Static Semantics as Program Transformation and Well-Founded Computation *

Stefania Costantini,[1] and Gaetano A. Lanzarone[1]

Università degli Studi di Milano, Dip. di Scienze dell'Informazione
via Comelico 39/41, I-20135 Milano, Italy
costanti@imiucca.csi.unimi.it lanzarone@hermes.mc.dsi.unimi.it

Abstract. In this paper, we propose a new constructive characterization of those semantics for disjunctive logic programs which are extensions of the well-founded semantics for normal programs. Based on considerations about how disjunctive information is treated by a given semantics, we divide the computation of that semantics into two phases. The first one is a program transformation phase, which applies axiom schemata expressing how derivations involving disjunctions are made in the given semantic framework. The second one is a constructive phase, based on a variation of the well-founded model construction for normal programs. We apply this two-phases procedural semantics to the computation of the static semantics of disjunctive logic programs as a case-study, showing how it works and what its results are in several examples. A main perspective of this proposal is a procedural semantics for disjunctive programs consisting of an inefficient preprocessing phase (implementing the program transformation procedure), to be however performed only once, and of an efficient runtime computation, obtained as a variation of any effective procedural semantics for the well-founded model.

1 Introduction

Disjunctive logic programs are significantly more expressive than normal programs; thus, all the semantics proposed for them are of very high complexity (for an overview of recently proposed semantics, their properties and their complexity, see [Bar91], [Dix91], [Dix92], [Got92], [JL92]). This means that it is not easy to conceive reasonable proof procedures for this kind of program.

In this paper, we propose an approach to computing a class of semantics of disjunctive logic programs. The approach divides the construction of a given semantics into parts, to be computed separately.

In our opinion, the following components underly any semantics for disjunctive programs with negation: (i) the specification of how to use explicit disjunctive information in derivation; (ii) the specification of how to relate explicit and

* We thank the anonymous referees for their careful readings of the manuscript and many useful comments.

implicit disjunctive information; (iii) the specification of how to relate disjunction and negation. By varying the choices for (i)-(iii), different (though related) semantics can be obtained.

Our proposal concerns those semantics which are extensions/modifications of the well-founded model semantics (like for instance [CB92], [Prz94]). We define a program transformation procedure, which includes the formalization of points (i)-(ii). To the resulting program, it is then possible to apply a variation of any procedure for the well-founded model construction for normal programs (see [Sch92] for a survey). The variation is defined by modifying the base step of the procedure so as to include the choice for point (iii), i.e. for relating negation and disjunction.

We illustrate our proposal with respect to the Static Semantics for disjunctive programs [Prz94], which appears to be a natural generalization of the well-founded model semantics. We believe, although this is still only a conjecture, that the approach could be easily adapted to other similar existing semantics.

A main perspective of this proposal is a procedural semantics for disjunctive programs consisting of an inefficient preprocessing phase (implementing the program transformation procedure), to be however performed only once, and of an efficient runtime computation, obtained as a variation of any effective procedural semantics for the well-founded model.

Trying to cope with the complexity of declarative and procedural semantics of disjunctive logic programs has been a motivating point of this research. In the present paper, however, the complexity issue is not specifically addressed. In fact, the main aim has been that of investigating the abstract properties of these semantics, independently of computational issues. We defer to future work a precise analysis of the complexity of the steps involved by the proposed approach.

After some preliminary definitions (Section 2) and a summary of the static semantics (Section 3) with some observations (Section 4), we introduce the program transformation procedure (Section 5) and the modified well-founded computation (Section 6). In Section 7 we discuss some examples, and in Section 8 we propose some final remarks.

2 Preliminary Definitions

A *disjunctive logic program* $\mathbf{P}$ is a set of clauses of the form:

$$\mathbf{A}_1 \vee \ldots \vee \mathbf{A_l} \leftarrow \mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m} \wedge \mathbf{not}\, \mathbf{C}_1 \wedge \ldots \wedge \mathbf{not}\, \mathbf{C_n} \tag{1}$$

where $\mathbf{l} \geq 1$, $\mathbf{m}, \mathbf{n} \geq 0$ and $\mathbf{A_i}$, $\mathbf{B_i}$ and $\mathbf{C_i}$'s are atomic formulae. We will call the $\mathbf{A_i}$'s and $\mathbf{B_i}$'s *atoms*, or *positive literals*, and the $\mathbf{not}\, \mathbf{C_i}$'s *negative literals*. If, for every clause, $\mathbf{l} = 1$, then the program is called *normal*. If, in addition, $\mathbf{n} = 0$, then the program is called *definite*.

As it is customary in the literature (see [HP90]), we assume that the program has already been instantiated, i.e. all its (possibly infinitely many) clauses are propositional. By **not** we mean a non–monotonic, commonsense negation whose informal meaning is: **notC** *holds whenever* **C** *is believed to be false, i.e. whenever* $\neg$**C** *is believed to be true*, where $\neg$ is classical negation of first-order predicate logic.

For the semantics of definite logic programs and the semantics of normal logic programs we will follow [Llo87] and [HP90], respectively. By "interpretations" and "models" we mean Herbrand interpretations and Herbrand models. $\mathbf{B_P}$ is, as usual, the Herbrand base of **P**. In the following, we will consider the *well-founded model semantics* of normal programs. The well-founded model of **P** (for short $\mathbf{WFM_P}$, or simply **WFM**) is unique, and is in general 3-valued. Following [Prz89], [HP90] we indicate $\mathbf{WFM_P}$ with $< \mathbf{T}(\mathbf{P}); \mathbf{F}(\mathbf{P}) >$, or with $< \mathbf{T}; \mathbf{F} >$ if there is no ambiguity about **P**. **T** is the set of atoms which are true w.r.t. the **WFM**, **F** the set of atoms which are false. All atoms belonging to $\mathbf{B_P} - (\mathbf{T} \cup \mathbf{F})$ have truth value undefined.

Various equivalent definitions of the well-founded model can be found in the literature, for instance [AVG90], [Gel93] [Prz89], [HP90], [SC94]. In the examples, we will use the constructive definition introduced in [Gel93] and [SC94], that we rephrase below.

Here and in the rest of the paper, literals **not A** are considered to be new atoms. We conventionally assume that, for every atom **A** in the language of **P**, $\mathbf{B_P}$ contains both **A** and **not A**. The following definitions introduce the computation of the **WFM** by means of the computation of all the literals which are true with respect to the **WFM**. The connection to the standard formulation in terms of the sets **T** and **F** is then stated in Theorem 4.

Definition 1 Enhanced Immediate Consequence Operator.
Let $\mathbf{J} \subseteq \mathbf{B_P}$ be the set of literals currently known to be true. I.e., **J** is the current approximation of the **WFM**. The VanEmden-Kowalski's operator **T** [Llo87] is modified into $\mathbf{T_J}$, as follows. In particular, in the base step, the negation **not A** of every atom **A** not belonging to **J** is assumed. The subsequent steps simply apply the clauses of the program, given the set of literals already computed.

$$\begin{aligned}
\mathbf{T_J} \uparrow 0 &= \{\mathbf{not\,A} \in \mathbf{B_P} : \mathbf{A} \notin \mathbf{J}\} \\
\mathbf{T_J} \uparrow \mathbf{n}+1 &= \{\mathbf{A} \in \mathbf{B_P} : \\
&\quad \mathbf{A} \leftarrow \mathbf{B}_1, \ldots, \mathbf{B_k} \text{ is a clause in } \mathbf{P}, \{\mathbf{B}_1, \ldots, \mathbf{B_k}\} \subseteq \mathbf{B_P}\}
\end{aligned}$$

Let $\mathbf{S}(\mathbf{J}) = \mathbf{T_J} \uparrow \omega$.

Let us now assume to iteratively apply the function **S**, starting from $\mathbf{J} = \emptyset$, and then taking **J** to be the result obtained at the previous iteration. We may notice that $\mathbf{S}(\emptyset) = \mathbf{T}_\emptyset \uparrow \omega$ will contain an overestimated set of consequences, since **not A** will be assumed for every $\mathbf{A} \in \mathbf{B_P}$. Vice versa, at the second iteration, $\mathbf{S}(\mathbf{S}(\emptyset)) = \mathbf{T}_{\mathbf{S}(\emptyset)} \uparrow \omega$ will contain an underestimated set of consequences, since the negation of atoms in $\mathbf{S}(\emptyset)$ cannot be assumed. Similarly, the subsequent iteration steps will develop into an *iterated fixpoint computation* [Gel93]. At each second step, an underestimated set of consequences, of growing size with respect to the previous one, is obtained. I. e., at every second step a better approximation of the **WFM** is computed. To capture this behaviour, a function Σ corresponding to a double application of **S** is defined below.

Definition 2 Alternating Fixpoint Computation. Let $\Sigma = \mathbf{S}^2$ correspond to a double application of $\mathbf{S}$, i.e.

$$
\begin{aligned}
\Sigma \uparrow 0 &= \emptyset \\
\Sigma \uparrow \mathbf{n} + 1 &= \Sigma(\Sigma \uparrow \mathbf{n}) = \mathbf{S}(\mathbf{S}(\Sigma \uparrow \mathbf{n}))
\end{aligned}
$$

The following theorems state that Σ computes as least fixed point a set **WFC** of literals, which exactly correspond to the set of literals which are true with respect to the **WFM**.

Theorem 3 Well-founded Model Construction.
Σ is monotonically increasing, and therefore for some ordinal δ:

$$\Sigma \uparrow \delta = \Sigma \uparrow \delta + 1 = \mathbf{WFC}$$

Theorem 4.
Let $\mathbf{A}$ be an atom. Let $\mathbf{T} = \{\mathbf{A} \in \mathbf{WFC}\}$ and $\mathbf{F} = \{\mathbf{A} : \mathbf{not\,A} \in \mathbf{WFC}\}$.
We have $\mathbf{WFM_P} =< \mathbf{T}; \mathbf{F} >$.

Example 1.
Given the normal program:

$$
\begin{aligned}
&\mathbf{a} \leftarrow \mathbf{not\,b} \\
&\mathbf{b} \leftarrow \mathbf{not\,a} \\
&\mathbf{e} \leftarrow \mathbf{f} \\
&\mathbf{f} \leftarrow \mathbf{not\,g} \\
&\mathbf{g} \leftarrow \mathbf{h}
\end{aligned}
$$

the computation of the well-founded model by means of the above procedure is the following.

$$
\begin{aligned}
\Sigma \uparrow \mathbf{0} &= \emptyset \\
\mathbf{S}(\emptyset) &= \{\mathbf{not\,a}, \mathbf{not\,b}, \mathbf{not\,e}, \mathbf{not\,f}, \mathbf{not\,g}, \mathbf{not\,h} \\
&\quad\ \mathbf{a}, \mathbf{b}, \mathbf{e}, \mathbf{f}\} \\
\Sigma \uparrow \mathbf{1} &= \mathbf{S}(\mathbf{S}(\emptyset)) = \{\mathbf{not\,g}, \mathbf{not\,h} \\
&\quad\ \mathbf{e}, \mathbf{f}\} \\
\mathbf{S}(\Sigma \uparrow 1) &= \{\mathbf{not\,a}, \mathbf{not\,b}, \mathbf{not\,g}, \mathbf{not\,h} \\
&\quad\ \mathbf{a}, \mathbf{b}, \mathbf{e}, \mathbf{f}\} \\
\Sigma \uparrow \mathbf{2} &= \mathbf{S}(\mathbf{S}(\Sigma \uparrow 1)) = \Sigma \uparrow 1
\end{aligned}
$$

Therefore, the fixpoint of the sequence is $\mathbf{WFC} = \Sigma \uparrow 1 = \{\mathbf{not\,g}, \mathbf{not\,h}, \mathbf{e}, \mathbf{f}\}$, and thus $\mathbf{T} = \{\mathbf{e}, \mathbf{f}\}$ and $\mathbf{F} = \{\mathbf{g}, \mathbf{h}\}$.

For the sake of simplicity, by abuse of notation, conjunctions and disjunctions will be treated as *sets*, so as not to be concerned about the order of the literals composing them. Conjunctions of positive (respectively, negative) literals will be denoted by names starting with $\mathcal{C}$ (respectively, $\overline{\mathcal{C}}$). Conjunctions of possibly intermixed positive and negative literals will be denoted by names starting with

$\hat{\mathcal{C}}$. Disjunctions of positive (respectively, negative) literals will be denoted by names starting with $\mathcal{D}$ (respectively, $\overline{\mathcal{D}}$). Disjunctions of possibly intermixed positive and negative literals will be denoted by names starting with $\hat{\mathcal{D}}$. By $\sim$ we will mean the operation of negating all the literals composing a conjunction or disjunction: e.g., if $\mathcal{D}\mathbf{A} = \mathbf{A} \vee \mathbf{B}$, we have $\overline{\mathcal{D}}\mathbf{A} = \sim\mathcal{D}\mathbf{A} = \mathbf{notA} \vee \mathbf{notB}$. We will call $\sim\mathcal{D}\mathbf{D}$ the *negation* of $\mathcal{D}\mathbf{D}$. Literals can be seen as particular cases of conjunctions and disjunctions (singleton sets). Again by abuse of notation, we will apply to conjunctions and disjunctions the usual set operators $\in$, $\cup$, $-$ (set difference). Set operators will be used also for comparing disjunctions and conjunctions. For instance, given disjunction $\mathcal{D}\mathbf{A}$ and conjunction $\mathcal{C}\mathbf{B}$ we will write $\mathcal{C}\mathbf{B} \not\subseteq \mathcal{D}\mathbf{A}$ to mean that the atoms composing $\mathcal{C}\mathbf{B}$ are not a subset of the atoms composing $\mathcal{D}\mathbf{A}$.

In this notation, a disjunctive clause of the form 1 is written as:

$$\mathcal{D}\mathbf{A} \leftarrow \mathcal{C}\mathbf{B}, \overline{\mathcal{C}}\mathbf{C}$$

3 A Summary of the Static Semantics

In this section we report the main definitions and results about the static semantics, as introduced in [Prz94]. The static semantics can be seen as an extension of the well-founded model semantics to disjunctive logic programs. In fact, it derives a minimal, in some sense, set of conclusions that can be inferred from a disjunctive program. This is obtained by translating the (instantiated version of) the program into a first-order *Autoepistemic Logic of Knowledge and Beliefs*. The language of these kinds of theories is a propositional language $\mathcal{L}_{\mathcal{B}}$, with standard connectives ($\vee$, and, $\wedge$, or, $\supset$, material implication, $\neg$, negation), the propositional letter $\perp$ (denoting *false*) and a modal operator $\mathcal{B}$, called the *belief* operator. The atomic formulae of the form $\mathcal{B}\mathbf{F}$, where $\mathbf{F}$ is an arbitrary formula of $\mathcal{L}_{\mathcal{B}}$, are called *belief atoms*. $\mathcal{B}\mathbf{F}$ intuitively means "$\mathbf{F}$ is believed" The formulae of $\mathcal{L}_{\mathcal{B}}$ in which $\mathcal{B}$ does not occur are called *objective* and the set of such formulae is denoted by $\mathcal{L}$.

Definition 5 Belief Theory.
An *autoepistemic theory of beliefs*, or just a *belief theory*, is an arbitrary theory in the language $\mathcal{L}_{\mathcal{B}}$.

Definition 6 Affirmative Belief Theory.
An *affirmative belief theory*, is a theory in the language $\mathcal{L}_{\mathcal{B}}$ consisting of a (possibly infinite) set of clauses of the form:

$$\mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m} \wedge \mathcal{B}\mathbf{G}_1 \wedge \ldots \wedge \mathcal{B}\mathbf{G_k} \wedge \neg\mathcal{B}\mathbf{F}_1 \wedge \ldots \wedge \neg\mathcal{B}\mathbf{F_n} \supset \mathbf{A}_1 \vee \ldots \vee \mathbf{A_l} \quad (2)$$

where $\mathbf{l} > 0$, $\mathbf{k}, \mathbf{m}, \mathbf{n} \geq 0$, $\mathbf{A_i}$'s and $\mathbf{B_i}$'s are objective atoms and $\mathbf{F_i}$'s and $\mathbf{G_i}$'s are arbitrary formulae.

A disjunctive logic program $\mathbf{P}$ is translated into a special case of affirmative belief theory, where every clause of the form 1 is considered to be translated into the corresponding axiom:

$$\mathbf{B}_1 \wedge \ldots \wedge \mathbf{B_m} \wedge \mathcal{B}\neg\mathbf{C}_1 \wedge \ldots \wedge \mathcal{B}\neg\mathbf{C_n} \supset \mathbf{A}_1 \vee \ldots \vee \mathbf{A_l} \tag{3}$$

where the $\mathbf{A_i}$'s, $\mathbf{B_i}$'s, and $\mathbf{C_i}$'s are objective atoms.

A *model* of a belief theory is a consistent set of literals. All models are assumed to be *total*, i.e., given atom $\mathbf{A}$, either $\mathbf{A}$ or $\neg\mathbf{A}$ belongs to $\mathbf{M}$. An atom $\mathbf{A}$ is **true** in a model $\mathbf{M}$ (resp. **false**) if $\mathbf{A}$ (resp. $\neg\mathbf{A}$) belongs to $\mathbf{M}$. A model $\mathbf{M}$ is smaller than a model $\mathbf{N}$, if it contains less positive literals (atoms).

Definition 7 Minimal Model of a Belief Theory.
A *minimal model* of a belief theory $\mathbf{T}$ is a model $\mathbf{M}$ of $\mathbf{T}$ such that there is no smaller model $\mathbf{N}$ of $\mathbf{T}$ which coincides with $\mathbf{M}$ on belief atoms.

$\mathbf{T} \models_{\mathbf{min}} \mathbf{F}$ means that $\mathbf{F}$ is true in all minimal models of $\mathbf{T}$, i.e. $\mathbf{F}$ is *minimally entailed* by $\mathbf{T}$.

In a theory $\mathbf{T}$, the following axioms are given, describing obvious properties of belief atoms.

(D) Consistency Axiom:

$$\neg\mathcal{B}\bot$$

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

$$\mathcal{B}(\mathbf{F} \supset \mathbf{G}) \supset (\mathcal{B}\mathbf{F} \supset \mathcal{B}\mathbf{G})$$

(N) Necessitation Rule: For any formula F

$$\frac{\mathbf{F}}{\mathcal{B}\mathbf{F}}$$

Axiom (N) implies that beliefs are distributive with respect to conjunctions, namely, for any formaulae $\mathbf{F}$ and $\mathbf{G}$,

$$\mathcal{B}(\mathbf{F} \wedge \mathbf{G}) \equiv (\mathcal{B}\mathbf{F} \wedge \mathcal{B}\mathbf{G})$$

Instead, it is not assumed that the belief operator is distributive with respect to disjunctions. A variation of the semantics can, however, be easily defined by assuming the

(J) Disjunctive belief axiom

$$\mathcal{B}(\mathbf{F} \vee \mathbf{G}) \equiv \mathcal{B}\mathbf{F} \vee \mathcal{B}\mathbf{G}$$

Definition 8 Formulae Derivable from a Belief Theory.
A formula F is derivable from a belief theory if it belongs to the smallest set, $\mathbf{Cn}_*(\mathbf{T})$, of formulae of the language $\mathcal{L}_{\mathcal{B}}$ which contains the theory T and all (the substitution instances of) the axioms (K), (D), and is closed under the necessitation rule (N). This is denoted by $\mathbf{T}\vdash_*\mathbf{F}$. Consequently,

$$\mathbf{Cn}_*(\mathbf{T}) = \{\mathbf{F} : \mathbf{T}\vdash_*\mathbf{F}\}$$

A belief theory T is called *consistent* if $\mathbf{Cn}_*(\mathbf{T})$ is consistent. Thus, T is consistent if $\mathbf{T} \not\vdash_* \bot$.

Notice that, by the consistency axiom (D),

$$\forall \mathbf{F} \text{ such that } \mathcal{B}\mathbf{F} \in \mathbf{T},\ \mathbf{T} \models \neg\mathcal{B}\neg\mathbf{F}$$

The *Belief Closure Operator* defined below is based on the idea of building extensions of a belief theory $\mathbf{T}$ based on another belief theory $\mathbf{S}$, obtained by augmenting $\mathbf{T}$ with precisely those belief atoms $\mathcal{B}\mathbf{F}$ which satisfy the condition that $\mathbf{F}$ is true in all minimal models of $\mathbf{S}$.

Definition 9 Belief Closure Operator.
For any belief theory $\mathbf{T}$ define the *belief closure* operator $\Psi_{\mathbf{T}}$ by the formula:

$$\Psi_{\mathbf{T}}(\mathbf{S}) = \mathbf{Cn}_*(\mathbf{T} \cup \{\mathcal{B}\mathbf{F} : \mathbf{S} \models_{\mathbf{min}} \mathbf{F}\})$$

where S is an arbitrary belief theory and F ranges over all formulae of $\mathcal{L}_{\mathcal{B}}$

Theorem 10 Monotonicity of the Belief Closure Operator.
Suppose that the theories $\mathbf{V}'$ and $\mathbf{V}''$ are extensions of a belief theory T obtained by adding some belief atoms $\mathcal{B}\mathbf{F}$ to T, and let $\mathbf{T}' = \mathbf{Cn}_*(\mathbf{V}')$ and $\mathbf{T}'' = \mathbf{Cn}_*(\mathbf{T}'')$.

$$\textit{If } \mathbf{T} \subseteq \mathbf{T}' \textit{ then } \Psi_{\mathbf{T}}(\mathbf{T}') \subseteq \Psi_{\mathbf{T}}(\mathbf{T}'')$$

A *Static Expansion* of a belief theory $\mathbf{T}$ is a fixed point of the operator $\Psi_{\mathbf{T}}$.

Definition 11 Static Expansions.
A theory $\hat{\mathbf{T}}$ is a static expansion of the belief theory T iff $\hat{\mathbf{T}} = \Psi_{\mathbf{T}}(\hat{\mathbf{T}})$.

Static expansions $\hat{\mathbf{T}}$ of a belief theory $\mathbf{T}$ are first-order extensions of $\mathbf{T}$, and thus provide the meaning of $\mathbf{T}$ consisting of the sentences logically derivable from $\hat{\mathbf{T}}$. Since $\Psi_{\mathbf{T}}$ is monotonic, it has a least fixed point, as stated below.

Theorem 12 Least Static Expansion.
Every belief theory $\mathbf{T}$ has the least *static expansion $\overline{\mathbf{T}}$, namely the least fixed point of the operator $\Psi_{\mathbf{T}}$.*

Definition 13 Static Completion.
The *least* static expansion $\overline{\mathbf{T}}$ of a belief theory $\mathbf{T}$ is called the *static completion* of $\mathbf{T}$.

Theorem 14 Consistency of Static Completions.
The static completion $\overline{\mathbf{T}}$ of any affirmative belief theory $\mathbf{T}$ is always consistent.

As a special case, the static completion of a belief theory which is the translation of a disjunctive logic program $\mathbf{P}$ is consistent.

Definition 15 Static Semantics.
The static semantics of a belief theory $\mathbf{T}$ is the set of all formulae which belong to the static completion $\overline{\mathbf{T}}$ of $\mathbf{T}$.

In agreement with Minker's *Generalized Closed Word Assumption GCWA*) [Min82] and with McCarthy's *Circumscription* [McC80], a formula $\mathbf{F}$ is believed to be true in the static semantics if and only if it is minimally entailed by $\overline{\mathbf{T}}$. Thus, $\mathcal{B}\mathbf{F}$ means that $\mathbf{F}$ is believed to be true, i.e. that $\mathbf{F}$ is true in all minimal models of $\overline{\mathbf{T}}$. By definition, $\mathbf{not}\,\mathbf{F}$ means $\mathcal{B}\neg\mathbf{F}$, i.e. $\mathbf{F}$ is believed to be false, i.e. $\mathbf{F}$ is false in all the minimal models of $\overline{\mathbf{T}}$, i.e. $\neg\mathbf{F}$ is believed to be true.

For normal programs, the well-founded, static and stationary [Prz91] semantics coincide. For positive disjunctive programs, the static semantics coincides with the minimal model semantics.

In the rest of the paper, let $\overline{\mathbf{P}}$ be the static completion of the affirmative belief theory corresponding to a disjunctive logic program $\mathbf{P}$.

Remark.
Notice that, in the context of an affirmative belief theory which is the translation of a disjunctive logic program $\mathbf{P}$, we are interested in a subset of $\overline{\mathbf{P}}$, consisting of the belief atoms $\mathcal{B}\mathbf{D}$ or $\mathcal{B} \sim \mathbf{D}$, where $\mathbf{D}$ is either a literal or a disjunction, containing at least one atom/disjunction which appears as the conclusion of a clause in $\mathbf{P}$. We will call this subset the *interesting subset* of $\overline{\mathbf{P}}$.

4 Observations on the Static Semantics

In this section we propose some observations about the static semantics, useful for what follows. Conventionally, we extend the notation for disjunctions and conjunctions introduced in Section 2 to conjunctions/disjunctions of formulae of $\mathcal{L}_{\mathcal{B}}$.

An affirmative belief theory is, as defined in the previous section, composed of a set of axioms represented as implications. Some of the axioms may have empty conditions, and will be called *facts*. In the following, given any extension $\hat{\mathbf{P}}$ of the belief theory corresponding to a disjunctive program P, we will call *interesting subset* of $\hat{\mathbf{P}}$ the set of axioms whose conclusion is a disjunction (as a particular case, an atom) containing at least one atom/disjunction which appears as the conclusion of a clause in $\mathbf{P}$.

Since any logical theory implies all the tautologies, this is also the case for belief theories, and for their static expansions. From the necessitation rule (N), it is thus possible to derive belief atoms corresponding to all the instances of all the tautologies of propositional logic over the language $\mathcal{L}_{\mathcal{B}}$, like for instance $\mathcal{B}(\mathbf{F} \vee \neg\mathbf{F})$ for any formula $\mathbf{F}$ of $\mathcal{L}_{\mathcal{B}}$. From the consistency axiom (D), formulae corresponding to the negation of all the tautologies are symmetrically obtained, like for instance $\neg\mathcal{B}((\mathbf{F} \wedge \neg\mathbf{F}))$. Axiom (K) is a distributive axiom which allows modus ponens to be applied to belief formulae. In an affirmative belief theory, several disjunctions can be derived, not only those explicitly present in the conditions of some of the axioms.

Example 2.
From theory:

$$\mathbf{A} \vee \mathbf{B}$$

$$\mathbf{A} \supset \mathbf{C}$$

where $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are objective atoms, given tautology

$$((\mathbf{A} \supset \mathbf{C}) \supset ((\mathbf{A} \vee \mathbf{B}) \supset (\mathbf{C} \vee \mathbf{B})))$$

by axiom (N) it is possible to derive

$$\mathcal{B}(\mathbf{A} \vee \mathbf{B})$$

$$\mathcal{B}(\mathbf{A} \supset \mathbf{C})$$

$$\mathcal{B}((\mathbf{A} \supset \mathbf{C}) \supset ((\mathbf{A} \vee \mathbf{B}) \supset (\mathbf{C} \vee \mathbf{B})))$$

From the last formula, by axiom (K), applied twice, we obtain

$$\mathcal{B}(\mathbf{A} \supset \mathbf{C}) \supset (\mathcal{B}(\mathbf{A} \vee \mathbf{B}) \supset \mathcal{B}(\mathbf{C} \vee \mathbf{B}))$$

Now, by modus ponens it follows

$$\mathcal{B}(\mathbf{A} \vee \mathbf{B}) \supset \mathcal{B}(\mathbf{C} \vee \mathbf{B})$$

and then

$$\mathcal{B}(\mathbf{C} \vee \mathbf{B})$$

The following is a trivial consequence of the definitions reported in the previous section.

Proposition 16.
Let $\mathbf{Z} \subseteq \overline{\mathbf{P}}$ be a set of belief atoms. $\Psi_{\mathbf{P}}(\mathbf{P} \cup \mathbf{Z}) \subseteq \overline{\mathbf{P}}$.

Proof. Since $\mathbf{Z} \subseteq \overline{\mathbf{P}}$, clearly $(\mathbf{P} \cup \mathbf{Z}) \subseteq (\mathbf{P} \cup \overline{\mathbf{P}})$. By the monotonicity of $\Psi_{\mathbf{P}}$, $\Psi_{\mathbf{P}}(\mathbf{P} \cup \mathbf{Z}) \subseteq \Psi_{\mathbf{P}}(\mathbf{P} \cup \overline{\mathbf{P}})$. Since $\mathbf{P} \subseteq \overline{\mathbf{P}}$, and $\overline{\mathbf{P}}$ is a fixed point of $\Psi_{\mathbf{P}}$, it follows $\Psi_{\mathbf{P}}(\mathbf{P} \cup \mathbf{Z}) \subseteq \overline{\mathbf{P}}$). □

In order to compute the static completion of disjunctive logic programs, it is important to characterize the formulae $\mathbf{F}$ such that $\mathbf{T} \models_{\mathbf{min}} \mathbf{F}$, given an affirmative belief theory $\mathbf{T}$. The definitions imply that one has to assign arbitrary truth values to belief atoms which are not in $\mathbf{T}$, then minimize the objective atoms, and select those which have the same truth value in all the minimal models. Some observations are in order about what this means.

Lemma 17.
Let $\mathbf{A}$ be an objective atom. Let $\hat{\mathcal{D}}\mathbf{D}$ be a disjunction such that $\mathbf{T} \models_{\mathbf{min}} \hat{\mathcal{D}}\mathbf{D}$, and $\mathbf{A} \in \hat{\mathcal{D}}\mathbf{D}$ (or, respectively, $\neg\mathbf{A} \in \hat{\mathcal{D}}\mathbf{D}$). Assume $\nexists \hat{\mathcal{D}}\mathbf{H}$, $\hat{\mathcal{D}}\mathbf{H} \subseteq \hat{\mathcal{D}}\mathbf{D}$, such that $\mathbf{A} \in \hat{\mathcal{D}}\mathbf{H}$ (or, respectively, $\neg\mathbf{A} \in \hat{\mathcal{D}}\mathbf{H}$). Then, $\mathbf{T} \not\models_{\mathbf{min}} \neg\mathbf{A}$

Proof. By hypothesis, $\hat{\mathcal{D}}\mathbf{D}$ is not redundant with respect to $\mathbf{A}$ (respectively, $\neg\mathbf{A}$). Since it cannot be decided which of the literals composing $\hat{\mathcal{D}}\mathbf{D}$ are true, any of them can be assumed in a minimal model. Thus, there will be minimal models where $\mathbf{A}$ is true, and others where it is false. Therefore, $\neg\mathbf{A}$ can't be minimally entailed. □

Example 3.
From theory T

$$\mathbf{A} \vee \mathbf{B} \vee \mathbf{C}$$

one cannot decide which literal is true. Thus, every minimal model will entail $\mathbf{A} \vee \mathbf{B} \vee \mathbf{C}$, but there will be three models, entailing $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ respectively. Thus, $\mathbf{T} \not\models_{\mathbf{min}} \mathbf{A}$, $\mathbf{T} \not\models_{\mathbf{min}} \mathbf{B}$, and $\mathbf{T} \not\models_{\mathbf{min}} \mathbf{C}$.
Given instead T',

$$\mathbf{A} \vee \mathbf{B} \vee \mathbf{C}$$

$$\mathbf{B} \vee \mathbf{C}$$

since the first disjunction is redundant with respect to the second one, every minimal model will entail $\neg\mathbf{A}$, and thus $\mathbf{T}' \models_{\mathbf{min}} \neg\mathbf{A}$.

Lemma 18.
If for every axiom in $\mathbf{T}$ *of the form* $\hat{\mathcal{C}}\mathbf{G} \supset \mathbf{F}$, $\exists \mathcal{B}\mathbf{L} \in \hat{\mathcal{C}}\mathbf{G} \ : \ \mathbf{T} \not\models \mathcal{B}\mathbf{L}$ *then* $\mathbf{T} \not\models_{\mathbf{min}} \mathbf{F}$ *and* $\mathbf{T} \not\models_{\mathbf{min}} \neg\mathbf{F}$

Proof. If we consider such an axiom, by varying the truth value of $\mathcal{B}\mathbf{L}$, $\mathbf{F}$ becomes alternatively true or false. Thus, neither $\mathbf{F}$ nor its negation may belong to all minimal models. □

Theorem 19.
For any formula F and any affirmative belief theory $\mathbf{T}$, $\mathbf{T} \models_{\mathbf{min}} \mathbf{F}$ *iff one of the following cases holds:*

(i) $\mathbf{F}$ *is an atom or a disjunction, and there exists an axiom* $\hat{\mathcal{C}}\mathbf{G} \supset \mathbf{F}$ *in* $\mathbf{T}$ *where* $\mathbf{T} \models \hat{\mathcal{C}}\mathbf{G}$
(ii) $\mathbf{F} = \neg\mathbf{A}$, *where* $\mathbf{A}$ *is an objective atom to which Lemma 17 does not apply, and for every axiom* $\hat{\mathcal{C}}\mathbf{G} \supset \mathbf{A}$ *in* $\mathbf{T}$ *there exists an objective formula* $\mathbf{R} \in \hat{\mathcal{C}}\mathbf{G}$ *such that* $\mathbf{T} \not\models \mathbf{R}$.
(iii) $\mathbf{F} = \neg\mathbf{A}$ *where* $\mathbf{A}$ *is an objective atom to which Lemma 17 does not apply, and there is no axiom* $\mathcal{C}\mathbf{G} \supset \mathbf{A}$ *in* $\mathbf{T}$.
(iv) $\mathbf{F} = \neg\mathbf{A}$ *where* $\mathbf{A}$ *is an objective atom, (i) does not apply to* $\mathbf{A}$, *and* $\mathbf{T} \models_{\mathbf{min}} \mathcal{D}\mathbf{D}$, $\mathbf{T} \models_{\mathbf{min}} \mathcal{D}\mathbf{H}$, $\mathbf{H} \subseteq \mathbf{D}$, $\mathbf{A} \in \mathbf{D}$, $\mathbf{A} \notin \mathbf{H}$.
(v) $\mathbf{F}$ *is a negative disjunction* $\overline{\mathcal{D}}\mathbf{D}$, $\mathbf{T} \models_{\mathbf{min}} \mathcal{D}\mathbf{D}$ *and* $\exists \mathbf{L} \in \mathcal{D}\mathbf{D}$, $\mathbf{T} \not\models_{\mathbf{min}} \mathbf{L}$.
(vi) $\mathbf{F}$ *is a logical consequence of formulae which fit in cases (i)-(v).*

Proof. Case (i) is fairly obvious. In cases (ii)-(iv), if Lemma 17 does not apply we can conclude that every minimal model will entail $\neg\mathbf{A}$, since it does not entail $\mathbf{A}$, and it is total. Case (v) holds because

$$\neg(\mathbf{A} \wedge \mathbf{B}) \equiv (\neg\mathbf{A} \vee \neg\mathbf{B})$$

is a tautology of first-order propositional logic. Case (vi) is also obvious. All the formulae not considered in cases (i)-(vi) are not minimally entailed since they are not necessary for satisfying axioms in $\mathbf{T}$, and are not logical consequences of those which are necessary. □

In [Prz94], the static completion is shown to be the least fixed point of a monotonic minimal model operator, based on the belief closure operator $\Psi_{\mathbf{T}}$, so that it can be iteratively constructed starting from **P**. In the following sections, we propose an analogous characterization, given by splitting the construction into two parts: a preliminary part of simplification of the program w.r.t. disjunction, and then a constructive part based on a variant of the well-founded model computation.

5 Program Transformation

The objective of the program transformation procedure described in this section is that of identifying and making explicit all potential deductions which can be performed, according to a given semantics, by means of the disjunctive knowledge represented in the program. The result is a new program **P**' (containing **P** as a proper subprogram) where disjunction may occur also in the conditions of the axioms. Thus, the definition of the procedure formalizes the rules for using disjunction in the inference process.

For the static semantics, that we are considering as a case-study, a disjunction of literals is transposed, by necessitation (axiom (N)), into a single belief atom, which is then used in derivations according to the usual first-order-logic rules. In the static semantics, in fact, it is not assumed that the belief operator is distributive with respect to disjunctions. It is however possible, as we will see, to optionally introduce the disjunctive belief axiom.

Following [Prz94], and as summarized in the previous section, we consider every clause in a disjunctive logic program **P** to be translated into a corresponding axiom of an affirmative belief theory, of the form 2. Therefore, we are able to use all features of first-order predicate logic (including moving literals from one side of the implication to the other, and introducing disjunctions in the conditions). Since we will not explicitly use this notation, please recall that **notA** means $\mathcal{B}\neg\mathbf{A}$. All disjunctions in the conditions of clauses will be enclosed in brackets, to indicate that they correspond to a single belief atom. Axioms in **P** and **P**' will be also called, by abuse of notation, *clauses*.

As shown in Example 2, in the context of theories which are the translation of disjunctive logic programs, a way for deriving new disjunctions is that of using tautologies of propositional logic. Useful tautologies concerning disjunction can be represented by means of the following axiom schemata. Let **Q**, **R**, **S**, **W** be objective atoms, **D** be a disjunction of objective atoms, and **C** any conjunction.

$$((\mathbf{C} \wedge \mathbf{W} \supset \mathbf{Q}) \supset (\mathbf{C} \wedge (\mathbf{W} \vee \mathbf{S})) \supset (\mathbf{Q} \vee \mathbf{S})) \tag{4}$$

$$((\mathbf{C} \supset \mathbf{D}) \supset (\mathbf{C} \wedge \mathbf{W} \supset (\mathbf{D} \vee \mathbf{S})) \tag{5}$$

The purpose of axiom 4 is that of introducing disjunctions in the conditions of clauses, so as to derive new disjunctions from those which are conclusions of some other clause in **P**. Axiom 5, insted, has the aim of eliminating redundant clauses.

Proposition 20.
Axiom schema 4 represents all possible derivations of disjunction from other disjunctions.

Proof. This is easy to see, since this axiom potentially introduces in the conditions and conclusion of clauses all possible disjunctions over the language of the program. □

In order to generate only the interesting clauses (in the sense specified in Section 4), we also introduce the following restricted form of axiom schema 4.

$$((\mathbf{C} \land \mathbf{W} \supset \mathbf{Q}) \land (\mathbf{C} \land \mathbf{R} \supset \mathbf{S}) \supset (\mathbf{C} \land (\mathbf{W} \lor \mathbf{R})) \supset (\mathbf{Q} \lor \mathbf{S})) \tag{6}$$

Axiom 6 joins two clauses, thus obtaining a new one with disjunctive conditions and conclusion.

The program transformation procedure performs a controlled application of the above schemata to the given program, so as to generate interesting axioms only. To do this, the application takes as basis disjunctions $\mathcal{D}\mathbf{D}$ appearing as the conclusion of a clause in $\mathbf{P}$, and their negation $\sim\mathcal{D}\mathbf{D}$. With this limitation, only the significant subset of the static completion is computed in the constructive step. Other limitations are introduced to avoid the generation of redundant clauses.

The program transformation procedure for the static semantics consists in repeatedly applying the steps defined below in the given order, each step being repeated until no further transformation is possible. Each transformation will be first illustrated by means of examples, and then defined formally.

Initial Step

The aim of this step is that of relating disjunctions expressed explicitly with disjunctions expressed implicitly, by applying axiom schema 6.

Example 4.
Given clauses:

$$\begin{aligned} \mathbf{a} &\leftarrow \mathbf{b}, \mathbf{g} \\ \mathbf{a} &\leftarrow \mathbf{c}, \mathbf{h} \\ \mathbf{b} \lor \mathbf{c} &\leftarrow \mathbf{e}, \mathbf{f} \end{aligned}$$

add axiom:

$$\mathbf{a} \leftarrow (\mathbf{b} \lor \mathbf{c}), \mathbf{g}, \mathbf{h}$$

Example 5.
Similarly to the previous example, given clauses:

$$\begin{aligned} &\mathbf{a} \lor \mathbf{b} \\ &\quad \mathbf{c} \leftarrow \mathbf{a},\ \mathbf{e} \\ &\quad \mathbf{d} \leftarrow \mathbf{b},\ \mathbf{not}\, \mathbf{f} \end{aligned}$$

add axiom:

$$\mathbf{c \vee d \leftarrow (a \vee b), e, not\, f}$$

Formally, this transformation step is defined as follows.

Definition 21.
Given clauses:

$$\begin{aligned} \mathcal{D}\mathbf{D}_1 &\leftarrow \mathcal{C}\mathbf{B}_1, \overline{\mathcal{C}}\mathbf{C}_1 \\ &\dots \\ \mathcal{D}\mathbf{D_n} &\leftarrow \mathcal{C}\mathbf{B_n}, \overline{\mathcal{C}}\mathbf{C_n} \\ \mathcal{D}\mathbf{T} &\leftarrow \mathcal{C}\mathbf{H}, \overline{\mathcal{C}}\mathbf{K} \end{aligned}$$

if $\forall \mathcal{C}\mathbf{B_i}\ \ \mathbf{i} = 1, \dots, \mathbf{n}\ \ \exists\, \mathbf{A_i} \in \mathcal{C}\mathbf{B_i}$ such that $\mathbf{A_i} \in \mathcal{D}\mathbf{T}$, then:
let $\mathcal{C}\mathbf{B'_i} = \bigcup_{\mathbf{i}=1}^{\mathbf{n}} (\mathcal{C}\mathbf{B_i} - \mathbf{A_i})$;
let $\overline{\mathcal{C}}\mathbf{C'_i} = \bigcup_{\mathbf{i}=1}^{\mathbf{n}} \overline{\mathcal{C}}\mathbf{C_i}$;
let $\mathcal{D}\mathbf{D'} = \bigcup_{\mathbf{i}=1}^{\mathbf{n}} \mathcal{D}\mathbf{D_i}$;
let $\mathcal{D}\mathbf{A} = \mathbf{A}_1 \vee \dots \vee \mathbf{A_n}$;
add axiom:

$$\mathcal{D}\mathbf{D'} \leftarrow \mathcal{D}\mathbf{A}, \mathcal{C}\mathbf{B'}, \overline{\mathcal{C}}\mathbf{C'}$$

As a special case, we may have $\mathcal{D}\mathbf{D}_1 = ... = \mathcal{D}\mathbf{D_n}$.

An analogous transformation involving negative literals allows us to optionally introduce the disjunctive belief axiom.

Example 6.
Given clauses:

$$\begin{aligned} \mathbf{a} &\leftarrow \mathbf{not\, b} \\ \mathbf{a} &\leftarrow \mathbf{not\, c} \\ \mathbf{b \vee c} &\leftarrow \mathbf{e, f} \end{aligned}$$

the axiom:

$$\mathbf{a \leftarrow (not\, b \vee not\, c)}$$

is an application of the disjunctive belief axiom. In fact, in terms of belief theories, the axiom schema which has been applied in this case is:

$$((\mathcal{B}\neg\mathbf{b} \supset \mathbf{a}) \wedge (\mathcal{B}\neg\mathbf{c} \supset \mathbf{a})) \supset (\mathcal{B}(\neg\mathbf{b} \vee \neg\mathbf{c}) \supset \mathbf{a})$$

which is equivalent to

$$((\mathcal{B}\neg\mathbf{b} \vee \mathcal{B}\neg\mathbf{c}) \supset \mathbf{a})) \supset (\mathcal{B}(\neg\mathbf{b} \vee \neg\mathbf{c}) \supset \mathbf{a})$$

Notice that, in performing derivations from the program $\mathbf{P}$, the disjunctive belief axiom makes sense only if applied to negative atoms, which are the only belief atoms in the program. An example of application is shown in Section 6.

Formally, we have the following

Definition 22.
Given the same clauses as in previous definition,
if $\forall \overline{\mathcal{C}}\mathbf{C_i}$ $\mathbf{i} = 1, \ldots, \mathbf{n}$ $\exists\, \mathbf{not A_i} \in \overline{\mathcal{C}}\mathbf{C_i}$ such that
$\mathbf{not A_i} \in {\sim}\mathcal{D}\mathbf{T}$, then:
let $\overline{\mathcal{C}}\mathbf{C'_i} = \overline{\mathcal{C}}\mathbf{C_i} - \mathbf{not A_i}$;
let $\mathcal{C}\mathbf{P} = \bigcup_{\mathbf{i=1}}^{\mathbf{n}} \mathcal{C}\mathbf{B_i}$ and $\overline{\mathcal{C}}\mathbf{Q} = \bigcup_{\mathbf{i=1}}^{\mathbf{n}} \overline{\mathcal{C}}\mathbf{C'_i}$;
let $\mathcal{D}\mathbf{D'} = \bigcup_{\mathbf{i=1}}^{\mathbf{n}} \mathcal{D}\mathbf{D_i}$;
let $\overline{\mathcal{D}}\mathbf{D} = \mathbf{not A_1} \vee \ldots \vee \mathbf{not A_n}$;
add axiom:

$$\mathcal{D}\mathbf{D'} \leftarrow \overline{\mathcal{D}}\mathbf{D}, \mathcal{C}\mathbf{P}, \overline{\mathcal{C}}\mathbf{Q}$$

Intermediate Step

This step carefully applies axiom schema 4, so as to exploit disjunctions which appear as the conclusion of some clause in $\mathbf{P}$.

Example 7.
Consider the following clauses.

$$\mathbf{t} \leftarrow \mathbf{a}, \mathbf{f}$$
$$\mathbf{a} \vee \mathbf{b} \leftarrow \mathbf{p}, \mathbf{not\, q}$$

add axiom:

$$\mathbf{t} \vee \mathbf{b} \leftarrow (\mathbf{a} \vee \mathbf{b}), \mathbf{f}$$

The formal definition of this transformation is the following.

Definition 23.
Given clauses:

$$\mathcal{D}\mathbf{A} \leftarrow \mathcal{C}\mathbf{B}, \overline{\mathcal{C}}\mathbf{C}$$
$$\mathcal{D}\mathbf{D} \leftarrow \mathcal{C}\mathbf{P}, \overline{\mathcal{C}}\mathbf{Q}$$

if $\mathcal{D}\mathbf{A} \not\subseteq \mathcal{D}\mathbf{D}$ and $\exists\, \mathbf{A} \in \mathcal{C}\mathbf{B}$ such that $\mathbf{A} \in \mathcal{D}\mathbf{D}$, then:
let $\mathcal{C}\mathbf{B'} = \mathcal{C}\mathbf{B} - \{\mathbf{A}\}$;
let $\mathcal{D}\mathbf{D'} = \mathcal{D}\mathbf{D} - \{\mathbf{A}\}$;
let $\mathcal{D}\mathbf{A'} = \mathcal{D}\mathbf{A} \cup \mathcal{D}\mathbf{D'}$;
add axiom:

$$\mathcal{D}\mathbf{A'} \leftarrow \mathcal{D}\mathbf{D}, \mathcal{C}\mathbf{B'}, \overline{\mathcal{C}}\mathbf{Q}$$

The limitation $\mathcal{D}\mathbf{A} \not\subseteq \mathcal{D}\mathbf{D}$ is aimed at avoiding redundancies, as exemplified below.

Example 8.
Given clauses:
$\mathbf{a} \vee \mathbf{b} \vee \mathbf{c}$
$\mathbf{c} \vee \mathbf{b} \leftarrow \mathbf{c}$
it is useless to generate the axiom:
$\mathbf{a} \vee \mathbf{b} \vee \mathbf{c} \leftarrow (\mathbf{a} \vee \mathbf{b} \vee \mathbf{c})$

Example 9.
Consider the following clauses.

$$\mathbf{g} \leftarrow \mathbf{k}, \mathbf{not\,c}$$
$$\mathbf{c} \vee \mathbf{d} \leftarrow \mathbf{m}, \mathbf{not\,n}$$

The situation is similar to previous example, but the condition $\mathbf{not\,c}$ in the first clause corresponds to the negation of atom $\mathbf{c}$, that appears in the disjunction $\mathbf{c} \vee \mathbf{d}$, which is the conclusion of the second clause. In this case, add axiom:

$$\mathbf{g} \vee \mathbf{not\,d} \leftarrow \mathbf{k}, (\mathbf{not\,c} \vee \mathbf{not\,d})$$

Formally, we have

Definition 24.
Given the same clauses as in previous definition,
if $\mathcal{D}\mathbf{A} \not\subseteq \mathcal{D}\mathbf{D}$ and $\exists\, \mathbf{not A} \in \overline{\mathcal{C}}\mathbf{C}$ such that $\mathbf{not A} \in \sim\mathcal{D}\mathbf{D}$, then:
let $\overline{\mathcal{C}}\mathbf{C}' = \overline{\mathcal{C}}\mathbf{C} - \{\mathbf{not A}\}$;
let $\overline{\mathcal{D}}\mathbf{D} = \sim\mathcal{D}\mathbf{D}$;
let $\overline{\mathcal{D}}\mathbf{D}' = \overline{\mathcal{D}}\mathbf{D} - \{\mathbf{not A}\}$; let $\hat{\mathcal{D}}\mathbf{E} = \mathcal{D}\mathbf{A} \vee \overline{\mathcal{D}}\mathbf{D}'$
add axiom:

$$\hat{\mathcal{D}}\mathbf{E} \leftarrow \overline{\mathcal{D}}\mathbf{D}, \mathcal{C}\mathbf{B}, \overline{\mathcal{C}}\mathbf{C}'$$

Again, the limitation $\mathcal{D}\mathbf{A} \not\subseteq \mathcal{D}\mathbf{D}$ is aimed at avoiding redundancies, such as the following.

Example 10.
Given clauses:
$\mathbf{a} \vee \mathbf{b} \vee \mathbf{c}$
$\mathbf{a} \vee \mathbf{b} \leftarrow \mathbf{not\,c}$
it is useless to generate the axiom:
$\mathbf{a} \vee \mathbf{b} \vee \mathbf{not\,a} \vee \mathbf{not\,b} \leftarrow (\mathbf{not\,a} \vee \mathbf{not\,b} \vee \mathbf{not\,c})$

Final Step
The aim is to eliminate redundant axioms, i.e. those axioms whose head and body are included in that of another (according to axiom schema 5).

Example 11.
Consider the following axioms.

$$\mathbf{a} \vee \mathbf{b} \leftarrow \mathbf{f}, \mathbf{g}$$
$$\mathbf{a} \vee \mathbf{b} \vee \mathbf{c} \leftarrow \mathbf{f}, \mathbf{g}, \mathbf{h}$$

The second axiom can be removed, since its conclusion is a trivial consequence of the conclusion of the first one.

Formally, we have

Definition 25.
Let $\mathbf{C}1$ be the axiom

$$\mathcal{D}\mathbf{A} \leftarrow \mathcal{C}\mathbf{B}, \overline{\mathcal{C}}\mathbf{C}$$

and $\mathbf{C}2$ be the axiom

$$\mathcal{D}\mathbf{F} \leftarrow \mathcal{C}\mathbf{G}, \overline{\mathcal{C}}\mathbf{H}$$

If $\mathcal{D}\mathbf{A} \subseteq \mathcal{D}\mathbf{F}$ and $\mathcal{C}\mathbf{B} \subseteq \mathcal{C}\mathbf{G}$ and $\overline{\mathcal{C}}\mathbf{C} \subseteq \overline{\mathcal{C}}\mathbf{H}$, then remove $\mathbf{C}2$.

In the rest of this paper, when considering one of the following: the disjunctive logic program $\mathbf{P}$; the new program $\mathbf{P}'$ obtained by applying the transformation procedure defined above; the consequences of **P**'; the results of the constructive phase; we will say that they *correspond* to an affermative belief theory (or more generally to a set of formulae of $\mathcal{L}_{\mathcal{B}}$) in the sense that: every atom **not A** must be read as $\mathcal{B}\neg\mathbf{A}$ and every disjunction $\mathbf{A} \vee \mathbf{B}$ in the conditions of axioms must be read read as $\mathcal{B}(\mathbf{A} \vee \mathbf{B})$. By abuse of notation, we indicate with the same names **P** and **P**' also the corresponding belief theories. By "model" we mean a model of a belief theory, as defined in Section 3.

Proposition 26.
P' *corresponds to an affirmative belief theory.*

Proof. From the definition of the program transformation steps, it is easy to see that the resulting axioms are still of the form 2. □

Theorem 27.
$\mathbf{M}$ *is a model of* $\mathbf{P}$ *iff* $\mathbf{M}$ *is a model of* $\mathbf{P}'$.

Proof. The axioms added to $\mathbf{P}$ by the program transformation procedure are logical consequences of $\mathbf{P}$, and therefore adding them explicitly does not change the models of the theory. □

Theorem 28.
$\mathbf{Cn}_*(\mathbf{P}) = \mathbf{Cn}_*(\mathbf{P}')$

Proof. The consequences obtained from the axioms of $\mathbf{P}'$ added by the program transformation procedure can be derived directly ¿from $\mathbf{P}$, given the tautologies of propositional logic. Therefore, the overall set of consequences is the same. □

Corollary 29.
The static completion of $\mathbf{P}$ *is the same as the static completion of* $\mathbf{P}'$.

Proof. By the definition of belief closure operator, since the set of minimal models and the consequences $\mathbf{Cn}_*$ are the same for $\mathbf{P}$ and $\mathbf{P}'$, the fixed points of the belief closure operators $\Psi_{\mathbf{P}}$ and and $\Psi_{\mathbf{P}'}$ must also be the same. □

The previous results imply that we can indifferently consider the minimal models and the static completion of $\mathbf{P}$ and $\mathbf{P}'$. In particular, we are entitled to compute the static completion of $\mathbf{P}$ by means of the transformed program $\mathbf{P}'$.

6 Iterated Fixpoint Computation

The static semantics can be computed by applying to the program $\mathbf{P}'$, resulting from the above-defined transformation procedure, a variation of the construction summarized in Section 2. In particular, it is only necessary to modify the base step, while the rest of the procedure remains the same.

Preliminarly, we assume that the Herbrand base of $\mathbf{P}'$ contains (as new atoms): all negative literals appearing in $\mathbf{P}'$; all disjunctions appearing in $\mathbf{P}'$; and the negation of each disjunction appearing in $\mathbf{P}'$. This extended Herbrand base will be denoted by $\mathbf{B}^{\mathbf{C}}_{\mathbf{P}}$. In this way, the program $\mathbf{P}$' can be treated, basically, as a definite Horn-clause program.

Notice that any set $\mathbf{Z} \subseteq \mathbf{B}^{\mathbf{C}}_{\mathbf{P}}$ corresponds to an affirmative belief theory composed of facts. Again by the abuse of notation, this affirmative belief theory will be called $\mathbf{Z}$.

Proposition 30. *Given a set* $\mathbf{Z} \subseteq \mathbf{B}^{\mathbf{C}}_{\mathbf{P}}$, *the logical consequences of* $\mathbf{P}' \cup \mathbf{Z}$ *correspond to the interesting subset of* $\mathbf{Cn}_*(\mathbf{P} \cup \mathbf{Z})$.

Proof. The program transformation procedure has added to $\mathbf{P}'$ new axioms which simulate the application of axioms (K) and (N), taking (D) into account. All the belief formulae in $\mathbf{P}$' and $\mathbf{Z}$ are represented as atoms of $\mathbf{B}^{\mathbf{C}}_{\mathbf{P}}$. The axioms are formulated so as to derive interesting disjunctions. Thus, this corresponds to applying $\mathbf{Cn}_*$ to $\mathbf{P} \cup \mathbf{Z}$. □

Proposition 31. *Given a set* $\mathbf{Z} \subseteq \mathbf{B}^{\mathbf{C}}_{\mathbf{P}}$, $\mathbf{Cn}_*(\mathbf{P} \cup \mathbf{Z})$ *is equivalent to the least Herbrand model of* $\mathbf{P} \cup \mathbf{Z}$.

Proof. This holds because, in the extended Herbrand base, $\mathbf{P} \cup \mathbf{Z}$ (or, equivalently, $\mathbf{P}' \cup \mathbf{Z}$) is a definite program. □

For the static semantics, in order to compute what is true in *all* minimal models of a belief theory corresponding to the translation of a disjunctive logic program, it is necessary to consider the requirements stated in Theorem 19. Since the steps of the iterated fixpoint procedure start from a current set of beliefs, which is an approximation of the final one, some of the requirements can be taken into account also in the base step.

Definition 32 Modified Base Step.
Let $\mathbf{A}, \mathbf{notA}, \mathcal{D}\mathbf{D}, \overline{\mathcal{D}}\mathbf{D} \in \mathbf{B_P^C}$. Let $\mathbf{J} \subseteq \mathbf{B_P^C}$.

$$
\begin{aligned}
\mathbf{T_J} \uparrow 0 \quad = & \\
& \textbf{(1)}\ \{\ \mathbf{notA} : \mathbf{A} \notin \mathbf{J} \\
& \qquad \text{and } \forall\, \mathcal{D}\mathbf{D} \text{ such that } \mathbf{A} \in \mathcal{D}\mathbf{D}\ \ \mathcal{D}\mathbf{D} \notin \mathbf{J} \\
& \qquad \text{and } \forall\, \overline{\mathcal{D}}\mathbf{D} \text{ such that } \mathbf{notA} \in \overline{\mathcal{D}}\mathbf{D}\ \ \overline{\mathcal{D}}\mathbf{D} \notin \mathbf{J}\} \quad \bigcup \\
& \textbf{(2)}\ \{\ \mathbf{notA} : \mathbf{A} \notin \mathbf{J} \\
& \qquad \text{and } \forall\, \mathcal{D}\mathbf{D} \text{ such that } \mathbf{A} \in \mathcal{D}\mathbf{D} \\
& \qquad \mathcal{D}\mathbf{D}' = \mathcal{D}\mathbf{D} - \{\mathbf{A}\} \in \mathbf{J}\} \quad \bigcup \\
& \textbf{(3)}\ \{\ \overline{\mathcal{D}}\mathbf{D} = \sim\mathcal{D}\mathbf{D} \text{ such that} \\
& \qquad \exists\, \mathbf{A} \in \mathcal{D}\mathbf{D}, \mathbf{A} \notin \mathbf{J}\}
\end{aligned}
$$

Lemma 33.
Let $\mathbf{J} \subseteq \mathbf{B_P^C}$, and let $\mathbf{J}$ be consistent. Let $\mathbf{F}$ be a negative literal or a negative disjunction. $\mathbf{T_J} \uparrow 0$ corresponds to the set $\{\mathcal{B}\mathbf{F}\ : \mathbf{J} \models_{\mathbf{min}} \mathbf{F}\}$.

Proof. $\mathbf{J}$ corresponds to an affirmative belief theory composed of facts. Points (1)-(3) of Definition 32 exactly correspond to points (iii)-(v) of Theorem 32, which are those applicable to this kind of theory. □

We will still call Σ the function resulting from Definitions 1 and 2 by substituting, in Definition 1, the original base step with the modified base step.

Theorem 34.
Σ is monotonically increasing, and therefore for some ordinal δ:

$$\Sigma \uparrow \delta = \Sigma \uparrow \delta + 1 = \mathbf{WFD}$$

The proof is like those in [Gel93] and [SC94], since the modified base step does not actually affect this point.

In the following, given $\mathbf{Z} \subseteq \mathbf{B_P^C}$, the comparison between $\Sigma(\mathbf{Z})$ and $\Psi_\mathbf{P}(\mathbf{P} \cup \mathbf{Z})$ (which is equal, by the results given in previous section, to $\Psi_{\mathbf{P}'}(\mathbf{P}' \cup \mathbf{Z})$) is meant with respect to the interesting subset of $\Psi_\mathbf{P}(\mathbf{P} \cup \mathbf{Z})$.

Theorem 35.
Let $\mathbf{Z} \subseteq \mathbf{B_P^C}$, $\mathbf{Z} \subseteq \overline{\mathbf{P}}$. $\mathbf{P} \cup \Sigma(\mathbf{Z}) = \Psi_\mathbf{P}(\mathbf{P} \cup \mathbf{Z})$.

Proof. By the definition of Σ,

$$\Sigma(\mathbf{Z}) = \mathbf{S}(\mathbf{S}(\mathbf{Z})) = \mathbf{T_Z} \uparrow \omega$$

By Proposition 31 and Lemma 33,

$$\mathbf{S}(\mathbf{Z}) = \mathbf{Cn}_*(\mathbf{P} \cup \{\mathcal{B}\mathbf{F}\ : \mathbf{Z} \models_{\mathbf{min}} \mathbf{F}\}$$

where F is either a negative literal or a negative disjunction. Notice, however, that these belief atoms (beliefs about negative formulae) are the only interesting ones in this context: in fact, they are the only belief atoms explicitly present in the clauses of **P**'.

$\mathbf{S}(\mathbf{Z})$ is an overestimated, possibly inconsistent set of consequences. In fact, it will contain both **A** and **not A** any time that $\mathbf{Z} \models_{\mathbf{min}} \mathbf{not\,A}$, but **A** is instead a consequence of $\mathbf{P} \cup \{\mathbf{not\,A}\}$. When computing $\mathbf{S}(\mathbf{S}(\mathbf{Z}))$, however, these inconsistent conclusions will be excluded, since point (1) of the modified base step will exclude **not A**. Then, the modified base step will compute the negative formulae **F** that are minimally entailed by the consistent subset of S(Z).

Since $\mathbf{Z} \subseteq \overline{\mathbf{P}}$, this corresponds to finding the formulae **F** that are minimally entailed by $\mathbf{P} \cup \mathbf{Z}$. Thus, by applying again the definition of Σ and Proposition 31, we have

$$\mathbf{P} \cup \mathbf{S}(\mathbf{S}(\mathbf{Z})) = \mathbf{P} \cup \Sigma(\mathbf{Z}) = \mathbf{Cn}_*(\mathbf{P} \cup \{\mathcal{B}\mathbf{F} \ : \mathbf{P} \cup \mathbf{Z} \models_{\mathbf{min}} \mathbf{F}\}$$

which is exactly $\Psi_{\mathbf{P}}(\mathbf{P} \cup \mathbf{Z})$. □

Corollary 36.
$\Sigma(\mathbf{Z}) \subseteq \overline{\mathbf{P}}$.

Proof. The result follows from the above Theorem, and from Proposition 16. □

Lemma 37.
$\mathbf{P} \cup (\Sigma \uparrow \mathbf{n} + 1) = \mathbf{P} \cup (\Sigma(\Sigma \uparrow \mathbf{n})) = \Psi_{\mathbf{P}}(\mathbf{P} \cup (\Sigma \uparrow \mathbf{n}))$.

Proof. Since $\Sigma \uparrow 0 = \emptyset$, and clearly $\emptyset \subseteq \overline{\mathbf{P}}$, by Theorem 35 $\mathbf{P} \cup (\Sigma \uparrow 1) = \Psi_{\mathbf{P}}(\mathbf{P} \cup (\Sigma \uparrow 0))$. The thesis easily follows by induction, assuming that $\Sigma \uparrow \mathbf{n} \subseteq \overline{\mathbf{P}}$, and applying Theorem 35. □

Corollary 38.
The sequence $\Psi_{\mathbf{P}}(\mathbf{P} \cup \Sigma \uparrow \mathbf{n})$ *is monotonically increasing, and thus has a least fixed point.*

Proof. It is an immediate consequence of Theorem 34 and Theorem 35. □

Corollary 39.
The fixed point of the sequence $\Psi_{\mathbf{P}}(\mathbf{P} \cup \Sigma \uparrow \mathbf{n})$ *is the least fixed point of the operator* $\Psi_{\mathbf{P}}$.

Proof. This follows from Corollary 36, since the fixed point of the sequence $\Psi_{\mathbf{P}}(\mathbf{P} \cup \Sigma \uparrow \mathbf{n})$ is reached in correspondence of the least fixed point $\Sigma \uparrow \delta$ of the sequence $\Sigma \uparrow \mathbf{n}$. In fact, by Theorem 35, $\mathbf{P} \cup \Sigma \uparrow \delta = \Psi_{\mathbf{P}}(\mathbf{P} \cup \Sigma \uparrow \delta)$ □

Therefore, we have proved the correspondence between the (set of consequences of) **WFD** and the static completion. In the next section we show this correspondence with respect to some examples.

7 Examples

In this Section we give some examples of the proposed approach. First, we consider two of the sample programs discussed in [Prz94] so as to show that the proposed constructive approach gives the same results. Then, we propose more complex new examples, in order to better illustrate the program transformation procedure.

Example 1. ([Prz94, Example 5.2] Let *wp1* stand for *Write_Paper_1*, *wp2* stand for *Write_Paper_2*, *gc* stand for *Get_Crazy*, *gf* stand for *Get_Fired*, and let the given program **P** be the following.

$$\begin{aligned}&\mathbf{wp1} \vee \mathbf{wp2}\\ &\qquad\mathbf{gc} \leftarrow \mathbf{wp1},\ \mathbf{wp2}\\ &\qquad\mathbf{gf} \leftarrow \mathbf{not\,wp1},\ \mathbf{not\,wp2}\end{aligned}$$

The program transformation procedure does not modify the program. The computation of the static completion by the proposed extended well-founded computation is the following.

$$\begin{aligned}
\Sigma \uparrow \mathbf{0} \quad &= \emptyset\\
\mathbf{S}(\emptyset) \quad &= \{\mathbf{not\,wp1} \vee \mathbf{not\,wp2}, \mathbf{not\,wp1}, \mathbf{not\,wp2}, \mathbf{not\,gc}, \mathbf{not\,gf},\\
&\quad \mathbf{wp1} \vee \mathbf{wp2}, \mathbf{gf}\}\\
\Sigma \uparrow \mathbf{1} \quad &= \mathbf{S}(\mathbf{S}(\emptyset)) = \{\mathbf{not\,wp1} \vee \mathbf{not\,wp2}, \mathbf{not\,gc},\\
&\quad \mathbf{wp1} \vee \mathbf{wp2}\}\\
\mathbf{S}(\Sigma \uparrow 1) &= \{\mathbf{not\,wp1} \vee \mathbf{not\,wp2}, \mathbf{not\,gc}, \mathbf{not\,gf},\\
&\quad \mathbf{wp1} \vee \mathbf{wp2}\}\\
\Sigma \uparrow \mathbf{2} \quad &= \mathbf{S}(\mathbf{S}(\Sigma \uparrow 1)) = \mathbf{S}(\Sigma \uparrow 1)
\end{aligned}$$

Therefore, the fixpoint of the sequence is
$\mathbf{WFD} = \Sigma \uparrow 2 = \{\mathbf{not\,wp1} \vee \mathbf{not\,wp2}, \mathbf{not\,gc}, \mathbf{not\,gf}, \mathbf{wp1} \vee \mathbf{wp2}\}$. In fact, given the belief theory **T** corresponding to **P**, the static completion is
$\overline{\mathbf{T}} = \mathbf{Con}{*}\,(\mathbf{T} \cup \{\mathcal{B}(\neg\mathbf{wp1} \vee \neg\mathbf{wp2}), \mathcal{B}\neg\mathbf{gc}, \mathcal{B}\neg\mathbf{gf}, \mathcal{B}(\mathbf{wp1} \vee \mathbf{wp2})\})$.

Example 2. ([Prz94, Example 5.4] Let *w* stand for *Work*, *t* stand for *Tired*, *s* stand for *Sleep*, *p* stand for *Paid*, *u* stand for *Unhappy*, and let the given program **P** be the following.

$$\begin{aligned}&\mathbf{w} \vee \mathbf{t} \vee \mathbf{s}\\ &\qquad\mathbf{w} \leftarrow \mathbf{not\,t}\\ &\qquad\mathbf{s} \leftarrow \mathbf{not\,w}\\ &\qquad\mathbf{t} \leftarrow \mathbf{not\,s}\\ &\qquad\mathbf{u} \leftarrow \mathbf{w},\ \mathbf{not\,p}\\ &\qquad\mathbf{p}\end{aligned}$$

The program transformation procedure does not modify the program. The computation of the static completion is the following.

$$
\begin{aligned}
\Sigma\uparrow \mathbf{0} &= \emptyset\\
\mathbf{S}(\emptyset) &= \{\mathbf{not\,w} \vee \mathbf{not\,t} \vee \mathbf{not\,s}, \mathbf{not\,w}, \mathbf{not\,t}, \mathbf{not\,s}, \mathbf{not\,u}, \mathbf{not\,p},\\
&\quad \mathbf{w} \vee \mathbf{t} \vee \mathbf{s}, \mathbf{w}, \mathbf{t}, \mathbf{s}, \mathbf{u}, \mathbf{p}\}\\
\Sigma\uparrow \mathbf{1} &= \mathbf{S}(\mathbf{S}(\emptyset)) = \{\mathbf{not\,w} \vee \mathbf{not\,t} \vee \mathbf{not\,s},\\
&\quad \mathbf{w} \vee \mathbf{t} \vee \mathbf{s}, \mathbf{p}\}\\
\mathbf{S}(\Sigma\uparrow 1) &= \{\mathbf{not\,w} \vee \mathbf{not\,t} \vee \mathbf{not\,s}, \mathbf{not\,u}\\
&\quad \mathbf{w} \vee \mathbf{t} \vee \mathbf{s}, \mathbf{p}\}\\
\Sigma\uparrow \mathbf{2} &= \mathbf{S}(\mathbf{S}(\Sigma\uparrow 1)) = \mathbf{S}(\Sigma\uparrow 1)
\end{aligned}
$$

Therefore, the fixpoint of the sequence is
$\mathbf{WFD} = \Sigma\uparrow 1 = \{\mathbf{not\,w} \vee \mathbf{not\,t} \vee \mathbf{not\,s}, \mathbf{not\,u}, \mathbf{p}, \mathbf{w} \vee \mathbf{t} \vee \mathbf{s}\}$. In fact, given the belief theory $\mathbf{T}$ corresponding to $\mathbf{P}$, the static completion is
$\overline{\mathbf{T}} = \mathbf{Con}*\,(\mathbf{T} \cup \{\mathcal{B}(\neg\mathbf{w} \vee \neg\mathbf{t} \vee \neg\mathbf{s}), \mathcal{B}\neg\mathbf{u}, \mathcal{B}\mathbf{p}, \mathcal{B}(\mathbf{w} \vee \mathbf{t} \vee \mathbf{s})\})$.

It is easy to see that the correspondence between **WFD** and Static Completion also holds for [Prz94, Example 5.3].

Example 3. Let us reconsider the predicates in Example 1, together with: *gs* standing for *Get_Salary*, *gr* standing for *Get_Rich*, *gf* standing for *Get_Famous*, *gh* standing for *Get_Happy*, *ga* standing for *Get_Angry*. Let the given program **P** be the following.

$$
\begin{aligned}
&\mathbf{wp}1 \vee \mathbf{wp}2\\
\mathbf{gr} &\leftarrow \mathbf{wp}1\\
\mathbf{gf} &\leftarrow \mathbf{wp}2\\
\mathbf{gc} &\leftarrow \mathbf{gr},\ \mathbf{gf}\\
\mathbf{gh} &\leftarrow \mathbf{gr}\\
\mathbf{gh} &\leftarrow \mathbf{gf}\\
\mathbf{gs} &\leftarrow \mathbf{wp}1\\
\mathbf{gs} &\leftarrow \mathbf{wp}2\\
\mathbf{ga} &\leftarrow \mathbf{not\,wp}1\\
\mathbf{ga} &\leftarrow \mathbf{not\,wp}2
\end{aligned}
$$

The initial step of the program transformation procedure adds the axioms:

$$
\begin{aligned}
\mathbf{gs} &\leftarrow (\mathbf{wp}1 \vee \mathbf{wp}2)\\
\mathbf{gh} &\leftarrow (\mathbf{gr} \vee \mathbf{gf})
\end{aligned}
$$

The intermediate and final steps add the axioms:

$$
\begin{aligned}
\mathbf{gr} \vee \mathbf{wp}2 &\leftarrow (\mathbf{wp}1 \vee \mathbf{wp}2)\\
\mathbf{gf} \vee \mathbf{wp}1 &\leftarrow (\mathbf{wp}1 \vee \mathbf{wp}2)
\end{aligned}
$$

$$\mathbf{ga} \vee \mathbf{not\ wp}1 \leftarrow (\mathbf{not\ wp}1 \vee \mathbf{not\ wp}2)$$
$$\mathbf{ga} \vee \mathbf{not\ wp}2 \leftarrow (\mathbf{not\ wp}1 \vee \mathbf{not\ wp}2)$$

The computation of the static completion by the proposed procedure is the following.

$$\begin{array}{ll}
\Sigma \uparrow \mathbf{0} & = \emptyset \\
\mathbf{S}(\emptyset) & = \{\mathbf{not\ wp}1 \vee \mathbf{not\ wp}2, \mathbf{not\ gr} \vee \mathbf{not\ gf}, \\
& \quad \mathbf{not\ wp}1, \mathbf{not\ wp}2, \mathbf{not\ gr}, \mathbf{not\ gf}, \\
& \quad \mathbf{not\ gh}, \mathbf{not\ gc}, \mathbf{not\ gs}, \mathbf{not\ ga}, \\
& \quad \mathbf{gr} \vee \mathbf{wp}2, \mathbf{gf} \vee \mathbf{wp}1, \\
& \quad \mathbf{ga} \vee \mathbf{not\ wp}1, \mathbf{ga} \vee \mathbf{not\ wp}2, \\
& \quad \mathbf{wp}1 \vee \mathbf{wp}2, \mathbf{gr} \vee \mathbf{gf}, \\
& \quad \mathbf{gh}, \mathbf{gs}, \mathbf{ga}\} \\
\Sigma \uparrow \mathbf{1} & = \mathbf{S}(\mathbf{S}(\emptyset)) = \{\mathbf{not\ wp}1 \vee \mathbf{not\ wp}2, \mathbf{not\ gr} \vee \mathbf{not\ gf}, \mathbf{not\ gc}, \\
& \quad \mathbf{wp}1 \vee \mathbf{wp}2, \mathbf{gr} \vee \mathbf{gf}, \\
& \quad \mathbf{gr} \vee \mathbf{wp}2, \mathbf{gf} \vee \mathbf{wp}1, \\
& \quad \mathbf{ga} \vee \mathbf{not\ wp}1, \mathbf{ga} \vee \mathbf{not\ wp}2, \\
& \quad \mathbf{gh}, \mathbf{gs}\} \\
\mathbf{S}(\Sigma \uparrow 1) & = \{\mathbf{not\ wp}1 \vee \mathbf{not\ wp}2, \mathbf{not\ gr} \vee \mathbf{not\ gf}, \mathbf{not\ gc}, \\
& \quad \mathbf{wp}1 \vee \mathbf{wp}2, \mathbf{gr} \vee \mathbf{gf}, \\
& \quad \mathbf{gr} \vee \mathbf{wp}2, \mathbf{gf} \vee \mathbf{wp}1, \\
& \quad \mathbf{ga} \vee \mathbf{not\ wp}1, \mathbf{ga} \vee \mathbf{not\ wp}2, \\
& \quad \mathbf{gh}, \mathbf{gs}\} \\
\Sigma \uparrow \mathbf{2} & = \mathbf{S}(\mathbf{S}(\Sigma \uparrow 1)) = \mathbf{S}(\Sigma \uparrow 1)
\end{array}$$

Therefore, the fixpoint of the sequence is
$\mathbf{WFD} = \Sigma \uparrow 2$.

By applying the negative part of the initial step, it is possible to implement the *disjunctive belief axiom*

$$\mathcal{B}(\mathbf{F} \vee \mathbf{G}) \equiv \mathcal{B}\mathbf{F} \vee \mathcal{B}\mathbf{G}$$

In this case, in fact, we would add axiom

$$\mathbf{ga} \leftarrow \mathbf{not\ wp}1 \vee \mathbf{not\ wp}2$$

which would lead to adding conclusion **ga** to **WFD**.

Example 4. Let *gq* stand *Good_Quality*, *rf* stand for *Refunded*, *st* stand for *Satisfied*, *ba* stand for *Buy_Again*, and let the given program **P** be the following.

$$\begin{array}{r}
\mathbf{gq} \vee \mathbf{rf} \\
\mathbf{st} \leftarrow \mathbf{gq} \\
\mathbf{ba} \leftarrow \mathbf{st} \\
\mathbf{ba} \leftarrow \mathbf{rf}
\end{array}$$

The program transformation procedure (initial and intermediate steps) adds the axioms:

$$\mathbf{ba} \leftarrow \mathbf{st} \vee \mathbf{rf}$$
$$\mathbf{st} \vee \mathbf{rf} \leftarrow (\mathbf{gq} \vee \mathbf{rf})$$

The first new axiom is obtained by joining the third and fourth clauses, thus making explicit the disjunctive information that they express. The second new axiom is obtained by a derivation involving the first two clauses. It is easy to verify that:
$\mathbf{WFD} = \{\mathbf{not\,gq} \vee \mathbf{not\,rf}, \mathbf{not\,st} \vee \mathbf{not\,rf}, \mathbf{gq} \vee \mathbf{rf}, \mathbf{st} \vee \mathbf{rf}, \mathbf{ba}\}$

Thus, it is clear that any shop which satisfies the condition $\mathbf{st} \vee \mathbf{rf}$, represented in the first clause of the program, is a good shop, where a customer will buy again.

8 Concluding Remarks

The idea of applying program transformation techniques for characterizing/computing the semantics of normal and disjunctive programs is not new in itself. It has been used, for instance, in [Cos95] for characterizing the stable model semantics [MG88] for normal programs, and in [MM93] for computing the well-founded and stationary semantics for normal and disjunctive programs.

In this paper, however, a main aim has been that of abstracting from computational issues and trying to analyze and characterize the general features of any semantics of disjunctive programs. An analysis and a classification of semantics for disjunctive programs have been proposed in [Dix92] [JD94]. We are indebted to these papers, and the spirit of the present paper is very much the same.

The perspective of this proposal is twofold. On the one hand, as already mentioned, the proposed approach might constitute the formal foundation of efficient procedural semantics for disjunctive logic programs. On the other hand, it is aimed to be a step towards the definition of a general framework for comparing different related semantics, and for possibly specifying parametrized definitions of semantics, to be adapted to the application domain at hand. What can be parametrized is the following. First, the choice of the axiom schemata on which the program transformation procedure is based. This influences the treatment of disjunctive information: in fact, the set of conclusions potentially derivable by the axioms given in Section 5 can be restricted, or changed, according to some kind of reasoning principle. Second, the modified base step, which basically defines the relationship between disjunction and negation. Also in this case, some other reasoning principles could be adopted, instead of minimal entailment.

The proposal in the present form could be potentially extended to versions of the stable model semantics for disjunctive programs, by suitably adapting those procedures which compute the stable models on the basis of the well-founded model, such as [Cos95], [VS92].

References

[AVG90] J.S. Schlipf A. Van Gelder, K.A. Ross. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3), 1990. abstract in: Proc. of the Seventh ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 1988.

[Bar91] C. Baral. *Issues in knowledge representation: semantics and knowledge combination*. PhD thesis, Department of Computer Science, 1991.

[CB92] J. Minker C. Baral, J. Lobo. Generalized disjunctive well-founded semantics for logic programs. *Annals of Mathematics and Artificial Intelligence*, 5, 1992.

[Cos95] S. Costantini. Contributions to the stable model semantics of logic programs with negation. *Theoretical Computer Science*, forthcoming, 1995. abstract in: A. Nerode (ed.), Logic Programming and Non-Monotonic Reasoning, Proc. of the Second International Workshop, The MIT Press, 1993.

[Dix91] J. Dix. Classifying semantics of logic programs. In V. S. Subrahmanian A. Nerode, W. Marek, editor, *Proc. of the First International Workshop on Logic Programming and Non-Monotonic Reasoning*. The MIT Press, 1991. Washington D. C., July 1991.

[Dix92] J. Dix. Classifying semantics of disjunctive logic programs. In K. Apt, editor, *Logic Programming, Proc. of the 1992 Joint Conference and Symposium*. The MIT Press, 1992. Washington D. C., November 1991.

[Gel93] A. Van Gelder. The alternating fixpoint of logic programs with negation. *Journal of Computer and System Sciences*, 47(1), 1993. abstract in: Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1989.

[Got92] G. Gottlob. The complexity of propositional default reasoning under the stationary semantics. Technical Report CD-TR-92/42, Institut fur Information-Systeme, Technische University, A-1040 Wien, Austria, 1992.

[HP90] T. C. Przymusinski H. Przymusinska. Semantic issues in deductive databases and logic programs. In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence*. North Holland, Amsterdam, 1990.

[JD94] M. Müller J. Dix. An axiomatic framework for representing and characterizing semantics of disjunctive logic programs. In P. Van Hentenryck, editor, *Logic Programming, Proc. of the 11th International Conference*. The MIT Press, 1994. Santa Margherita Ligure, June 1994.

[JL92] A. Rajasekar J. Lobo, J. Minker. *Foundations of Disjunctive Logic Programming*. The MIT Press, 1992.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second, extended edition, 1987.

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

[MG88] V. Lifschitz M. Gelfond. The stable model semantics for logic programming. In R.A. Kowalski K. Bowen, editor, *Logic Programming, Proceedings of the Fifth Symposium*. The MIT Press, 1988.

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

[MM93] J. Dix M. Müller. Implementing semantics of disjunctive logic programs using fringes and abstract properties. In A. Nerode, editor, *Logic Programming and*

Non-Monotonic Reasoning, Proc. of the Second International Workshop. The MIT Press, 1993.

[Prz89] T. C. Przymusinski. Every logic program has a natural stratification and an iterated fixpoint model. In *Proc. of the Eight ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM Press, 1989. Philadelphia, Pennsylvania.

[Prz91] T. C. Przymusinski. Semantics of disjunctive logic programs and deductive databases. In Y. Masunega C. Delobel, M. Kifer, editor, *DOOD'91, Proc. of the 2nd International Conference on Deductive and Object-Oriented Databases*, Berlin, 1991. Springer-Verlag. Munich, Germany.

[Prz94] T. C. Przymusinski. Static semantics for normal and disjunctive logic programs. *the Annals of Mathematics and Artificial Intelligence*, ??, 1994. Special Issue on Disjunctive Programs.

[SC94] G.A. Lanzarone S. Costantini. Metalevel negation and non-monotonic reasoning. *Methods of Logic in Computer Science: An International Journal*, 1(1), 1994. abstract in: Proc. of the Workshop on Non-Monotonic Reasoning and Logic Programming, Austin, TX, November 1-2, 1990.

[Sch92] J.S. Schlipf. *A survey of complexity and undecidability results in logic programming*. In *Proc. of the Workshop on Structural Complexity and Recursion-Theoretic Methods in Logic Programming*, November 1992. held in conjunction to JICSLP'92, Washington, D. C.

[VS92] C.Vago V.S. Subrahmanian, D. Nau. Wfs + branch and bound = stable models. Technical Report CS-TR-2935 UMIACS-TR-92_82, University of Maryland, July 1992. submitted to IEEE Transactions on Knowledge and Data Engineering.

This article was processed using the LaTeX macro package with LLNCS style