we mean that the theory $P \cup \{\mathcal{D}_F : \{F_s\}_{s\in S} \models F\}$ has this property.*

Example 4.2 *The static completion $\overline{P'}$ of the logic program P' given by:*

$$\begin{array}{rcl} Fly & \subset & \mathcal{D}_{\neg Ab} \\ \neg Fly & & \end{array}$$

consists of:

$$\overline{P'} = P' \cup \{\mathcal{D}_{\neg Ab},\ \mathcal{D}_{Fly}\}.$$

It is therefore inconsistent, which reflects the fact that the program seems to describe contradictory information.

The last example shows that the static completion $\overline{P}$ of a consistent logic program P may be inconsistent. It follows from Theorem 4.1 and from Corollary 4.3 that a program P either has a *consistent* least static expansion (or static completion) $\overline{P}$ or it does *not* have any consistent static expansions in which case its least and greatest static expansions coincide. The following theorem shows that static completions of *non-negative* generalized logic programs (6) are always consistent.

Theorem 4.4 *Static completion $\overline{P}$ of any non-negative generalized logic program P is always consistent. In particular, static completion $\overline{P}$ of any normal or disjunctive logic program P is always consistent.*

Proof. We will prove by induction that P^α, as defined in Theorem 4.1, is consistent, for every α. To see that $P^0 = P$ is consistent it suffices to take an interpretation of $\mathcal{L}^*$ in which all default atoms are false and all objective atoms are true.

Suppose that we already proved that P^α is consistent, for any $\alpha < \beta$. If β is a limit ordinal then, by the Compactness Theorem, P^β must also be consistent as a union of an increasing sequence of consistent theories. If $\beta = \alpha + 1$ then

$$P^\beta = P \cup \{\mathcal{D}_F : P^\alpha \ \models_{min} \ F\}.$$

Since P^α is consistent, the class of formulae F minimally entailed by P^α constitutes a consistent theory and therefore it does of contain any tautologically false formulae. Define an interpretation M so that all the objective atoms and all the default atoms $\mathcal{D}_F$ such that $P^\alpha \ \models_{min} \ F$ are true in M while all the remaining default atoms are false. Clearly M is a model of P^β which satisfies axioms (2) and (3). We conclude that P^β is consistent, which completes the inductive proof. □

Observe that the definition of static expansions and completions $\widehat{P}$ carefully distinguishes between these formulae F which are *known to be true* in $\widehat{P}$, i.e., those for which $\widehat{P} \models F$, and those formulae F which can only be *assumed* to be true by default, i.e., those for which $\widehat{P} \models \mathcal{D}_F$. This important distinction not only increases the *expressiveness* of the language but is in fact quite crucial for many forms of reasoning, e.g., for common-sense and

abductive reasoning. However, if we wanted to use the *closed world assumption* to ensure that a formula F is always true whenever it is true by default, we could easily achieve this goal by adding the simple *derivation rule* $\frac{\mathcal{D}_F}{F}$.

We are now ready to discuss more examples and prove additional properties of static expansions and static semantics.

5 Static Expansions of Normal Programs

We first study normal, non-disjunctive programs.

Example 5.1 *Consider the following program P:*

$$\begin{array}{rcl} Bird & \leftarrow & \\ Ab & \leftarrow & not\, Bird \\ Fly & \leftarrow & not\ Ab, \end{array}$$

or, after translation:

$$\begin{array}{rcl} Bird & & \\ Ab & \subset & \mathcal{D}_{\neg Bird} \\ Fly & \subset & \mathcal{D}_{\neg Ab}. \end{array}$$

In order to compute its static completion $\overline{P}$ we let $P^0 = P$ and observe that since $P^0 \models_{\min} Bird$ we have

$$P^1 = \Psi_P(P^0) = P \cup \{\mathcal{D}_{Bird}\}.$$

By Lemma 2.1, $P^1 \models \neg\mathcal{D}_{\neg Bird}$ and therefore $P^1 \models_{\min} \neg Ab$ which implies:

$$P^2 = \Psi_P(P^1) = P \cup \{\mathcal{D}_{Bird},\ \mathcal{D}_{\neg Ab}\}.$$

Since $P^2 \models Fly$ we obtain:

$$P^3 = \Psi_P(P^2) = P \cup \{\mathcal{D}_{Bird}, \mathcal{D}_{\neg Ab}, \mathcal{D}_{Fly}\}.$$

One easily checks that $P^3 = \Psi_P(P^3)$ is a fixed point of Ψ_P and therefore $\overline{P} = P^3$ is the static completion of P.

Using Theorem 4.2 one easily verifies that this program does not have any other (consistent) static expansions. There is an obvious correspondence between the unique static expansion $\overline{P}$ of P and the unique perfect *(*well-founded*, or* stable*) model $M = \{Bird, \neg Ab, Fly\}$ of P in which Bird and Fly are true and Ab is false.*

Example 5.2 *Consider the following normal program:*

$$\begin{array}{lcl} A & \subset & \mathcal{D}_{\neg B} \\ B & \subset & \mathcal{D}_{\neg A} \\ Q & \subset & A \\ Q & \subset & B\,. \end{array}$$

Since P has minimal models in which both $\mathcal{D}_{\neg A}$ and $\mathcal{D}_{\neg B}$ are true (respectively, false) and thus all of A, B and Q are true (respectively, false) $P \not\models_{\min} A$, $P \not\models_{\min} \neg A$, etc., which shows that $\overline{P} \not\models \mathcal{D}_A$, $\overline{P} \not\models \mathcal{D}_{\neg A}$, etc. Thus (subject to Convention 4.1):

$$\overline{P} \;=\; P\,.$$

However, in addition to the least static expansion $\widehat{P}_0 = \overline{P}$, this program has two more static expansions, namely:

$$\begin{array}{lcl} \widehat{P}_1 & = & \overline{P \cup \{\mathcal{D}_{\neg B}\}} = P \cup \{\mathcal{D}_A,\ \mathcal{D}_{\neg B}, \mathcal{D}_Q\} \\ \widehat{P}_2 & = & \overline{P \cup \{\mathcal{D}_{\neg A}\}} = P \cup \{\mathcal{D}_{\neg A},\ \mathcal{D}_B,\ \mathcal{D}_Q\}. \end{array}$$

Indeed, according to Theorem 4.2, to show that $\widehat{P}_1$ (respectively, $\widehat{P}_2$) is a static expansion it suffices to show that $\overline{P \cup \{\mathcal{D}_{\neg B}\}} \models_{\min} \neg B$ (respectively, $\overline{P \cup \{\mathcal{D}_{\neg A}\}} \models_{\min} \neg A$). Clearly, $P \cup \{\mathcal{D}_{\neg B}\} \models A \wedge Q$ and thus $\overline{P \cup \{\mathcal{D}_{\neg B}\}} \models \mathcal{D}_A \wedge \mathcal{D}_Q$. By Lemma 2.1, $\overline{P \cup \{\mathcal{D}_{\neg B}\}} \models \neg \mathcal{D}_{\neg A}$ and therefore $\overline{P \cup \{\mathcal{D}_{\neg B}\}} \models_{\min} \neg B$ which implies that $\overline{P \cup \{\mathcal{D}_{\neg B}\}} \models \mathcal{D}_{\neg B}$. The proof that $\overline{P \cup \{\mathcal{D}_{\neg A}\}} \models_{\min} \neg A$ is completely symmetric.

It is easy to see that P does not have any other (consistent) static expansions. Observe that there exists a one-to-one correspondence between stationary models M_i of P [Prz90, Prz91c] and static expansions of $\widehat{P}_i$ of P. Indeed, P has three stationary models, namely:

$$\begin{array}{lcl} M_0 & = & \{\} \\ M_1 & = & \{A, \neg B, Q\} \\ M_2 & = & \{\neg A, B, Q\} \end{array}$$

which precisely correspond to static expansions $\widehat{P}_0$, $\widehat{P}_1$ and $\widehat{P}_2$, respectively. Models M_1 and M_2 are stable but the least (well-founded) model M_0 is a partial model which corresponds to the least static expansion (static completion) $\widehat{P}_0 = \overline{P}$.

The above example underlines that fact that $\mathcal{D}_{\neg A}$ represents a *strong interpretation* of negation by default *not* A, namely, the one that requires that we establish that A is false in all minimal models in order to prove $\mathcal{D}_{\neg A}$. Since, in the preceding example, we cannot prove that either A is false in all minimal models or that B is false in all minimal models we cannot conclude either $\mathcal{D}_{\neg A}$ or $\mathcal{D}_{\neg B}$ in $\overline{P}$.

Example 5.3 *Consider the following normal program due to Van Gelder [VGRS90]:*

$$\begin{array}{rcl} A & \subset & \mathcal{D}_{\neg B} \\ B & \subset & \mathcal{D}_{\neg A} \\ Q & \subset & \mathcal{D}_{\neg Q} \\ Q & \subset & \mathcal{D}_{\neg A}. \end{array}$$

This program has precisely three (consistent) static expansions, namely:

$$\begin{array}{rcl} \widehat{P}_0 & = & P \\ \widehat{P}_1 & = & \overline{P \cup \{\mathcal{D}_{\neg B}\}} = P \cup \{\mathcal{D}_A,\ \mathcal{D}_{\neg B}\} \\ \widehat{P}_2 & = & \overline{P \cup \{\mathcal{D}_{\neg A}\}} = P \cup \{\mathcal{D}_{\neg A},\ \mathcal{D}_B, \mathcal{D}_Q\}. \end{array}$$

The proof that $\widehat{P}_i$'s are static expansions is identical as in the previous example. To verify that $\widehat{P}_1$ does not contain $\mathcal{D}_Q$ observe that $\widehat{P}_1 = \overline{P \cup \{\mathcal{D}_A,\ \mathcal{D}_{\neg B}\}}$ and $P \cup \{\mathcal{D}_A,\ \mathcal{D}_{\neg B}\}$ has precisely two types of minimal models, namely:

$$\begin{array}{rcl} M_1 & = & \{A, \neg B,\ Q, \mathcal{D}_A, \mathcal{D}_{\neg B}, \mathcal{D}_{\neg Q}\} \\ M_2 & = & \{A, \neg B,\ \neg Q, \mathcal{D}_A, \mathcal{D}_{\neg B}, \neg\mathcal{D}_{\neg Q}\}. \end{array}$$

which implies that:

$$\widehat{P}_1 \not\models_{min} Q \ and\ \widehat{P}_1 \not\models_{min} \neg Q \ and therefore\ \widehat{P}_1 \not\models \mathcal{D}_Q \ and\ \widehat{P}_1 \not\models \mathcal{D}_{\neg Q}\ .$$

Again, there exists a one-to-one correspondence between stationary models M_i of P [Prz90, Prz91c] and static expansions of $\widehat{P}_i$ of P. Indeed, P has three stationary models, namely:

$$\begin{array}{rcl} M_0 & = & \{\} \\ M_1 & = & \{A, \neg B\} \\ M_2 & = & \{\neg A, B, Q\} \end{array}$$

which precisely correspond to static expansions $\widehat{P}_0$, $\widehat{P}_1$ and $\widehat{P}_2$, respectively. Model M_2 is stable but both models M_1 and M_2 are stationary. As before, the well-founded model M_0 corresponds to the static completion $\overline{P}$.

It turns out that the one-to-one correspondence between stationary models and consistent static expansions holds for *all* normal programs:

Theorem 5.1 *Let P be an arbitrary normal program and let L denote an arbitrary (objective) literal. There is a one-to-one correspondence between stationary (or partial stable) models M of P and consistent static expansions $\widehat{P}$ of P. Namely, if M is a stationary model of P, then*

$$\widehat{P} = \overline{P \cup \{\mathcal{D}_L :\ L \in M\}}$$

is a consistent static expansion of P. Conversely, if $\widehat{P}$ is a consistent static expansion then

$$M = \{L : \mathcal{D}_L \in \widehat{P}\}$$

is a stationary model of P.

In other words:

$$L \in M \quad \equiv \quad \mathcal{D}_L \in \widehat{P}.$$

Proof. Proof is contained in the Appendix. □

Static expansions provide therefore an equivalent description of stationary models. It has been shown in [Prz90, Prz91c] that every logic program P has at least one stationary model and that among all of its stationary models there is always the *smallest* stationary model which coincides with the *well-founded* model. In view of the above Theorem, we immediately conclude:

Corollary 5.2 *For every normal program P the well-founded model M_P of P corresponds to the static completion $\overline{P}$ of P. Consequently, for normal programs, the static, the stationary and the well-founded semantics all* coincide.

Stationary (or partial stable) models of a normal program P include as a proper *subset* the class of all (total) *stable models*. From Theorem 5.1 we immediately infer:

Corollary 5.3 *Let P be an arbitrary normal program. A model M of P is stable if and only if the static expansion $\widehat{P}$ that corresponds to M satisfies the condition that:*

$$\mathcal{D}_A \in \widehat{P} \quad or \quad \mathcal{D}_{\neg A} \in \widehat{P}, \quad for any (objective) atom\ A$$

i.e., if it completely decides the default status of all objective literals.

6 Static Expansions of Generalized Logic Programs

We now consider examples of generalized logic programs (5), and, in particular, examples of *disjunctive programs* (4). According to Theorem 4.4, every non-negative generalized logic program, and, in particular, every disjunctive logic program P, has a *consistent* least static expansion $\overline{P}$. According to Theorem 5.1, the class of static expansions of generalized logic programs extends the class of stationary models of normal programs and therefore the static semantics extends the well-founded semantics of normal programs. As we will see below, static semantics also extends the *minimal model semantics* of positive disjunctive programs. We first consider the following example.

Example 6.1 *Let P be the positive disjunctive program consisting of a simple disjunction:*

$$A \vee B\ .$$

Since

$$P \ \models_{min} \ A \vee B \ and \ P \ \models_{min} \ \neg A \vee \neg B.$$

we obtain:

$$P^1 = \Psi_P(P) = P \cup \{\mathcal{D}_{A \vee B},\ \mathcal{D}_{\neg A \vee \neg B}\}.$$

It is easy to see that $P^1 = \Psi_P(P^1)$ *is a fixed point and therefore* $\overline{P} = P^1$ *is the static completion*[4] *of* P. *It describes therefore the* minimal model semantics *of* P, *namely, the semantics consisting of formulae* $A \vee B$ *and* $\neg A \vee \neg B$. *Program* P *does not have any other (consistent) static expansions.* □

The next result proves that the static semantics of positive disjunctive programs *always* coincides with the *minimal model semantics.*

Theorem 6.1 *Every positive disjunctive program* P *has precisely one consistent static expansion (or static completion)* $\overline{P}$ *which naturally corresponds to the minimal model semantics of* P, *namely:*

$$\overline{P} = P \cup \{\mathcal{D}_F : P \models_{\min} F\}\ .$$

Proof. Since P does not contain any default premises its static completion $\overline{P}$ given by Theorem 4.1 is obtained after just one iteration, i.e.,

$$\overline{P} = P \cup \{\mathcal{D}_F : P\ \models_{min}\ F\}.\ \square$$

Example 6.2 *Let* P *be the disjunctive program consisting of clauses:*

$$\begin{array}{ll} A \vee B & \\ C & \subset\ \ \mathcal{D}_{\neg A} \wedge \mathcal{D}_{\neg B}\ . \end{array}$$

Let $P^0 = P$. *Clearly, in all minimal models of* P^0 *the* objective *disjunctions* $A \vee B$ *and* $\neg A \vee \neg B$ *hold true. Therefore:*

$$P^0\ \models_{min}\ A \vee B \text{ and } P^0\ \models_{min}\ \neg A \vee \neg B$$

and, consequently:

$$P^1 = \Psi_P(P^0) = P^0 \cup \{\mathcal{D}_{A \vee B}, \mathcal{D}_{\neg A \vee \neg B}\}.$$

Now, from Lemma 2.1 it follows that $P^1 \models \neg\mathcal{D}_{\neg A \wedge \neg B}$ *and therefore* $P^1 \models \neg\mathcal{D}_{\neg A} \vee \neg\mathcal{D}_{\neg B}$. *Consequently:*

$$P^1\ \models_{min}\ \neg C \text{ andthus } P^2 = \Psi_P(P^1) = P^1 \cup \{\mathcal{D}_{\neg C}\}.$$

It is easy to see that $P^2 = \Psi_P(P^2) = P^3$ *is a fixed point and therefore:*

$$\overline{P} = P \cup \{\mathcal{D}_{A \vee B}, \mathcal{D}_{\neg A \vee \neg B},\ \mathcal{D}_{\neg C}\}.$$

The resulting semantics coincides with the perfect model semantics of disjunctive programs introduced in [Prz88]. One easily verifies that P *does not have any other (consistent) static expansions.*

[4] We recall again Convention 4.1.

Example 6.3 *Consider now the following non-negative generalized logic program P describing the state of mind of a person planning a trip to either Australia or Europe.*

$$\begin{aligned} Australia \vee Europe & \\ Cry &\subset \mathcal{D}_{\neg Australia \wedge \neg Europe} \\ Happy &\subset \mathcal{D}_{\neg Cry} \\ Bunkrupt &\subset \mathcal{D}_{Australia \wedge Europe} \\ Save\_Money &\subset \mathcal{D}_{\neg Australia \vee \neg Europe} \\ Cancel\_Reservation &\subset \mathcal{D}_{\neg Australia} \\ Cancel\_Reservation &\subset \mathcal{D}_{\neg Europe} \\ Buy\_Tickets &\subset \mathcal{D}_{Australia} \\ Buy\_Tickets &\subset \mathcal{D}_{Europe}\ . \end{aligned}$$

Let $P^0 = P$. Clearly, in all minimal models of P^0 the objective *disjunctions $Australia \vee Europe$ and $\neg Australia \vee \neg Europe$ hold true. Therefore:*

$$P^0 \models_{min} Australia \vee Europe \text{ and } P^0 \models_{min} \neg Australia \vee \neg Europe$$

and, consequently:

$$P^1 = \Psi_P(P^0) = P \cup \{\mathcal{D}_{Australia \vee Europe}, \mathcal{D}_{\neg Australia \vee \neg Europe}\}.$$

Now, from Lemma 2.1 it follows that $P^1 \models \neg\mathcal{D}_{\neg Australia \wedge \neg Europe}$ and $P^1 \models \neg\mathcal{D}_{Australia \wedge Europe}$ and therefore:

$$P^1 \models_{min} \neg Cry,\ P^1 \models_{min} \neg Bunkrupt \text{ and } P^1 \models_{min} Save\_Money$$

and thus

$$P^2 = \Psi_P(P^1) = P \cup \{\mathcal{D}_{Australia \vee Europe},\ \mathcal{D}_{\neg Australia \vee \neg Europe}, \mathcal{D}_{Save\_Money}, \mathcal{D}_{\neg Bankrupt}, \mathcal{D}_{\neg Cry}\}.$$

Consequently, $P^2 \models_{min} Happy$ and therefore:

$$P^3 = \Psi_P(P^2) = P^2 \cup \{\mathcal{D}_{Happy}\}.$$

It is easy to see that $P^3 = \Psi_P(P^3) = P^4$ is the fixed point and therefore:

$$\overline{P} = P \cup \{\mathcal{D}_{Australia \vee Europe},\ \mathcal{D}_{\neg Australia \vee \neg Europe}, \mathcal{D}_{Save\_Money}, \mathcal{D}_{\neg Bankrupt}, \mathcal{D}_{\neg Cry}, \mathcal{D}_{Happy}\}.$$

It establishes that the individual is expected to travel either to Australia or Europe and thus not Cry and be $Happy$ but he is also expected not to do both trips and thus to $Save\_Money$ and not become $Bankrupt$.

Finally, it is easy to see that there is a minimal model of $\overline{P}$ in which $\mathcal{D}_{Australia}$ and $\neg\mathcal{D}_{\neg Europe}$ are true and thus also $\neg\mathcal{D}_{\neg Australia}$, $\neg Cancel\_Reservation$ and $Buy\_Tickets$ must be true. Similarly, there is a minimal model of $\overline{P}$ in which $\mathcal{D}_{\neg Australia}$ and $\neg\mathcal{D}_{Europe}$

are true and thus also $\neg\mathcal{D}_{Australia}$, $\neg Buy\_Tickets$ *and* $Cancel\_reservation$ *must be true. Consequently (using obvious abbreviations):*

$$\overline{P} \not\models_{\min} CR \ , \ \overline{P} \not\models_{\min} BT \ \ and \ \overline{P} \not\models_{\min} \neg CR \ , \ \overline{P} \not\models_{\min} \neg BT \ .$$

One easily verifies that P *does not have any other (consistent) static expansions.*

Observe that the above construction closely resembles the method used to define the *perfect model* semantics of *stratified programs* (see [ABW88, VG89, Prz88]). Its added significance, however, lies in the fact that, as opposed to stratification, it works for *all* programs. More work is needed, however, to determine *effective* ways of computing static completions of disjunctive programs. The main stumbling block is the problem of computing the *minimal model semantics of disjunctive programs.*

It is important to point out that the static completion (or static semantics) $\overline{P}$ of the previously considered program P does *not* contain $\mathcal{D}_{\neg Australia} \vee \mathcal{D}_{\neg Europe}$ or $\mathcal{D}_{Australia} \vee \mathcal{D}_{Europe}$ even though it contains $\mathcal{D}_{\neg Australia \vee \neg Europe}$ and $\mathcal{D}_{Australia \vee Europe}$. As a result, as we have seen, $Cancel\_Reservation$ and $Buy\_Tickets$ are *not* logical consequences of the static semantics[5]. This reflects the notion that from the fact that a disjunction $F \vee G$ is true by default, one does *not* necessarily want to conclude that either F is true by default or G is true by default. In this case, we do not want to cancel our reservations (respectively, buy tickets) to either *Australia* or to *Europe* until we find out precisely *which* one of them we will actually visit and which one we will not. In other words, we usually do *not* want to assume that the default operator $\mathcal{D}$ is *distributive* with respect to disjunctions[6].

However, if needed, we could easily ensure distributivity w.r.t. disjunctions by assuming the *distributive axiom schema* $\mathcal{D}_{F \vee G} \equiv \mathcal{D}_F \vee \mathcal{D}_G$. The fact that the definition of static expansions can be easily modified to accommodate various domain-specific requirements underlines the flexibility of the proposed framework.

Example 6.4 *Finally, let us consider the following generalized logic program* P *(cf. [Prz91c]):*

$$\begin{aligned} Work \vee Tired \vee Sleep & \\ Work &\subset \mathcal{D}_{\neg Tired} \\ Sleep &\subset \mathcal{D}_{\neg Work} \\ Tired &\subset \mathcal{D}_{\neg Sleep} \\ Happy &\subset \mathcal{D}_{\neg Abnormal} \\ \neg Happy &\subset Work \wedge \mathcal{D}_{\neg Paid} \\ Paid. & \end{aligned}$$

[5]However, by Lemma 2.1, $\overline{P}$ contains the weaker formulae $\neg D_{Australia \wedge Europe} \equiv \neg D_{Australia} \vee \neg D_{Europe}$. and $\neg D_{\neg Australia \wedge \neg Europe} \equiv \neg D_{\neg Australia} \vee \neg D_{\neg Europe}$.

[6]This feature of static expansions represents one of several major differences between static expansions and stationary expansions introduced earlier in [Prz91b].

Intuitively, it says that the person involved is either asleep, tired or working but it is not exactly clear which. Moreover, (s)he is normally happy, except when (s)he works and is not paid for it. However, (s)he has been paid and therefore has no reason to be unhappy.

This program has precisely one (consistent) static expansion, namely:

$$\overline{P} = P \cup \{\mathcal{D}_{Work \vee Tired \vee Sleep}, \mathcal{D}_{\neg Work \vee \neg Tired \vee \neg Sleep}, \mathcal{D}_{Paid}, \mathcal{D}_{\neg Abnormal}, \mathcal{D}_{Happy}\},$$

which seems to agree with its intended meaning. Observe that the program contains (true classical) negation in the head of one of its clauses. Moreover, it does not have any (partial or total) disjunctive stable models *[Prz91c, GL90], and therefore its disjunctive (partial or total) stable semantics is undefined.* □

7 Extensions

The proposed formalism is quite flexible by allowing various extensions and modifications. In this section we discuss some of such possible extensions.

7.1 Using a Different Default Formalism

In our approach we used the minimal model semantics $\mathcal{E} \models_{\min} F$ or the *Generalized Closed World Assumption GCWA* [Min82] to define the meaning of satisfaction by default $\mathcal{D}_F$. In other words, $\mathcal{D}_F$ is supposed to be true in a static expansion $\mathcal{E}$ if and only if F is true in all minimal models of $\mathcal{E}$. While the choice of *predicate minimization* (or circumscription) as a default formalism for satisfaction by default $\mathcal{D}_F$ is very natural and seems to closely correspond to the intuitive meaning of negation in logic programs and deductive databases, other non-monotonic default formalisms can be used in place of predicate minimization, naturally leading to different notions of default satisfaction and thus to different semantics.

For example, by using the weak minimal model semantics $\mathcal{E} \models_{wmin} F$ or the *Weak Generalized Closed World Assumption WGCWA* [RLM89, RT88] requiring that $\mathcal{D}_F$ is true in a static expansion $\mathcal{E}$ if and only if F is true in all *weakly minimal models* of $\mathcal{E}$, one can ensure that disjunctions are treated *inclusively* rather than *exclusively.*

Example 7.1 *Consider the following program P :*

$$\begin{array}{ll} A \vee B & \\ C & \subset \ \mathcal{D}_A \wedge \mathcal{D}_B \end{array}$$

Let $P^0 = P$. Clearly, in all weakly *minimal models of P^0 the* objective *disjunction $A \vee B$ holds. On the other hand, while the disjunction $\neg A \vee \neg B$ is true in all minimal models of P it is not true in all weakly minimal models of P. Therefore:*

$$P^0 \models_{wmin} A \vee B \text{ and } P^0 \models_{\min} \neg A \vee \neg B \text{ andyet } P^0 \not\models_{wmin} \neg A \vee \neg B.$$

As a result:

$$P^1 = \Psi_P(P^0) = P^0 \cup \{\mathcal{D}_{A\vee B},\ \mathcal{D}_{\neg A\vee\neg B}\}\ \textit{but}\ P^1_w = \Psi^w_P(P^0) = P^0 \cup \{\mathcal{D}_{A\vee B}\},$$

where by the index "w" we indicate the fact that we are using WGCWA instead of GCWA in the definition of static expansions and static completion. Now, from Lemma 2.1 it follows that $P^1 \models \neg\mathcal{D}_{A\wedge B}$ *and therefore* $P^1 \models \neg\mathcal{D}_A \vee \neg\mathcal{D}_B$. *Consequently:*

$$P^1\ \models_{min}\ \neg C\ \textit{andthus}\ P^2 = P \cup \{\mathcal{D}_{A\vee B},\ \mathcal{D}_{\neg A\vee\neg B},\ \mathcal{D}_{\neg C}\},$$

It is easy to see that $P^2 = \Psi_P(P^2)$ *is a fixed point and therefore:*

$$\overline{P} = P \cup \{\mathcal{D}_{A\vee B}, \mathcal{D}_{\neg A\vee\neg B},\ \mathcal{D}_{\neg C}\}.$$

On the other hand, it is easy to verify that $P^1_w = \Psi^w_P(P^1_w)$ *is a fixed point and therefore:*

$$\overline{P}^w = P \cup \{\mathcal{D}_{A\vee B}\}.$$

We conclude that under GCWA we can derive that both $A \vee B$ *and* $\neg A \vee \neg B$ *as well as* $\neg C$ *hold by default, whereas WGCWA only allows us to derive* $A \vee B$.

7.2 Adding Explicit or "Classical" Negation

As we already know, the negation operator *not* F used in logic programs does not represent the *classical negation*, but rather the so called *negation by default*. Gelfond and Lifschitz pointed out [GL90] that in logic programming, as well as in other areas of non-monotonic reasoning, it is often useful to use *both* the negation by default and classical negation. They developed a semantics for such programs, based on the stable model semantics. As pointed out by many researchers, the form of negation proposed by Gelfond and Lifschitz, which we will denote here by $-A$, does *not* really represent *true* classical negation but rather its *weaker* form which does not require the law of excluded middle $A \vee -A$. Following Pereira et.al. [PAA91] we will call $-A$ *explicit negation*.

For example, the static semantics of the program P:

$$\begin{array}{lcl} A & \subset & B \\ A & \subset & \neg B, \end{array}$$

where $\neg B$ represents *true* classical negation, implies that A is true, because in classical logic $A \vee \neg A$ always holds. On the other hand, according to Gelfond-Lifschitz' semantics, the program P':

$$\begin{array}{lcl} A & \subset & B \\ A & \subset & -B, \end{array}$$

where $-B$ represents *explicit negation*, does *not* imply A, but rather *not* A instead.

One can easily extend our approach to the broader class of programs which permit the use of *explicit negation* and thus allow *three types of negation*, namely:

- Classical negation $\neg F$;
- Negation by default $\mathcal{D}_{\neg F}$;
- Explicit negation $-A$.

In order to use explicit negation it suffices to add to the *original* language $\mathcal{L}$ new *objective propositional symbols* "$-A$", with the intended meaning that $-A$ is the "*explicit negation of* A", and assume the *explicit negation axiom* schema:

$$A \wedge -A \supset false \; or, equivalently, \; -A \supset \neg A \tag{10}$$

for any objective proposition A in the original language $\mathcal{L}$. Observe that, as opposed to *true classical negation* $\neg$, the law of excluded middle $A \vee -A$ is *not* assumed.

As pointed out by Bob Kowalski, the proposition A may describe the property of being *"good"* while proposition $-A$ describes the property of being *"bad"*. The explicit negation axiom states that things cannot be both *"good"* and *"bad"*. We do not assume, however, that things must always be either *"good"* or *"bad"*.

The resulting framework provides a strict *generalization* of Gelfond-Lifschitz' approach. In particular, when applied to normal logic programs with explicit negation, one easily proves a one-to-one correspondence between static expansions and stable models which directly generalizes Corollary 5.3:

Corollary 7.1 *Let P be an arbitrary normal program with explicit negation. There is a one-to-one correspondence between stable models M of P (in the sense of [GL90]) and those static expansions $\widehat{P}$ of P that satisfy the condition that:*

$$\mathcal{D}_A \in \widehat{P} \quad or \quad \mathcal{D}_{\neg A} \in \widehat{P}, \quad for any (objective) atom \; A$$

i.e., those expansions that completely decide the default status of all objective literals.

This approach also eliminates some problems (pointed out by Pereira et.al. [PAA92]), in the definition of partial stable models of programs with explicit negation given in [Prz90]:

Example 7.2 *[PAA92] Consider the program P:*

$$\begin{array}{lll} -A & \leftarrow & \\ A & \leftarrow & not\, B \\ B & \leftarrow & not\, A \end{array}$$

or, after translation:

$$\begin{array}{lll} -A & \subset & \\ A & \subset & \mathcal{D}_{\neg B} \\ B & \subset & \mathcal{D}_{\neg A} \end{array}$$

where $-A$ denotes explicit negation. Let $P^0 = P$. From the explicit negation axioms (10) it follows that $P^0 \models \neg A$. Thus $P^0 \models_{\min} \neg A$ and therefore

$$P^1 = P^0 \cup \{\mathcal{D}_{\neg A}\}.$$

This proves that $P^1 \models_{\min} B$ and therefore one can easily verify that:

$$\overline{P} = P^2 = P^0 \cup \{\mathcal{D}_{\neg A},\ \mathcal{D}_B\}$$

is the static completion of P, as intended.

7.3 Adding More Axioms

One can easily add other axioms or inference rules in order to tailor the formalism of static expansions to the needs of specific application domains. For example, by assuming the *Disjunctive Distributive Axiom Schema:*

$$\mathcal{D}_{F \vee G} \equiv \mathcal{D}_F \vee \mathcal{D}_G$$

one can ensure that the default operator is *distributive with respect to disjunctions.* In some application domains, e.g., in *disjunctive deductive databases*, such an assumption may be very natural. It would ensure, for example, that from the database P:

$$\begin{array}{lll} A \vee B & & \\ C & \subset & \mathcal{D}_{\neg A} \\ C & \subset & \mathcal{D}_{\neg B} \end{array}$$

one can deduce that C is true. Indeed, since $P \models_{\min} A \vee B$ and $P \models_{\min} \neg A \vee \neg B$ we have $\overline{P} \models \mathcal{D}_{A \vee B}$ and $\overline{P} \models \mathcal{D}_{\neg A \vee \neg B}$. In view of the above distributive axiom, we then infer that $\overline{P} \models \mathcal{D}_{\neg A} \vee \mathcal{D}_{\neg B}$ and therefore $\overline{P} \models C$ and $\overline{P} \models \mathcal{D}_C$.

The fact that the definition of static expansions can be easily modified to accommodate various domain-specific requirements underlines the flexibility of the proposed framework.

7.4 Combining Static and Stable Expansions

One can combine static and stable expansions using the following extended definition of static expansions which results in an even more expressive semantic framework:

Definition 7.1 *A theory $\mathcal{E}$ is called an* extended static expansion *of a program P if it satisfies the following fixed-point equation:*

$$\mathcal{E} = P \cup \{\mathcal{D}_F : \mathcal{E} \ \models_{\min} \ F\} \cup \{\mathcal{L}_F : \mathcal{E} \models F\} \ \cup \{\neg\mathcal{L}_F : \mathcal{E} \not\models F\}.$$

Here by $\mathcal{L}_F$ we denote the epistemic *belief operator* as defined in Moore's autoepistemic logic AEL [Moo85]. Observe that the second part of the definition is identical to the definition of *stable expansions* while the first part coincides with the definition of *static expansions*. Naturally, we assume now that the extended language $\mathcal{L}^*$is now closed under the nested use of both operators $\mathcal{D}_F$ and $\mathcal{L}_F$.

The resulting non-monotonic framework strictly extends several non-monotonic formalisms, including *circumscription*, *autoepistemic logic*, various *semantics* proposed for logic programs and deductive databases (stable semantics, well-founded semantics and stationary semantics) as well as Gelfond's *epistemic specifications*. It is studied in detail in [Prz93].

7.5 Choosing a Different Translation of Negation by Default

One could translate negation by default $not\, F$ by using a weaker formula

$$not\, F \stackrel{def}{\equiv} \neg \mathcal{D}_F\,,$$

instead of $not\, F \stackrel{def}{\equiv} \mathcal{D}_{\neg F}$, thus giving it the intended meaning of *"F is* not *true by default"*. This would lead to the translation of the informal program clauses (1) into formulae:

$$A_1 \vee \ldots \vee A_l \subset B_1 \wedge \ldots \wedge B_m \wedge \neg \mathcal{D}_{C_1} \wedge \ldots \wedge \neg \mathcal{D}_{C_n}$$

of propositional logic. For normal programs the resulting semantics is essentially the same but for disjunctive programs it is generally stronger than the static semantics.

8 Concluding Remarks

The class of static expansions presents a semantic framework which bases the notion of *negation by default* $not\, F$ on the simple idea of building a completion $\mathcal{E}$ of a logic program P in which a formula F is satisfied by default, i.e., $\mathcal{D}_F \in \mathcal{E}$, if and only if F is true in all minimal models of $\mathcal{E}$ and subsequently interpreting $not\, F$ to mean $\mathcal{D}_{\neg F}$, i.e., $\neg F$ is true by default. Although similar to the notion of stable autoepistemic expansions [Moo85], it uses satisfaction by default $\mathcal{D}_F$ instead of logical implication $\mathcal{L}_F$ and does *not* explicitly specify when $\mathcal{D}_F$ is supposed to be false.

Static expansions extend to disjunctive programs the class of *stationary* or *partial stable* models of normal, non-disjunctive programs and thus generalize both the *stable* and *well-founded* models. They also extend the *minimal model semantics* of positive disjunctive programs.

Static expansions differ significantly from the class of *stationary expansions* introduced earlier in [Prz91b, Prz91d]. The definition of stationary expansions was based on the idea

of building a completion $\mathcal{E}$ of a logic program P in which negation by default $Not\,F$ of a formula F holds if F is false in all minimal models of $\mathcal{E}$ and $\neg Not\,F$ holds if F is *true* in all minimal models of $\mathcal{E}$. As a result, it blurred the essential distinction between $\neg Not\,F$ and $Not\,\neg F$ leading to some non intuitive results. It also assumed distributivity of Not with respect to *disjunctions,* i.e., $Not(F \vee G) \equiv Not\,F \vee Not\,G$, thus making it impossible to differentiate between *"disjunction of defaults"* and *"default disjunctions"*, like those in Example 6.3. Moreover, it used the so called *"distributive inference rule"* which is no longer assumed in static expansions.

Although, for normal programs, the static, the stationary and the well-founded semantics all *coincide* (see Theorem 5.1), the notions of a static and a stationary expansion differ considerably in the class of *disjunctive* programs, and, consequently, the static and stationary semantics of disjunctive programs are significantly different. Static expansions are also defined for a much broader class of logic programs and they constitute a special case of a more general non-monotonic framework introduced in [Prz93] which generalizes several formalizations of non-monotonic reasoning, including Gelfond's *epistemic specifications* [Gel92].

The proposed approach also differs significantly from the other major semantics proposed recently for disjunctive logic programs and databases (see [LMR92]). In particular:

Generalized well-founded semantics, introduced by Minker et. al. [BLM90, BLM89], extends the minimal model semantics of disjunctive programs but it does *not* extend the well-founded or the stationary or stable models of normal programs.

Extended well-founded semantics, introduced by Ross [Ros89], extends the well-founded semantics of normal programs and the minimal model semantics of disjunctive programs. It does *not*, however, extend the stationary or stable models of normal programs.

Disjunctive (partial) stable semantics, introduced in [Prz91c, GL90], extends both classes of stable and stationary models of normal programs as well as the minimal model semantics of disjunctive programs but it is defined only for a fairly restricted subclass of the class of disjunctive programs.

Similar comments apply to the semantics introduced recently by Dix [Dix92]. As it is the case with most semantics for disjunctive programs, the static semantics is not cumulative (see [Dix91]).

Acknowledgments

The author is grateful to Roland Bol, Juergen Dix, Michael Gelfond, Vladimir Lifschitz, Louis Pereira, Halina Przymusinska, Mirek Truszczynski and D.S. Warren for their helpful

comments.

References

[ABW88] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–142, Morgan Kaufmann, Los Altos, CA., 1988.

[BLM89] C. Baral, J. Lobo, and J. Minker. *Generalized Disjunctive Well-Founded Semantics for Logic Programs: Declarative Semantics.* Research report UMIACS TR 90-39, CS TR 2436, University of Maryland, College Park, Maryland 20742, 1989.

[BLM90] C. Baral, J. Lobo, and J. Minker. Generalized well-founded semantics for logic programs. In *10th International Conference on Automated Deduction, West Germany*, 1990.

[Dix91] J. Dix. Classifying semantics of logic programs. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Proceedings of the First International Workshop on Logic Programming and Non-monotonic Reasoning, Washington, D.C., July 1991*, pages 166–180, MIT Press, Cambridge, Mass., 1991.

[Dix92] Jürgen Dix. Classifying Semantics of Disjunctive Logic Programs. In K. Apt, editor, *LOGIC PROGRAMMING: Proceedings of the 1992 Joint International Conference and Symposium*, pages 798–812, MIT Press, November 1992.

[Gel92] M. Gelfond. *Logic Programming and Reasoning with Incomplete Information.* Technical report, University of Texas at El Paso, 1992.

[GL88] 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, Association for Logic Programming, MIT Press, Cambridge, Mass., 1988.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceedings of the Seventh International Logic Programming Conference, Jerusalem, Israel*, pages 579–597, Association for Logic Programming, MIT Press, Cambridge, Mass., 1990.

[GPP89] M. Gelfond, H. Przymusinska, and T. Przymusinski. On the relationship between circumscription and negation as failure. *Journal of Artificial Intelligence*, 38:75–94, 1989.

[HTT87] J. Horty, R. Thomason, and D. Touretzky. A skeptical theory of inheritance in non-monotonic semantic nets. In *Proceedings AAAI-87*, page , American Association for Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1987.

[Lif86] V. Lifschitz. Pointwise circumscription: a preliminary report. In *Proceedings AAAI-86*, pages 406–410, American Association for Artificial Intelligence, Morgan Kaufmann, Los Altos, CA, 1986.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* MIT Press, Cambridge, Massachusetts, 1992.

[McC80] J. McCarthy. Circumscription– a form of non-monotonic reasoning. *Journal of Artificial Intelligence*, 13:27–39, 1980.

[Min82] J. Minker. On indefinite data bases and the closed world assumption. In *Proc. 6-th Conference on Automated Deduction*, pages 292–308, Springer Verlag, New York, 1982.

[Moo85] R.C. Moore. Semantic considerations on non-monotonic logic. *Journal of Artificial Intelligence*, 25:75–94, 1985.

[PAA91] L.M. Pereira, J.A. Aparicio, and J.J. Alferes. Non-monotonic reasoning with well-founded semantics. In *Proceedings of the Eight International Logic Programming Conference, Paris, France*, pages 475–489, Association for Logic Programming, MIT Press, Cambridge, Mass., 1991.

[PAA92] L.M. Pereira, J.J. Alferes, and J.A. Aparicio. *Well Founded Semantics with Explicit Negation and Default Theory.* Research report, New University of Lisbon, 1992.

[PP90] H. Przymusinska and T. C. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in Artificial Intelligence*, pages 321–367, North-Holland, Amsterdam, 1990.

[Prz88] T. C. Przymusinski. On the declarative semantics of stratified deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216, Morgan Kaufmann, Los Altos, CA., 1988.

[Prz90] T. C. Przymusinski. The well-founded semantics coincides with the three-valued stable semantics. *Fundamenta Informaticae*, 13(4):445–464, 1990.

[Prz91a] T. C. Przymusinski. Autoepistemic logics of closed beliefs and logic programming. In A. Nerode, W. Marek, and V.S. Subrahmanian, editors, *Proceedings of the First International Workshop on Logic Programming and Non-monotonic Reasoning, Washington, D.C., July 1991*, pages 3–20, MIT Press, Cambridge, Mass., 1991.

[Prz91b] T. C. Przymusinski. Semantics of disjunctive logic programs and deductive databases. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proceedings of the Second International Conference on Deductive and Object-Oriented Databases DOOD'81*, pages 85–107, Springer Verlag, Munich, Germany, 1991.

[Prz91c] T. C. Przymusinski. Stable semantics for disjunctive programs. *New Generation Computing Journal*, 9:401–424, 1991. (Extended abstract appeared in: Extended stable semantics for normal and disjunctive logic programs. *Proceedings of the 7-th International Logic Programming Conference, Jerusalem*, pages 459–477, 1990. MIT Press.).

[Prz91d] T. C. Przymusinski. Well-founded completions of logic programs. In *Proceedings of the Eight International Logic Programming Conference, Paris, France*, pages 726–744, Association for Logic Programming, MIT Press, Cambridge, Mass., 1991.

[Prz93] T. C. Przymusinski. *Epistemic Logic of Defaults.* Research report, University of California at Riverside, 1993.

[RLM89] A. Rajasekar, J. Lobo, and J. Minker. Weak generalized closed world assumption. *Journal of Automated Reasoning*, 5:293–307, 1989.

[Ros89] K. Ross. The well founded semantics for disjunctive logic programs. In *Proceedings of the First International Conference on Deductive and Object Oriented Databases*, pages 352–369, 1989.

[RT88] K. Ross and R. Topor. Inferring negative information from disjunctive databases. *Journal of Automated Reasoning*, 4:397–424, 1988.

[VG89] A. Van Gelder. Negation as failure using tight derivations for general logic programs. *Journal of Logic Programming*, 6(1):109–133, 1989. Preliminary versions appeared in *Third IEEE Symposium Logic Programming* (1986), and *Foundations of Deductive Databases and Logic Programming*, J. Minker, ed., Morgan Kaufmann, 1988.

[VGRS90] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 1990. (to appear). Preliminary abstract appeared in Seventh ACM Symposium on Principles of Database Systems, March 1988, pp. 221–230.

[YYar] L. Yuan and J. You. Autoepistemic circumscription and logic programming. Technical Report TR92-18, Dept. of Computing Science, University of Alberta, 1992. *Journal of Automated Reasoning*, (to appear).

A Appendix

A.1 Proof of Theorem 5.1

We first recall the characterization of *stationary models* of normal programs [Prz90] given in [Prz91c]. It uses the Gelfond_Lifschitz *quotient operator* $\frac{P}{M}$ [GL88] which assigns to any normal logic program P and any interpretation M a *positive* normal logic program $\frac{P}{M}$ obtained by *deleting* all clauses containing negative premises *not* A such that A is true in M and *removing* all negative premises *not* A from the remaining clauses.

Theorem A.1 (Stationary Models) *[Prz91c] Suppose that P is a normal logic program and M is an arbitrary partial interpretation. Define M^+ (respectively, M^-) to be the* total *interpretation obtained by making all undefined atoms in M true (respectively, false).*

An interpretation M is a stationary model of P if and only if $M = M_{Pos} \cup M_{Neg}$, where M_{Pos} (respectively, M_{Neg}) is the set of positive (respectively, negative) literals in the least model of the program $\frac{P}{M^+}$ (respectively, $\frac{P}{M^-}$).

We now prove Theorem 5.1.

($\Longrightarrow$) Suppose that $M = M_{Pos} \cup M_{Neg}$ is a stationary model of P and let:

$$\widehat{P} = \overline{P \cup \{\mathcal{D}_L : \ L \in M\}}.$$

We will show that $\widehat{P}$ is a static expansion of P. By Theorem 4.2, it suffices to show that $\widehat{P} \models_{\min} L$, for every $L \in M$. Observe that if $A \in M_{Pos}$ then $\mathcal{D}_A \in \widehat{P}$ and therefore, by Lemma 2.1, $\neg\mathcal{D}_{\neg A} \in \widehat{P}$. Similarly, if $\neg A \in M_{Neg}$ then $\mathcal{D}_{\neg A} \in \widehat{P}$ and therefore, by Lemma 2.1, $\neg\mathcal{D}_A \in \widehat{P}$. This means that a negative premise of the form $\mathcal{D}_{\neg A}$ in a program clause is always true in any model N of $\widehat{P}$ if $\neg A \in M$ and always false in any model N of $\widehat{P}$ if $A \in M$.

Let N be any minimal model of $\widehat{P}$. Since $\widehat{P}$ contains, as a subtheory, the program $\frac{P}{M^+}$, all the positive atoms in the least model of $\frac{P}{M^+}$ must belong to N. Consequently, $M_{Pos} \subseteq N$.

Similarly, since the set of all satisfiable clauses in $\widehat{P}$ is contained in the program $\frac{P}{M^-}$, all the negative literals in the least model of $\frac{P}{M^-}$ must also belong to N. Consequently, $M_{Neg} \subseteq N$. This shows that $\widehat{P} \models_{\min} L$, for every $L \in M$ and thus completes the proof of the first part.

($\Longleftarrow$) The proof in the opposite direction is completely analogous. □