Super Logic Programs

Stefan Brass
Computer Science Dept.
University of Hannover
Lange Laube 22,
D-30159 Hannover, Germany

Jürgen Dix
Computer Science Dept.
University of Koblenz
Rheinau 1,
D-56075 Koblenz, Germany

Teodor C. Przymusinski
Computer Science Dept.
University of California
Riverside,
CA 92521, USA

Abstract

Recently, considerable interest and research effort has been given to the problem of finding a suitable extension of the *logic programming paradigm* beyond the class of normal logic programs. In order to demonstrate that a class of programs can be justifiably called an *extension of logic programs* one should be able to argue that:

- the proposed *syntax* of such programs resembles the syntax of logic programs but it applies to a significantly broader class of programs;
- the proposed *semantics* of such programs constitutes an intuitively natural extension of the semantics of normal logic programs;
- there exists a reasonably simple procedural mechanism allowing, at least in principle, to compute the semantics;
- the proposed class of programs and their semantics is a special case of a more general non-monotonic formalism which clearly links it to other well-established non-monotonic formalisms.

In this paper we propose a specific class of extended logic programs which will be (modestly) called *super logic programs* or just *super-programs*. We will argue that the class of super-programs satisfies all of the above conditions, and, in addition, is sufficiently flexible to allow various application-dependent extensions and modifications. We also provide a brief description of a Prolog implementation of a *query-answering interpreter* for the class of super-programs which is available via FTP and WWW.

1 Introduction

Recently, considerable interest and research effort[1] has been given to the problem of finding a suitable extension of the *logic programming paradigm* beyond the class of normal logic programs. In particular, considerable work has been devoted to the problem of defining natural extensions of logic programming that ensure a proper treatment of *disjunctive information*. However, the problem of finding a suitable semantics for disjunctive programs and databases proved to be far more complex than it is in the case of normal, non-disjunctive programs[2].

There are good reasons justifying this extensive research effort. In natural discourse as well as in various programming applications we often use *disjunctive statements*. One particular example of such a situation is *reasoning by cases*. Other obvious examples include:

- *Null values:* for instance, an age "around 30" can be 28, 29, 30, 31, or 32;
- *Legal rules:* the judge always has some freedom for his decision, otherwise he/she would not be needed; so laws cannot have unique models;
- *Diagnosis:* only at the end of a fault diagnosis we know exactly which part of some machine was faulty but as long as we are searching, different possibilities exist;
- *Biological inheritance:* if the parents have blood groups A and 0, the child must also have one of these two blood groups (example from [Lip79]);
- *Natural language understanding:* here there are many possibilities for ambiguity and they are

[1]It suffices just to mention several recent workshops on *Extensions of Logic Programming* specifically devoted to this subject ([DPP95, DHSH96]).

[2]The book by Minker et. al. [LMR92] provides a detailed and well-organized account of the extensive research effort in this area. See also [Dix95, Min93].

represented most naturally by multiple intended models;

- *Conflicts in multiple inheritance:* if we want to keep as much information as possible, we should assume disjunction of the inherited values [BL93].

Formalisms promoting disjunctive reasoning are more *expressive* and *natural to use* since they permit direct translation of disjunctive statements from natural language and from informal specifications. The additional expressive power of disjunctive logic programs significantly simplifies the problem of *translation* of non-monotonic formalisms into logic programs, and, consequently, facilitates using logic programming as an *inference engine* for non-monotonic reasoning. Moreover, extensive recent work devoted to theoretic and algorithmic foundations of disjunctive programming, suggests that there are good prospects for *extending the logic programming paradigm* to disjunctive programs.

What then should be viewed as an "extension of logic programming"? We believe that in order to demonstrate that a class of programs can be justifiably called an *extension of logic programs* one should be able to argue that:

- the proposed *syntax* of such programs resembles the syntax of logic programs but it applies to a significantly broader class of programs, which includes the class of disjunctive logic programs as well as the class of logic programs with "classical" (or strong) negation;
- the proposed *semantics* of such programs constitutes an intuitively natural extension of the semantics of normal logic programs, which, when restricted to normal logic programs, coincides with one of the well-established semantics of normal logic programs;
- there exists a reasonably simple procedural mechanism allowing, at least in principle, to compute the semantics[3].

It would be also desirable for the proposed class of programs and their semantics to be a special case of a more general non-monotonic formalism which would clearly link it to other well-established non-monotonic formalisms.

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

In this paper we propose a specific class of such extended logic programs which will be (modestly) called *super logic programs* or just *super-programs.* We will argue that the class of super-programs satisfies all of the above conditions, and, in addition, is sufficiently flexible to allow various application-dependent extensions and modifications. We also provide a brief description of a Prolog implementation of a *query-answering interpreter* for the class of super-programs which is available via FTP and WWW[4].

The paper is organized as follows. In the next Section 2 we recall the definition of a general and highly expressive non-monotonic formalism, introduced earlier by Przymusinski in [Prz95a], by introducing the class of *non-monotonic knowledge databases*. Subsequently, we establish a new and powerful fixed-point characterization of the semantics of such knowledge databases, which provides the foundation for the remaining results obtained in the paper. In Section 3 we introduce the class of super logic programs as a special subclass of the class of all non-monotonic knowledge databases and we establish basic properties of super programs. In the following Sections 4 and 5 we describe two important characterizations, one of which is syntactic and the other model-theoretic, of the semantics of super-programs which, due to the restricted nature of super programs, are significantly simpler than those applicable to arbitrary non-monotonic knowledge databases. Finally, in Section 6 we briefly describe our implementation of a *query-answering interpreter* for super-programs which is based on the previously established model-theoretic characterization of their semantics. We conclude with some final remarks in Section 7.

2 Non-Monotonic Knowledge Databases

We first define the notion of a non-monotonic knowledge database. Super logic programs, defined in the next section, are special knowledge databases.

Consider a fixed propositional language $\mathcal{L}$ with standard connectives ($\vee$, $\wedge$, $\rightarrow$, $\neg$) and the propositional letters *true* and *false*. We denote the set of its propositions by $At_{\mathcal{L}}$. Extend the language $\mathcal{L}$ to a propositional modal language $\mathcal{L}_{Not}$ by augmenting it with a modal operator Not, called the *default negation* operator. The atomic formulae of the form $Not\, F$, where F is an arbitrary formula of $\mathcal{L}_{Not}$, are called *default negation atoms* or just *default atoms.* The formulae of the

[4]See `ftp://ftp.informatik.uni-hannover.de/-software/static/static.html`.

original language $\mathcal{L}$ are called *objective*. Any propositional theory in the modal language $\mathcal{L}_{Not}$ will be called a *non-monotonic knowledge database*. Observe that arbitrarily deep level of *nested default negations* is allowed in the language. For the sake of simplicity, in this paper we consider only *finite* theories.

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

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

$$Not\,(F \vee G) \leftrightarrow Not\,F \wedge Not\,G \tag{1}$$

(D) Consistency Axiom:

$$Not\,(false) \text{ and } \neg Not\,(true) \tag{2}$$

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

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

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

Definition 2.1 (Derivable Formulae)
For any knowledge database T we denote by $Cn_{Not}(T)$ the smallest set of formulae of the language $\mathcal{L}_{Not}$ which contains T, all the (substitution instances of) the axioms (K') and (D) and is closed under both propositional consequence and the invariance rule (N').

We say that a formula F is derivable *from the knowledge database T if F belongs to $Cn_{Not}(T)$. A knowledge database T is* consistent *if $Cn_{Not}(T)$ is consistent.*

The notion of a non-monotonic knowledge database defined above is equivalent to the notion of a *belief theory* in the *Autoepistemic Logic of Beliefs*, *AELB*, originally defined by Przymusinski in [Prz95a]. Although a slightly different notation is used, one can easily see that after replacing everywhere $Not\,F$ by $\mathcal{B}\neg F$ the axioms (K'), (D) and (N') introduced above become equivalent to the axioms (K), (D) and (N) originally used in [Prz95a]. However, the definition given in here is better suited for the introduction of the class of super logic programs.

As the name indicates, non-monotonic knowledge databases must be equipped with a *non-monotonic semantics*. Intuitively this means that we need to provide a meaning to the default negation atoms $Not\,F$. We want the intended meaning of default atoms $Not\,F$ to be based on the principle of *predicate minimization*:

$$\begin{array}{c} Not\,F \text{ holds if } \neg F \text{ is minimally entailed} \\ \text{iff} \\ F \text{ is false in all minimal models.} \end{array}$$

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

Definition 2.2 (Minimal Models) *[Prz95a]*
By a minimal model *of a knowledge database T we mean a model M of T with the property that there is* no *smaller model N of T which coincides with M on default atoms $Not\,F$. If a formula F is true in all minimal models of T then we write: $T \models_{\min} F$ and say that F is* minimally entailed[5] *by T.*

Like it is the case with Clark's completion of a logic program P, the non-monotonic semantics of a knowledge database T is given by defining its *static completion $\overline{T}$*.

Definition 2.3 (Static Completion)
The static completion $\overline{T}$ of a knowledge database T is defined as:

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

where the F's range over all formulae of $\mathcal{L}_{Not}$. The static semantics *of a knowledge database T is defined as the set of all formulae in its completion $\overline{T}$ and thus it coincides with $\overline{T}$.*

Static completion (or static semantics) of a knowledge base T is obtained therefore by: (1) adding to T all default atoms $Not\,F$ satisfying the condition that the formula $\neg F$ is minimally entailed by T, and, (2) closing the resulting theory under the consequence operator

[5] For readers familiar with *circumscription*, this means that we are considering circumscription $CIRC(T;\mathcal{L})$ of the knowledge database T in which objective atoms are minimized while the default atoms $Not\,F$ are fixed, i.e., $T \models_{\min} F \equiv CIRC(T;\mathcal{L}) \models F$. In other words, minimal models are obtained by first assigning *arbitrary* truth values to the default atoms and then *minimizing* objective atoms (see also [YY93]).

Cn_{Not}. Consequently, the definition of static completions *enforces* the intended meaning of default atoms described above.

The above definition of static completion is very simple and straightforward and it does *not* involve any fixed-point computation. The main result of this paper, which provides the foundation for all the remaining results, proves that static completion can be equivalently defined as the *least fixed point* of a natural operator defined below. In particular, it shows that the notion of a static completion given above coincides with the notion of the *least static expansion* in $AELB$ mentioned above. The existence of an equivalent non-fixed point definition of a static completion significantly simplifies this notion and the underlying theory. First, we consider a very simple example.

Example 2.4 Consider a knowledge database T given by $Not\, a \rightarrow b$. It is easy to see that $T \models_{min} \neg a$ because a is false in all minimal models of T. As a result, the static completion $\overline{T}$ of T contains $Not\, a$. Consequently, as expected, the static semantics of T derives b.

Let T be a fixed knowledge database. We now define (see [Prz95a]) the monotonic[6] *default closure* operator Ψ_T which transforms any database S into its default closure:

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

obtained by (1) adding to T all default atoms $Not\, F$ satisfying the condition that the formula $\neg F$ is minimally entailed by S, and, (2) closing the resulting theory under the consequence operator Cn_{Not}. The F's range over all formulae of $\mathcal{L}_{Not}$.

Any fixed point of the operator Ψ_T is called a static expansion of T:

Definition 2.5 (Static Expansion) *[Prz95a]*
A knowledge database $T^\diamond$ is called a static expansion *of a knowledge database T if it is a fixed-point of the operator Ψ_T:*

$$T^\diamond = \Psi_T(T^\diamond) =_{def} Cn_{Not}(T \cup \{Not\, F : T^\diamond \models_{\min} \neg F\}),$$

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

Clearly, the static completion $\overline{T}$ of T can be equivalently defined as $\overline{T} = \Psi_T(T)$. In other words, $\overline{T}$ is obtained as a result of a *single iteration* of the operator Ψ_T.

[6] Strictly speaking the operator is only monotone on the lattice of all theories of the form $T \cup M_{Not}$ where T is fixed and M_{Not} ranges over all sets of default negation atoms.

It follows from the results obtained in [Prz95a] that every database T has the *least* (in the sense of set-theoretic inclusion) static expansion $T^\diamond$, i.e., the least fixed point of the operator Ψ_T. However, the following powerful and yet surprising result shows that for every database T its least static expansion is obtained by a *single iteration* of the operator Ψ_T and thus it always coincides with the static completion $\overline{T}$ of T.

Theorem 2.6 (Main)
Let T be any knowledge database. The least static expansion of T is always obtained by a single iteration of the operator Ψ_T. In other words, the least fixed point of the operator Ψ_T coincides with the static completion $\overline{T}$ of T:

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

Since static completion is the least static expansion, a knowledge database T either has a *consistent* completion $\overline{T}$ or it does *not* have any consistent static expansions at all.

Corollary 2.7 *The notion of a static completion given above coincides with the notion of the* least static expansion *in the Autoepistemic Logic of Beliefs,* AELB, *defined in [Prz95a].*

Before presenting the next simple example, we will need the following technical lemma. Its first statement says that if a formula F is assumed to be false by default then its negation $\neg F$ is not.

Lemma 2.8 *If $T = Cn_{Not}(T)$ then for any two formulae F and G of $\mathcal{L}_{Not}$:*

$$T \models Not\, F \rightarrow \neg Not\, \neg F$$
$$T \models (Not\, F \leftrightarrow Not\, G) \leftrightarrow Not\, \neg(\neg F \leftrightarrow \neg G).$$

Example 2.9 Consider a knowledge database T:

$$Not\, a \rightarrow b \qquad (4)$$
$$Not\, b \rightarrow c. \qquad (5)$$

Obviously $T \models_{min} \neg a$ and $T \models_{min} \neg Not\, b \leftrightarrow \neg c$. Thus the static completion $\overline{T}$ contains $Not\, a$, b and $Not\, \neg(\neg Not\, b \leftrightarrow \neg c)$. By Lemma 2.8, $\overline{T}$ also contains $Not\, Not\, b \leftrightarrow Not\, c$. Moreover, $\overline{T}$ contains $Not\, \neg b$ and thus, by Lemma 2.8, $\neg Not\, b$. Consequently, $\overline{T}$ contains $Not\, Not\, b$ and thus also $Not\, c$. As expected, the static semantics of T derives $Not\, a$, b, $Not\, c$.

3 Super Logic Programs

We now introduce the class of *super logic programs*, briefly called *super programs*, as a subclass of the class of all non-monotonic knowledge databases. This powerful class of databases includes all *propositional theories*, all *disjunctive logic programs* and all *extended logic programs* with "classical" or strong negation [GL90, APP96].

Definition 3.1 (Super Logic Programs)
A Super Logic Program is a non-monotonic knowledge database consisting of super-clauses of the form:

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

where F and G are arbitrary objective *formulae and H is an arbitrary* positive[7] *objective formula.*

Observe that since any formula $F \leftarrow G$ can be written as a finite conjunction of disjunctions of literals, we can always replace a super-clause by a finite set of clauses whose heads are disjunctions $A_1 \vee \ldots \vee A_k$ of atoms and whose positive premises are conjunctions $B_1 \wedge \ldots \wedge B_m$ of objective atoms. Finally, since the operator *Not* obeys the *distributive law for disjunction* (K'), the default atom $Not\, H$ can be replaced by the conjunction $Not\, C_1 \wedge \ldots \wedge Not\, C_n$ of default atoms $Not\, C_i$, where each of the C_i's is a conjunction of objective atoms. This allows us to establish the following useful fact.

Proposition 3.2 *A super logic program P can be equivalently defined as a set of clauses of the form:*

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

where A_i's and B_i's are objective atoms and C_i's are conjunctions of objective atoms.

If $k \neq 0$, for all clauses of P, then the program is called affirmative.

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

- It allows constraints, i.e., headless rules. In particular, it allows the addition of *strong* or *"classical"* negation to such programs by introducing *new objective atoms*, say $\neg A$, and assuming the *strong negation axioms* [APP96] $\leftarrow A \wedge \neg A$, which themselves are program rules (rather than meta-level constraints).

- It allows premises of the form $Not\, C$, where C is not just an atom but a conjunction of atoms. This proves very important when reasoning with default assumptions which themselves are *rules*, such as $Not(man \wedge \neg human)$, which can be interpreted as stating that we can assume by default that every man is a human (note that $\neg human$ represents strong negation of $human$).

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

In the following two sections we provide two such powerful characterizations of static completions of super programs and one of them is later used to implement a query-answering interpreter for super programs. First, however, we discuss general properties of super programs and give a simple example.

The class of super programs has a number of useful properties. Our first result shows that for any formula F the static semantics $\overline{P}$ of a super program P derives $Not\, F$ if and only if $\neg F$ is minimally entailed by $\overline{P}$. This further clarifies the meaning of negation by default.

Theorem 3.3 *Let P be any super logic program and let $\overline{P}$ be its static completion. For any formula F, $\overline{P} \models Not\, F \;\equiv\; \overline{P} \models_{min} \neg F$.*

The next result follows from Corollary 2.7 and the results established in [Prz95a].

Theorem 3.4 (Basic Properties)

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

- *For any normal logic program static semantics coincides with the well-founded semantics. Moreover, there is a one-to-one correspondence between consistent static expansions of a normal program and its partial stable models. Under this correspondence, (total) stable models correspond to those consistent static expansions which satisfy the axiom $Not\, A \vee Not\, \neg A$, for every objective atom A.*

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

- *For any positive disjunctive logic program static semantics coincides with the minimal model semantics.*

Example 3.5 Consider a simple super-program P:

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

Given the usual exclusive interpretation of disjunctions we would most likely conclude that we will visit either Australia or Europe but not both and thus we are likely to expect the answer to the query $VisitEurope \vee VisitAustralia$ to be positive resulting in a positive answer to the query $Happy$. Similarly, we expect the answer to the query $Not\,(VisitEurope \wedge VisitAustralia)$ to be positive resulting in a positive answer to the query $Prudent$.

On the other hand, we expect the answer to the query $VisitEurope \wedge VisitAustralia$ to be negative thus resulting in a negative answer to $Bankrupt$. Similarly, we expect the answer to the query $Not\,(VisitEurope \vee VisitAustralia)$, or, equivalently $Not\,VisitEurope \wedge Not\,VisitAustralia$ to be negative thus resulting in a negative answer to the query $Disappoint.$.

The query $Not\,(VisitEurope \wedge VisitAustralia)$ intuitively means *"can it be assumed by default that we don't visit both Europe and Australia?"* and is different from the query $Not\,VisitEurope \wedge Not\,VisitAustralia$ which says *"can it be assumed by default that we don't visit Europe and can it be assumed by default that we don't visit Australia?"*.

It turns out that static semantics produces precisely the intended meaning discussed above. Indeed, clearly $Happy$ must belong to the static completion of P. It is easy to check that $\neg VisitEurope \vee \neg VisitAustralia$ holds in all minimal models of the program and therefore $Not\,(VisitEurope \wedge VisitAustralia)$ must be true in the completion. This proves that $Prudent$ must hold in the completion. Moreover, since $Bankrupt$ is false in all minimal models $Not\,Bankrupt$ must also belong to the completion. Finally, since $VisitEurope \vee VisitAustralia$ is true we infer from (D), (N') that $\neg Not\,(VisitEurope \vee VisitAustralia)$ holds and thus $\neg Disappoint.$ is minimally entailed which implies that $Not\,Disappoint.$ belongs to the completion.

4 Simple Fixed Point Characterization of Static Semantics

Due to Proposition 3.2, we can consider the language of super logic programs to be restricted to the subset:

$$\mathcal{L}^*_{Not} = \mathcal{L} \cup \{Not\,E : E \text{ is a conj. of objective atoms}\}$$

of the full language $\mathcal{L}_{Not}$. In particular, the language $\mathcal{L}^*_{Not}$ does *not* allow any *nesting* of default negations. In fact, default negation can be only applied to conjunctions of objective atoms.

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

Theorem 4.1 (Fixed-Point Characterization) *The static completion $\overline{P}$ of a super program P can be constructed as follows. Let $P^0 = P$ and suppose that P^n has already been defined for some natural number n. Define P^{n+1} as follows:*

$$\begin{aligned} Cn(\ P\cup\ \ &\{Not\,E_1 \wedge \ldots \wedge Not\,E_n \rightarrow Not\,E_0 : \\ &\quad P^n\ \models_{\min} \neg E_1 \wedge \ldots \wedge \neg E_n \rightarrow \neg E_0\}\ ), \end{aligned}$$

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

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

The above Theorem represents a considerable simplification over the original definition of static completions given in Definition 2.3. First of all, instead of using the modal consequence operator Cn_{Not} the definition uses the *standard* propositional consequence operator

Cn. Moreover, instead of ranging over the set of all (arbitrarily deeply nested) formulae of the language $\mathcal{L}_{Not}$ it involves only conjunctions of objective atoms. These two simplifications greatly enhance the implementability of static semantics. On the other hand, however, due to the restriction to the language $\mathcal{L}^*_{Not}$, we can no longer expect the fixed point to be reached in just one step.

5 Model-Theoretic Characterization

In this section, we complement Theorem 4.1 by providing a *model-theoretic* characterization of static completions of super programs. More precisely, we characterize models of static completions restricted to the narrower language $\mathcal{L}^*_{Not}$. The resulting characterization was directly used in a prototype implementation of a query answering interpreter for static semantics described briefly in the next section.

Definition 5.1 (Reduced Interpretations)

1. *Let Obj be the set of all propositional valuations of the objective atoms $At_{\mathcal{L}}$.*

2. *Let Def be the set of all propositional valuations of the default atoms*

$$\{Not\,(p_1 \wedge \cdots \wedge p_n) \mid p_i \in At_{\mathcal{L}}\}.$$

3. *We write interpretation I of the language $\mathcal{L}^*_{Not}$, i.e. a valuation of*

$$At_{\mathcal{L}} \cup \{Not\,(p_1 \wedge \cdots \wedge p_n) \mid p_i \in At_{\mathcal{L}}\},$$

as $I = I_{obj} \cup I_{def}$ with $I_{obj} \in Obj$ and $I_{def} \in Def$.

4. *In order to emphasize the difference, we sometimes call an interpretation $\mathcal{I}$ of the complete language $\mathcal{L}_{Not}$, i.e. a valuation for $At_{\mathcal{L}} \cup \{Not\,(F) \mid F \in \mathcal{L}_{Not}\}$, a full interpretation.*

5. *Given a full interpretation $\mathcal{I}$, we call its restriction $I = I_{obj} \cup I_{def}$ to the language $\mathcal{L}^*_{Not}$ the reduct of $\mathcal{I}$.*

Since an interpretation $I = I_{obj} \cup I_{def}$ assigns truth values to all atoms occurring in a super program, the "is model of" relation between such interpretations and super programs is defined. We call $I = I_{obj} \cup I_{def}$ a minimal model of a super program P if there is no objective interpretation I'_{obj} satisfying

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

such that $I' = I'_{obj} \cup I_{def}$ is also a model of P. This is completely compatible with the corresponding notion for full interpretations.

Now our goal in this section is to characterize the reducts of the full models of the static completion $\overline{P}$ of a super program P. In fact, the objective parts of these reducts are easy, once we have the possible interpretations of the default atoms. The idea how to compute them is closely related to Kripke structures (which is not surprising for a modal logic). The worlds in such a Kripke structure are marked with objective interpretations, and the formula $Not\,(p_1 \wedge \cdots \wedge p_n)$ is true in a world w iff $\neg(p_1 \wedge \cdots \wedge p_n)$ is true in all worlds w' which can be "seen" from w. Due to the consistency axiom, every world w must see at least one world w'. Now all the static semantics does is to ensure that every world w sees only worlds marked with minimal objective models. So given a set $\mathcal{M}_{obj}$ of minimal objective models, an interpretation I_{def} of the default atoms is possible iff there is some subset $\mathcal{M}'_{obj} \subseteq \mathcal{M}_{obj}$ (namely the objective interpretations in the worlds w') such that $I_{def} \models Not\,(p_1 \wedge \cdots \wedge p_n)$ iff $I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n)$ for all $I_{obj} \in \mathcal{M}'_{obj}$. But the minimal objective models $\mathcal{M}_{obj}$ also depend conversely on the possible interpretations of the default atoms (since a minimal model contains an objective part and a default part). So when we restrict the possible default interpretations, we also get less minimal objective models. This obviously results in a fixpoint computation, formalized by the following operators:

Definition 5.2 (Possible Int. of Default Atoms)

Let P be a super program.

1. *The operator $\Phi_P\colon 2^{Def} \to 2^{Obj}$ yields the minimal objective models given a set of possible interpretations of the default atoms. For every $\mathcal{M}_{def} \subseteq Def$ let $\Phi_P(\mathcal{M}_{def}) :=$*

$$\{I_{obj} \in Obj \mid \text{there is a } I_{def} \in \mathcal{M}_{def} \text{ such that } I = I_{obj} \cup I_{def} \text{ is min. model of } P\}.$$

2. *The operator $\Psi_P\colon 2^{Obj} \to 2^{Def}$ yields the possible interpretations of the default atoms, given a set of minimal objective models. For every $\mathcal{M}_{obj} \subseteq Obj$ let $\Psi_P(\mathcal{M}_{obj}) :=$*

$$\{I_{def} \in Def \mid \text{there is a non-empty } \mathcal{M}'_{obj} \subseteq \mathcal{M}_{obj} \text{ such that for every } p_1, \ldots, p_n \in At_{\mathcal{L}}\colon\ I_{def} \models Not\,(p_1 \wedge \cdots \wedge p_n) \iff \text{for every } I_{obj} \in \mathcal{M}'_{obj}\colon\ I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n)\}.$$

3. *The operator* $\Theta_P: 2^{Def} \to 2^{Def}$ *is the composition* $\Theta_P := \Phi_P \circ \Psi_P$ *of* Φ_P *and* Ψ_P.

Theorem 5.3 (Model-Theor. Characterization)

1. *The operator* Θ_P *is monotone and thus its iteration beginning from the set* Def *of all interpretations of the default atoms has a fixed point* $Def^\diamond$.

2. $Def^\diamond$ *consists exactly of all* $I_{def} \in Def$ *that are default parts of a full model* $\mathcal{I}$ *of* $\overline{P}$.

3. *A reduced interpretation* $I_{obj} \cup I_{def}$ *is a reduct of a full model* $\mathcal{I}$ *of* $\overline{P}$ *iff* $I_{def} \in Def^\diamond$ *and* $I_{obj} \cup I_{def} \models P$.

Example 5.4 Let us consider the following logic program P:

$$\begin{aligned} p \vee q &\leftarrow Not(\neg r). \\ q &\leftarrow Not(\neg q). \\ r &\leftarrow q. \end{aligned}$$

Since only belief atoms with a single proposition occur in P, it suffices to consider only such belief atoms. We start with the set Bel containing all eight possible valuations of the three belief atoms $Not(\neg p)$, $Not(\neg q)$, and $Not(\neg r)$. These can be extended to the minimal reduced models listed in the following table. Note that models number 6 to 10 are equal to models number 1 to 5, except that $Not(\neg p)$ is true in them.

No.	$Not(\neg p)$	$Not(\neg q)$	$Not(\neg r)$	p	q	r
1.	0	0	0	0	0	0
2.	0	0	1	1	0	0
3.	0	0	1	0	1	1
4.	0	1	0	0	1	1
5.	0	1	1	0	1	1
6.	1	0	0	0	0	0
7.	1	0	1	1	0	0
8.	1	0	1	0	1	1
9.	1	1	0	0	1	1
10.	1	1	1	0	1	1

So we see that there are only three possible valuations for the objective parts, i.e. $\Phi_P(Bel)$ is:

p	q	r
0	0	0
1	0	0
0	1	1

Now these minimal objective interpretations are the basis for the next round of belief interpretations. Of course, from every minimal objective interpretation we immediately get a possible belief belief interpretation if we translate the truth of p to the falsity of $Not(\neg p)$.

But we can also combine minimal objective parts conjunctively and let $Not(\neg p)$ be true only if $\neg p$ is true in all elements of some set of minimal models. Thus, we get the following four valuations for the belief atoms in $\Psi_P(\Phi_P(Bel))$:

$Not(\neg p)$	$Not(\neg q)$	$Not(\neg r)$
1	1	1
0	1	1
1	0	0
0	0	0

This means that only the models number $1, 5, 6, 10$ remain possible given the current knowledge about the beliefs.

Now p is false in all of these models, so $Not(\neg p)$ must be assumed, and only the following two valuations of the belief atoms are possible in the fixed point $Bel^\diamond$:

$Not(\neg p)$	$Not(\neg q)$	$Not(\neg r)$
1	1	1
1	0	0

The reduced models of the least static expansion $P^\diamond$ are all reduced interpretations which satisfy the rules of P and have one of these two belief parts.

Taking the reduct of a full interpretation was easy. However, with the following Kripke structure $\mathcal{K} = \langle W, R, V\rangle$ we can extend the reduced interpretations back to full interpretations, which are models of the least static expansion:

- The set of worlds W are the reduced interpretations $I = I_{obj} \cup I_{bel}$ which are models of P and satisfying $I_{bel} \in Bel^\diamond$.

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

  $$I \models Not(\neg p_1 \vee \cdots \vee \neg p_n) \implies I' \models \neg p_1 \vee \cdots \vee \neg p_n.$$

- The valuation $V(I)$ of a reduced interpretation $I = I_{obj} \cup I_{bel}$ is the objective part I_{obj}.

Now in order to get from a reduced interpretation $I \in W$ to a full interpretation, we simply assign the formulas $Not(F)$ the truth values they have in the world I.

However, let us finally note that not all full models of the least static expansion are reconstructed in this way, only one representative from every equivalence class with the same reduct. Already the program $P := \{p \leftarrow Not(\neg p)\}$ has infinitely many different full models.

6 Implementation of the Model-Theoretic Characterization

We implemented a *query-answering interpreter for the static semantics* in the class of super programs based on the above model-theoretic characterization[8]. Let us explain in a little more detail how our prototype implementation works. First, because of the large number of possible default atoms, it seems impossible to construct explicitly the set *Def* of all valuations — it would have 2^{2^n} elements, where n is the number of objective atoms. However, it suffices to consider only the default atoms which occur explicitly in the program, other default atoms have no influence on the minimal models and we can always extend a valuation of these "critical" default atoms consistently to a valuation of all default atoms.

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

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

because we know that the body can never be true (this is a special case of the "negative reduction" transformation used in [BD95]). In this example, deleting the rule also eliminates two default atoms.

So the first step of our algorithm is to compute the residual program and to apply the reduction operators defined in [BD95]. This transformation preserves the minimal models, and often significantly reduces the number of default atoms we have to consider (for instance, no default atoms remain if the input program is stratified and non-disjunctive).

Then we apply the definition of the operators Φ_P and Ψ_P from Section 5 quite literally. However, two points are worth mentioning. In order to compute the minimal models for the Φ_P-operator, we use an extension of Clark's completion. Given an interpretation of the default atoms occurring in the residual program, we evaluate the rule bodies, so we get a set of positive disjunctions $p_1 \vee \cdots \vee p_n$. We compute the completion by treating $p_1 \vee \cdots \vee p_n$ like $p_i \leftarrow \neg p_1 \wedge \cdots \wedge \neg p_{i-1} \wedge \neg p_{i+1} \wedge \cdots \wedge \neg p_n$. Then we use any model generation algorithm.

Next, it would be very inefficient to consider all subsets $\mathcal{M}'_{obj} \subseteq \mathcal{M}_{obj}$, as required in the definition of the Ψ_P-operator. However, in order to check whether a given interpretation I_{def} of the default atoms is selected by Θ_P, only the maximal $\mathcal{M}'_{obj}$ is interesting. We construct it as the set consisting of all $I_{obj} \in \mathcal{M}_{obj}$ satisfying the condition

for every default atom: if $I_{def} \models \mathit{Not}\,(p_1 \wedge \cdots \wedge p_n)$, then $I_{obj} \models \neg(p_1 \wedge \cdots \wedge p_n)$.

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

The program prints the fixed point of Θ_P (only the evaluations of the critical default atoms), and then the minimal reduced models the default part of which is contained in Θ_P°. It then waits for queries. We allow *arbitrary queries* to super programs which are answered entirely on the basis of these minimal models.

7 Conclusion

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

We proved two powerful syntactic and set-theoretic characterizations of the semantics of super-programs which lead to procedural mechanisms allowing its computation. We used one of these characterizations as a basis for the implementation of a *query-answering interpreter* for super-programs which is available via FTP and WWW. We noted that while no such computational mechanism can be efficient, due to the inherent NP-completeness of the problem of computing answers to just positive disjunctive programs, they can become efficient when restricted to specific subclasses of programs and queries. Moreover, further research is likely to produce more efficient *approximation methods*.

There is an interesting relation of our work to [YY93] where the authors defined the *well-founded circumscriptive* semantics for disjunctive programs (without nested default negations). They introduced the con-

[8] See `ftp://ftp.informatik.uni-hannover.de/-software/static/static.html` from which the interpreter is available via FTP and WWW.

cept of minimal model entailment with *fixed* belief atoms and defined their semantics as a *single step application* of circumscription. Our work can therefore be seen as an extension to theirs.

The class of super programs with static semantics constitutes a special case of a much more expressive non-monotonic formalism called the *Autoepistemic Logic of Knowledge and Beliefs* , *AELB*, and introduced earlier in [Prz95a]. *AELB* isomorphically includes the well-known non-monotonic formalisms of Moore's *Autoepistemic Logic* and McCarthy's *Circumscription*. Via this embedding, the semantics of super programs is clearly linked to other well-established non-monotonic formalisms.

The proposed semantic framework for super programs is sufficiently flexible to allow various application-dependent extensions and modifications. We have already seen in Theorem 3.4 that by assuming an additional axiom we can produce an extension of the stable semantics instead of the well-founded semantics. By adding the distributive axiom for conjunction we can obtain a semantics that extends the *disjunctive stationary semantics* of logic programs introduced (see [Prz95b]). Many other modifications and extensions are possible including variations of the notion of a minimal model resulting in an *inclusive* instead of *exclusive* interpretation of disjunctions.

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

References

[APP96] J. Alferes, L. Pereira, and T. C. Przymusinski. 'classical' negation in non-monotonic reasoning. In *Proceedings of the International Conference on Artificial Intelligence and Mathematics, AI&Math'96, January'96*, pages 1–4, 1996.

[BD95a] Stefan Brass and Jürgen Dix. A General Approach to Bottom-Up Computation of Disjunctive Semantics. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Nonmonotonic Extensions of Logic Programming*, LNAI 927, pages 127–155. Springer, Berlin, 1995.

[BD95] Stefan Brass and Jürgen Dix. Disjunctive Semantics based upon Partial and Bottom-Up Evaluation. In Leon Sterling, editor, *Proceedings of the 12th Int. Conf. on Logic Programming, Tokyo*, pages 199–213. MIT Press, June 1995.

[BD96] Stefan Brass and Jürgen Dix. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *Journal of Logic Programming*, forthcoming, 1996.

[BL93] Stefan Brass and Udo W. Lipeck. Bottom-up query evaluation with partially ordered defaults. In Stefano Ceri, Katsumi Tanaka, and Shalom Tsur, editors, *Deductive and Object-Oriented Databases, Third Int. Conf., (DOOD'93)*, number 760 in LNCS, pages 253–266, Berlin, 1993. Springer.

[Bry89] François Bry. Logic programming as constructivism: A formalization and its application to databases. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'89)*, pages 34–50, 1989.

[Bry90] François Bry. Negation in logic programming: A formalization in constructive logic. In Dimitris Karagiannis, editor, *Information Systems and Artificial Intelligence: Integration Aspects*, pages 30–46, Berlin, 1990. Springer.

[DHSH96] Roy Dyckhoff, Heinrich Herre, and Peter Schroeder-Heister, editors. *Extensions of Logic Programming*, LNAI 1050, Berlin, 1996. Springer.

[Dix95] Jürgen Dix. Semantics of Logic Programs: Their Intuitions and Formal Properties. An Overview. In Andre Fuhrmann and Hans Rott, editors, *Logic, Action and Information – Essays on Logic in Philosophy and Artificial Intelligence*, pages 241–327. DeGruyter, 1995.

[DK89a] P. M. Dung and K. Kanchansut. A fixpoint approach to declarative semantics of logic programs. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of North American Conference Cleveland,Ohio, USA*. MIT Press, October 1989.

[DK89b] P. M. Dung and K. Kanchansut. A natural semantics for logic programs with negation. In E.L. Lusk and R.A. Overbeek, editors, *Proceedings of the 9th Conference on Foundations of Software Technology and Theoretical Computer Science*, Berlin, 1989. Springer.

[DPP95] J. Dix, L. Pereira, and T. Przymusinski, editors. *Non-Monotonic Extensions of Logic Programming*, LNAI 927, Berlin, 1995. Springer.

[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, Cambridge, Mass., 1990. Association for Logic Programming, MIT Press.

[Lip79] W. Lipski, Jr. On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, 4:262–296, 1979.

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

[Min93] Jack Minker. An Overview of Nonmonotonic Reasoning and Logic Programming. *Journal of Logic Programming, Special Issue*, 17, 1993.

[Prz95a] T. C. Przymusinski. Autoepistemic logic of knowledge and beliefs. (In preparation), University of California at Riverside, 1995. (Extended abstract appeared in 'A knowledge representation framework based on autoepistemic logic of minimal beliefs' In *Proceedings of the Twelfth National Conference on Artificial Intelligence, AAAI-94, Seattle, Washington, August 1994*, pages 952-959, Los Altos, CA, 1994. American Association for Artificial Intelligence, Morgan Kaufmann.).

[Prz95b] Teodor Przymusinski. Semantics of normal and disjunctive logic programs: A unifying framework. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Proceedings of the Workshop on Non-Monotonic Extensions of Logic Programming at the Eleventh International Logic Programming Conference, ICLP'95, Santa Margherita Ligure, Italy, June 1994*, pages 43–67. Springer Verlag, 1995.

[SS95] Chiaki Sakama and Hirohisa Seki. Partial Deduction of Disjunctive Logic Programs: A Declarative Approach. In *Logic Program Synthesis and Transformation – Meta Programming in Logic*, LNCS 883, pages 170–182, Berlin, 1995. Springer.

[YY93] Jia-Huai You and Li-Yan Yuan. Autoepistemic Circumscription and Logic Programming. *Journal of Automated Reasoning*, 10, pages 143–160, 1993.

A Proof of the Main Theorem via Clark's Completion

In this section we present a syntactic characterization of static semantic which shows that the least static expansion of a super-program P can be obtained in just *one step* by closing the theory $P_{comp} = P \cup \{Not\, F : F \in compl(P_{res})\}$ under Cn_{Not}. Here P_{res} denotes the *residuum* of the program P [BD95] and $compl(P_{res})$ denotes (the disjunctive version of) *Clark's completion* of P_{res}. This result not only eliminates the need for multiple iterations in the computation of static expansions of super-programs but it also underscores a very interesting and somewhat intriguing relationship between static semantics and Clark's completion. Since the theory P_{comp} is easily computable, this result reduces reasoning under the static semantics to theorem-proving in the modal logic AELB (either by hand or with an automated theorem prover). In particular, the fixpoint construction of the least static expansion can be altogether avoided.

The idea is to add to P the set of formulae which ensure that models "seen" through the modality *Not* are in fact minimal. An appropriate generalization of Clark's completion accomplishes this, but only for programs *without positive body literals*. Such "residual programs" were introduced and investigated in [Bry89, Bry90, DK89a, DK89b, BD95, BD95a]. Fortunately, it turns out that every program P can be transformed into a residual program P_{res} without changing the set of minimal models. In fact, the two elementary program transformations, "unfolding" (or partial evaluation, GPPE [BD95, SS95]) and "elimination of tautologies", are sufficient to obtain a residual program.

The efficient computation of the residual program and the relation to elementary program transformations is explained in [BD95a]. Here we only give an example. Consider the following program P and its residual program P_{res}:

$$
\begin{array}{rcl}
 & P: & \\
p \vee q & \leftarrow & Not\,(\neg r). \\
q & \leftarrow & Not\,(\neg q). \\
r & \leftarrow & q.
\end{array}
\qquad
\begin{array}{rcl}
 & P_{res}: & \\
p \vee q & \leftarrow & Not\,(\neg r). \\
q & \leftarrow & Not\,(\neg q). \\
p \vee r & \leftarrow & Not\,(\neg r). \\
r & \leftarrow & Not\,(\neg q).
\end{array}
$$

The first two rules already have the required form. In the last rule, we unfold the body literal q and obtain the residual program P_{res}. This is in fact nothing else but a form of hyperresolution.

We now define an appropriate version of Clark's completion for residual programs. As usual, the goal is to ensure that an objective atom A is true only if it really has to be true, i.e., if there is a rule with A in the head, in which the body is true and all the other head literals are false.

Definition A.1 (Clark's Completion of P_{res})
Let P_{res} be a super program without positive body literals. Let $A \in At_{\mathcal{L}}$ be an objective atom and

$$
\begin{array}{ccc}
A \vee B_{1,1} \vee \cdots \vee B_{1,m_1} & \leftarrow & Not\,(\neg C_{1,1}) \wedge \cdots \wedge Not\,(\neg C_{1,n_1}) \\
\vdots & \vdots & \vdots \\
A \vee B_{k,1} \vee \cdots \vee B_{1,m_k} & \leftarrow & Not\,(\neg C_{k,1}) \wedge \cdots \wedge Not\,(\neg C_{1,n_k})
\end{array}
$$

be all rules in P_{res} with A in the head. Then the completion axiom for A is the formula

$$
A \leftrightarrow \bigvee_{i=1}^{k} \big(\neg B_{i,1} \wedge \cdots \wedge \neg B_{i,m_i} \wedge Not\,(\neg C_{i,1}) \wedge \cdots \wedge Not\,(\neg C_{i,n_i})\big)
$$

As usual, Clark's completion $comp(P_{res})$ of the whole residual program P_{res} is the set of all completion axioms, for every $A \in At_{\mathcal{L}}$. We also use the shorthand

$$
Not\,\big(comp(P_{res})\big) := \big\{Not\,(F) \;\big|\; F \in comp(P_{res})\big\}.
$$

Lemma A.2
An interpretation $\mathcal{I}$ is a minimal model of a super program P iff it is a model of $comp(P_{res})$.

It is interesting to note that when we replace $Not\,(\neg C_{i,j})$ by $\neg C_{i,j}$, we obtain the stable models of P. We used this fact in [BD95a] to compute the stable semantics of disjunctive programs.

Theorem A.3 (Characterization by $comp(P_{res})$)

For every super program P, the least static expansion $P^{\diamond}$ of P is $P^{\diamond} = Cn_{AELB}(P \cup Not\,\big(comp(P_{res})\big)$.

A surprising corollary of the above Theorem is that for super programs a single application of the belief closure operator Ψ_T is sufficient. Indeed, due to Lemma A.2, the completion axioms hold in all minimal models, so they are already believed in the first iteration of Ψ_T.