Optimal Models of Disjunctive Logic Programs: Semantics, Complexity, and Computation*

Nicola Leone † **Francesco Scarcello**‡ **V.S. Subrahmanian**§

Abstract

Almost all semantics for logic programs with negation identify a *set*, $SEM(P)$, of models of program P, as the intended semantics of P, and any model M in this class is considered a possible meaning of P w.r.t. the semantics the user has in mind. Thus, for example, in the case of stable models [6], choice models [24], answer sets [7], etc., different possible models correspond to different ways of "completing" the incomplete information in the logic program. However, different end-users may have different ideas on which of these different models in $SEM(P)$ is a reasonable one from their point of view. For instance, given $SEM(P)$, user U_1 may *prefer* model $M_1 \in SEM(P)$ to model $M_2 \in SEM(P)$ based on some evaluation criterion that she has. In this paper, we will develop a notion of logic program semantics based on the concept of an *Optimal Model*. This semantics doesn't add yet another semantics to the logic programming arena – rather, it takes as input, an existing semantics $SEM(P)$ *and* a user-specified objective function Obj, and yields a new semantics $\texttt{Opt}(P) \subseteq SEM(P)$ that realizes the objective function *within the framework of preferred models identified already by* $SEM(P)$ in different ways. Thus, the user who may or may not know anything about logic programming has considerable flexibility in making the system reflect her own objectives by building "on top" of existing semantics known to the system. In addition to the declarative semantics characterization, we provide a complete complexity analysis and algorithms to compute optimal models under varied conditions when $SEM(P)$ is the stable model semantics, the minimal models semantics, and the all-models semantics.

1 Introduction

There are now a vast number of semantics for logic programs and extensions of logic programs. All these semantics identify a set of models of logic program P as the intended semantics, $SEM(P)$, of P. From the point of view of a user of a logic programming application, she is stuck with the semantics assigned by the logic programming interpreter, but cannot specify that some of these models are "better" from her point of view, than others. In other words, the end-user (not the logic programmer building applications!) cannot make the program select and answer queries w.r.t. models that she deems appropriate. Surprisingly, as we shall show through a "cooking example", different users may have different preferences even when the same program and the same semantics are considered. In this paper, we shall make the following contributions:

*Due to space reasons, not all proofs are included here. Full proofs of all results may be obtained in the technical report version of this paper[16] obtainable from www.cs.umd.edu/Library/TRs/CS-TR-4298.ps.Z. Research partly funded by ARO grant DAAD190010484, ARL grant DAAL0197K0135, and ARL's CTA on advanced decision architectures.

†Department of Mathematics, University of Calabria, I-87030 Rende, Italy, leone@unical.it

‡D.E.I.S., University of Calabria, I-87030 Rende, Italy, scarcello@deis.unical.it

§Department of Computer Science, University of Maryland, College Park, MD 29742, vs@cs.umd.edu

- First, we develop the notion of an Herbrand Objective Function that allows the user to specify an expression e that can be evaluated w.r.t. each model. Without loss of generality, we will always assume the user wishes to minimize e, i.e. to find a model M in $SEM(P)$, such that e's value is *minimized*[1].
- A model M is optimal according to a specific semantics $SEM(P)$ if $M \in SEM(P)$ and there is no model $M' \in SEM(P)$ such that the value of the expression e on M' is strictly less than the value e assumes on M.
- We then consider the cases when $SEM(P)$ is the set of stable models, minimal models, and all models of a *disjunctive* logic program (DLP). We look at two classes of programs — general DLPs and positive DLPs which have no negations in rule bodies. Finally, we identify two kinds of objective functions — those that satisfy a "monotonic aggregation strategy" property defined later in the paper and those that may or may not do so. This leads to twelve possible combinations of semantics, disjunctive logic program syntax, and aggregation-strategy type.
- For each of the above six combinations, we study the complexity of four problems.
  - Checking whether a given model is optimal;
  - Brave reasoning, which checks if a given atom is true in some optimal model;
  - Cautious reasoning, which checks if a given atom is true in all optimal models;
  - Approximation, which checks if there exists a model containing a given set of ground atoms such that the value of the Herbrand Objective Function on that model is less than or equal to a given bound.

  This leads to a total of twenty-four complexity results that are neatly summarized in Tables 1 and 2. In particular, we show that, in general, the complexity depends upon the properties of the Herbrand Objective Functions in question, monotonicity properties decrease the complexity.
- We develop a comprehensive set of algorithms that may be used to compute optimal models, both under the monotonicity assumption, and without it, for stable models, minimal models, and all models of disjunctive logic programs.

The rest of this paper is organized as follows. Section 2 motivates the paper through two examples — one is a simple "cooking" example, and the other is a more serious combinatorial auction (cf. Sandholm [25]) example. Section 3 provides basic definitions in disjunctive logic programming and computational complexity — these concepts are needed in order to follow the rest of the paper. In Section 4, we formally define what it means for a model M drawn from a family of models to be optimal. Such families of models can include stable models, minimal models, and all models. In Section 4, we develop the twenty four complexity results discussed earlier. In Section 6, we develop algorithms to compute optimal models of disjunctive logic programs under all the cases described above. In Section 7, we compare our work with existing results in the research literature. In Section 8, we summarize the contributions of this paper and provide pointers to future research directions.

2 Motivating Examples

In this section, we present two simple examples motivating our research: planning a dinner according to different criteria, and determining the winner of an auction.

[1]Note that maximizing e is the same as minimizing $-e$, and hence, the assumption that objective functions are used only for minimization is not a limitation.

2.1 Planning a Dinner

Let us suppose that a chef is planning a dinner for a group of people. Dinner consists of three courses – an appetizer, an entree, and a dessert. However, different guests have different likes and dislikes. The job of the chef is to devise a dinner which satisfies the tastes of the guests, while at the same time, optimizing various criteria for the chef (e.g. minimizing cost, minimizing preparation time, etc.).

The relation `dish` is a 4-ary predicate consisting of the following fields: a `name` field specifying the name of the dish, a `type` field specifying whether it is an appetizer, an entree, or a dessert, a `cost` field estimating the unit (i.e. per serving) cost of the dish, and a `time` field estimating the time (in minutes) required to prepare the dish. An extensional database showing the `dish` relation is contained in Appendix A.1.

The relation `dislikes` is a binary predicate containing the name of a person and the name of a dish, indicating whether the person dislikes the dish. This information allows the chef to propose only menus that are compatible with all guests.

Models: For any given collection of guests expected at dinner and encoded through a unary relation `guest`, one or more *menu*s may be feasible. For example, it may turn out that both `Menu1` and `Menu2` listed below are possible dinners:

$$\texttt{Menu1}: \ \texttt{appetizer} = \texttt{caprese}; \texttt{entree} = \texttt{spaghetti\_alla\_carbonara}; \texttt{dessert} = \texttt{tiramisu}$$
$$\texttt{Menu2}: \ \texttt{appetizer} = \texttt{samosa}; \texttt{entree} = \texttt{matar\_paneer}; \texttt{dessert} = \texttt{rasgulla}$$

In this paper, we will argue that different *models* of a (possibly disjunctive) logic program neatly capture different menus in this example. In general, models represent scenarios that *could* be true, given certain information about the domain of discourse. Different scenarios could have different utilities to different people.

Utilities/Costs of Models: For example, the cost of preparing `Menu1` may be \$ 6.00 per person, while the cost of preparing `Menu2` may be \$ 5.00 per person. However, it may turn out that preparing `Menu1` takes 25 minutes, while preparing `Menu2` takes 60 minutes. In this case, the lazy chef who wishes to minimize the time he spends in the kitchen may prefer to prepare `Menu1`. On the other hand, the chef who wishes to minimize cost may well prefer `Menu2`.

Optimal Models: Informally speaking, a model is a way of satisfying the requirements encoded in a problem specification. As we have seen above, different solutions to the problem may have different costs/benefits to an end-user (i.e. to the person interested in solving the problem). However, what is important to a person is subjective and may vary from one individual to another. Consequently, any way of defining optimality of a model (or solution) must take the evaluation criterion (e.g. minimize cost of dinner, minimize preparation time) as an input.

2.2 Determining Winners in Combinatorial Auctions

A combinatorial auction is one where a set $\mathcal{O}$ of objects are for sale and there is a set $\mathcal{B}$ of bidders. Bidders may offer bids on a set of items. For example bidder b_1 may bid \$ 500 for items a, b, c together and \$ 200 for item a alone. Formally, a *bid* is a triple of the form (b, X, p) where $b \in \mathcal{B}$ is a bidder, $X \subseteq \mathcal{O}$ is a set of items, and $p > 0$ is a price. The auctioneer receives a set **bids** of bids. Without loss of generality, we may assume that **bids** does not contain two triples of the form $(-, X, -)$. This is because if two bids are received for the same set X of objects, the one with the lower price can be eliminated. [2]

Models: The task of the auctioneer is to determine which bids to "accept." Given a set **bids** of bids, a *potential winner* is a subset $\mathbf{win} \subseteq \mathbf{bids}$ such that

$$(b_1, X_1, p_1), (b_2, X_2, p_2) \in \mathbf{win} \rightarrow X_1 \cap X_2 = \emptyset.$$

Each potential winner set represents a possible scenario, i.e., a possible outcome of the auction.

Utilities/Costs of Models: Potential winners may be very different according to the auctioneer's point of view. Indeed, she clearly wants to maximize her revenue, and thus evaluates a model $\mathbf{win}$ according to the following measure:

$$R(\mathbf{win}) = \sum_{(b,X,p) \in \mathbf{win}} p.$$

Optimal Models: The auctioneer is interested in the potential winners $\mathbf{win}$ such that her revenue $R(\mathbf{win})$ is maximized. The *winner determination problem* is to find such an optimal potential winner.

In this paper, we will formalize the intuitive discussion given above so as to allow users to select models that are optimal from their point of view.

3 Preliminaries

3.1 Logic Programs

We assume the existence of an arbitrary logical language L generated by a finite set of constant, function and predicate symbols, and an infinite set of variable symbols.

Definition 3.1 If $A_1, \ldots, A_n, B_1, \ldots, B_m, B_{m+1}, \ldots B_{m+k}$ are atoms, then

$$A_1 \vee \cdots \vee A_n \quad \leftarrow \quad B_1 \,\&\, \ldots \,\&\, B_m \,\&\, \mathbf{not}(B_{m+1}) \,\&\, \ldots \,\&\, \mathbf{not}(B_{m+k}) \tag{1}$$

is a *rule*. A rule is *normal* if $n = 1$ and *positive* if $k = 0$. A rule with an empty body (i.e., $m = k = 0$) is called a *fact*. A *(normal, positive) logic program* is a finite set of (normal, positive) rules.

For a rule r, we denote by $H(r)$ (resp., $B(r)$) the set of literals occurring in the head (resp., body) of r.

A logic program may be viewed as composed of two kind of rules: (1) facts that define the so called extensional database predicates (EDB), and (ii) rules (typically with a non-empty body) that define other predicates, representing the intensional database (IDB). EDB predicates are often implemented as relations in a relational database. We often use the workds "relation" and "predicate" interchangeably. Thus, if we say that a relation r is an EDB relation for a program P, we mean that for each tuple $(v_1, \ldots, v_n) \in r$, a

[2] It is still possible that there are two bids of the form $(b_1, X, p_1), (b_2, X, p_2)$ with $p_1 = p_2$. In such a case most auctioneers eliminate one of the two bids using some pre-announced protocol – e.g. the bid received at a later time may be discarded.

fact $r(v_1, \ldots, v_n) \leftarrow$ belongs to P, even if it is not explicitly listed. These facts provide the definition of the EDB predicate r in the program P.

Given a logic program P, the Herbrand Base B_P of P is the set of all ground (i.e., variable-free) atoms that can be constructed by using the constants, predicates and function symbols appearing in P. Given a set X of literals, $\neg.X$ denotes the set containing the negation of the literals in X (the negation of a is $\neg a$ and vice versa). Moreover, X^+ denotes the set of atoms occurring as positive literals in X, and X^- the set of atoms whose negation is the set of negative literals in X. An interpretation I for a program P is any subset of $B_P \cup \neg.B_P$. Interpretation I satisfies a ground rule of the form shown in (1) above if either $I \cap H(r) \neq \emptyset$ or $B(r) \not\subseteq I$. Interpretation I satisfies a (non-ground) rule r if it satisfies each ground instance of r. We denote by I^+ the set of atoms occurring in I, and by I^- the set of atoms whose negation belongs to I. I is a *total* interpretation if $I^+ \cup I^- = B_P$. A set M of atoms is a *model* for P if the total interpretation $M \cup \neg.(B_P - M)$ satisfies each rule in P. Model M is a *minimal model* [19] for P if there is no model N for P such that $N \subset M$. Given a disjunctive logic program P, we use $\mathsf{MM}(P)$ and $\mathsf{MOD}(P)$ to denote the set of all minimal models of P and the set of all models of P, respectively. M is a *stable model* (or *answer set*) [7] of P if M is a minimal model of P^M where P^M is defined as follows:

$$\begin{array}{ll} P^M = \{ & A_1 \vee \cdots \vee A_n \leftarrow B_1 \,\&\, \ldots \,\&\, B_m \ : \\ & A_1 \vee \cdots \vee A_n \leftarrow B_1 \,\&\, \ldots \,\&\, B_m \,\&\, \mathbf{not}(D_1) \,\&\, \ldots \,\&\, \mathbf{not}(D_k) \text{ is a ground instance of} \\ & \text{a rule in } P \text{ and } \{D_1, \ldots, D_k\} \cap M = \emptyset \quad \}. \end{array}$$

We use $\mathsf{ST}(P)$ to denote the set of all stable models of P.

We now show how to express the cooking and auctions examples in this framework using the stable model semantics. In order to improve the readability of programs, we will also use rules without a head, which represents constraints to be satisfied in any allowed model. The following rule r

$$\leftarrow \quad B_1 \,\&\, \ldots \,\&\, B_m \,\&\, \mathbf{not}(D_1) \,\&\, \ldots \,\&\, \mathbf{not}(D_k)$$

means that its body should evaluate to false in every intended model of the program. Such constraints can be easily expressed under the stable model semantics [4]. In particular, r is shorthand for the following normal rule:

$$no\_stable \quad \leftarrow \quad B_1 \,\&\, \ldots \,\&\, B_m \,\&\, \mathbf{not}(D_1) \,\&\, \ldots \,\&\, \mathbf{not}(D_k) \,\&\, \mathbf{not}(no\_stable)$$

It is easy to see that in any stable model of every program containing this rule, the body of r should evaluate to false.

Example 3.1 The logic program associated with the cooking example is shown below:

$$\begin{array}{l} dinner(A, E, D) \leftarrow appetizer(A) \,\&\, entree(E) \,\&\, dessert(D) \\ appetizer(A) \vee not\_take(A) \leftarrow dish(A, appetizer, C, T) \\ entree(A) \vee not\_take(A) \leftarrow dish(A, entree, C, T) \\ dessert(A) \vee not\_take(A) \leftarrow dish(A, dessert, C, T) \\ not\_take(A) \leftarrow dish(A, \_, \_, \_), guest(P), dislikes(P, A) \\ \leftarrow appetizer(A) \,\&\, appetizer(B) \,\&\, A \neq B \\ \leftarrow entree(A) \,\&\, entree(B) \,\&\, A \neq B \\ \leftarrow dessert(A) \,\&\, dessert(B) \,\&\, A \neq B \\ chosen\_dinner \leftarrow dinner(A, E, D) \\ \leftarrow \mathbf{not}(chosen\_dinner) \end{array}$$

It is not difficult to see that each stable model M of this program contains exactly one appetizer, one dessert, and one entree. Appendix A.2 shows the stable models of this program with the EDB defined in Appendix A.1.

Example 3.2 Suppose $\mathbf{bids}$ is the set of all bids received in a combinatorial auction. We construct a disjunctive logic program $P_{\mathbf{bids}}$ as follows. Two EDB predicates *bids* and *requires* encode the input data about the bids received by the auctioneer. In particular, for each bid $(b, x, p) \in \mathbf{bids}$, $P_{\mathbf{bids}}$ contains the fact $bids(b, x, p) \leftarrow$, and the facts describing the set of items x required by the bidder b, namely, a fact $requires(x, i) \leftarrow$, for each item $i \in x$.

Moreover, an IDB predicate *wins* encodes the potential winner set of bids chosen by the auctioneer. It is defined by the following rule:

$$wins(B, X, P) \vee loses(B, X, P) \quad \leftarrow \quad bids(B, X, P).$$

Finally, we have the constraint:

$$\leftarrow \quad wins(B_1, X_1, P_1) \,\&\, win(B_2, X_2, P_2) \,\&\, X_1 \neq X_2 \,\&\, requires(X_1, I_1) \,\&\, requires(X_2, I_2)$$

It is easy to see that the stable models of $P_{\mathbf{bids}}$ correspond exactly to the potential winners of the auction.

3.2 Complexity Overview

In this section, we provide a brief overview of complexity classes — these classes will be used extensively later in the paper when we derive complexity results. The reader seeking details of complexity theory is referred to [23].

The classes Σ_k^P, Π_k^P and Δ_k^P of the Polynomial Hierarchy (PH) (cf. [28]) are defined as follows:

$$\Delta_0^P = \Sigma_0^P = \Pi_0^P = \mathrm{P} \quad \text{and for all } k \geq 1, \quad \Delta_k^P = \mathrm{P}^{\Sigma_{k-1}^P}, \;\; \Sigma_k^P = \mathrm{NP}^{\Sigma_{k-1}^P}, \;\; \Pi_k^P = \text{co-}\Sigma_k^P.$$

In particular, $\mathrm{NP} = \Sigma_1^P$, $\text{co-NP} = \Pi_1^P$, and $\Delta_2^P = \mathrm{P}^{\mathrm{NP}}$. Here P^C and NP^C denote the classes of problems that are solvable in polynomial time on a deterministic (resp. nondeterministic) Turing machine with an oracle for any problem π in the class C.

The classes Δ_k^P, $k \geq 2$, have been refined by the class $\Delta_k^P[O(\log n)]$, in which the number of calls to the oracle in each computation is bounded by $O(\log n)$, where n is the size of the input.

Later in the paper, we will establish a number of complexity results in which known problems that are hard for these classes are polynomially reduced to the problem we are studying. Many of these "known" problems will relate to Quantified Boolean Formulas (QBFs) and hence we introduce them here. A QBF is an expression of the form

$$Q_1 X_1 Q_2 X_2 \cdots Q_k X_k \, E, \quad k \geq 1, \tag{2}$$

where E is a Boolean expression whose atoms are from pairwise disjoint nonempty sets of variables $X_1, \ldots, X_k$, and the Q_i's are alternating quantifiers from $\{\exists, \forall\}$, for all $i = 1, \ldots, k$. If $Q_1 = \exists$ the we say the QBF is k-existential, otherwise it is k-universal. *Validity* of QBFs is recursively defined in the obvious way. We use $\mathrm{QBF}_{k,\exists}$ (resp., $\mathrm{QBF}_{k,\forall}$) to denote the set of all valid k-existential (resp., k-universal) QBFs (2).

Given a k-existential QBF Φ (resp. a k-universal QBF Ψ), deciding whether $\Phi \in \mathrm{QBF}_{k,\exists}$ (resp. $\Psi \in \mathrm{QBF}_{k,\forall}$), is a classical Σ_k^P-complete (resp. Π_k^P-complete) problem.

The class Δ_k^P also has complete problems, for all $k \geq 2$. For example, given a formula E on variables $X_1, ..., X_n, Y_1, ..., Y_r$, $r \geq 0$, and a quantifier pattern $Q_1 Y_1, ..., Q_r Y_r$, let ϕ be the truth-value assignment to $X_1, \ldots, X_n$ that is lexicographically minimum with respect to $\langle X_1, ..., X_n \rangle$[3] and such that $Q_1 Y_1 \cdots Q_r Y_r E_\phi \in$ QBF$_{k-2,\forall}$ (where such a ϕ is known to exist). Then, the problem of deciding whether $\phi(X_n) = true$ is complete for Δ_k^P (cf. [33, 12]).[4]

The hardness of the problems is unaffected even if the following restrictions are required: (i) E in (2) is in conjunctive normal form and each clause contains three literals (3CNF) when $Q_k = \exists$, and (ii) E in (2) is in disjunctive normal form and each conjunct contains three literals (3DNF) when $Q_k = \forall$ [29].

4 Optimal Models

In this section, we introduce the notion of optimal model for logic programs. First, we formally define weight assignments to atoms. We then introduce aggregation strategies which provide aggregate information on a given model. Finally, we define optimal models with respect to a given semantics, i.e., with respect to a given family of intended models.

Definition 4.1 An *atomic weight assignment*, $\wp$, for a program P, is a map from the Herbrand Base B_P of P to $\mathbf{R}^+$, where $\mathbf{R}^+$ denotes the set of non-negative real numbers (including zero).

Example 4.1 Returning to our "Dinner" Example, the following maps $\wp_1, \wp_2$ are atomic weight assignments:

Atom A	$\wp_1(A)$	$\wp_2(A)$
entree(E)	C if $dish(E, entree, C, T)$ holds.	T if $dish(E, entree, C, T)$
dessert(D)	C if $dish(D, dessert, C, T)$ holds.	T if $dish(D, dessert, C, T)$
appetizer(A)	C if $dish(A, appetizer, C, T)$ holds.	T if $dish(A, appetizer, C, T)$
other atoms	0	0

Intuitively, the first function assigns weights to entrees based on the price/cost of those entrees while the second function assigns weights based on the time taken to prepare those entrees.

Example 4.2 For the auction example, the following map $\wp_3$ is an atomic weight assignment:

Atom A	$\wp_3(A)$
loses(B,X,P)	P
other atoms	0

Weights are assigned to atoms *loses* corresponding to the bids that cannot be accepted and hence yield no revenue to the auctioneer.

Given a set X, we use M^X to denote the set of all multisets whose elements are in X. Membership and inclusion between multisets are defined in the standard way.

[3] ϕ is lexicographically greater than ψ w.r.t. $\langle X_1, \ldots, X_n \rangle$ if $\phi(X_j) = true$, $\psi(X_j) = false$ for the least j such that $\phi(X_j) \neq \psi(X_j)$.

[4] QBF$_{0,\forall}$ = QBF$_{0,\exists}$ is the set of all variable-free true formulas.

Definition 4.2 An *aggregation strategy* $\mathcal{A}$ is a map from $M^{\mathbf{R}^+}$ to $\mathbf{R}$.

An aggregation strategy $\mathcal{A}$ is said to be *monotonic* if, for all $S_1, S_2 \in M^{\mathbf{R}^+}$, $S_1 \subseteq S_2$ implies that $\mathcal{A}(S_1) \leq \mathcal{A}(S_2)$. The set of aggregation strategies may be ordered as follows: $\mathcal{A}_1 \leq \mathcal{A}_2$ if for all $S, \mathcal{A}_1(S) \leq \mathcal{A}_2(S)$. We list below, some sample aggregation strategies:

1. $\texttt{count}(S)$: This is defined as the number of elements in S. Clearly, this aggregation strategy is monotonic.
2. $\texttt{sum}(S)$: This aggregation strategy sums up all elements in S. It is clearly monotonic.
3. $\texttt{prod}(S)$: This aggregation strategy takes the product of all elements in S. It is monotonic if we require that the numbers in S are greater than or equal to 1.
4. $\texttt{avg}(S)$: This computes the average of all elements in S. This aggregation strategy is *not* monotonic.
5. $\texttt{min}(S)$, $\texttt{max}(S)$: This computes the smallest (resp. largest) element of S. $\texttt{min}(S)$ is not monotonic, but $\texttt{max}(S)$ is.

Definition 4.3 Suppose $\mathcal{A}$ is an aggregation strategy and $\wp$ is an atomic weight assignment. The *Herbrand Objective Function*, $\mathsf{HOF}(\wp, \mathcal{A})$ is a map from 2^{B_P} to $\mathbf{R}$ defined as follows:

$$\mathsf{HOF}(\wp, \mathcal{A})(M) \quad = \quad \mathcal{A}\left(\{\wp(A) \mid A \in M\}\right).$$

Intuitively, $\mathsf{HOF}(\wp, \mathcal{A})(M)$ looks at each ground atom $A \in M$, computes $\wp(A)$, and puts $\wp(A)$ in a multiset. It then applies the aggregation strategy to the multiset created in this way. This is illustrated by the following example.

Example 4.3 Suppose we consider the weights $\wp_1$ and $\wp_2$ described in Example 4.1. Consider the stable models M_1 and M_2 of Appendix A.2. If we use the aggregation strategy $\texttt{sum}$, then:

$\mathsf{HOF}(\wp_1, \texttt{sum})(M_1) = 6.8$ $\qquad$ $\mathsf{HOF}(\wp_1, \texttt{sum})(M_2) = 5.15$
$\mathsf{HOF}(\wp_2, \texttt{sum})(M_1) = 60$ $\qquad$ $\mathsf{HOF}(\wp_2, \texttt{sum})(M_2) = 70$

The following result says that, whenever a monotonic aggregation function $\wp$ is considered, the function $\mathsf{HOF}(\wp, \mathcal{A})$ is also monotonic.

Proposition 4.1 *Suppose $\mathcal{A}$ is any monotonic aggregation function. Then $\mathsf{HOF}(\wp, \mathcal{A})$ is monotonic, i.e. for all M, M', if $M \subseteq M'$ then $\mathsf{HOF}(\wp, \mathcal{A})(M) \leq \mathsf{HOF}(\wp, \mathcal{A})(M')$.*

We are now ready to define what it means for a model of a (possibly disjunctive) logic program P to be *optimal* with respect to $\wp$ and $\mathcal{A}$ and a selected family of models. Note that all semantics for logic programs identify a "family" of models (e.g. stable semantics identifies all stable models of logic programs, minimal model semantics identifies all minimal models of a logic program, and so on).

Definition 4.4 Let P be a logic program, $\wp$ an atomic weight assignment, and $\mathcal{A}$ an aggregation strategy. Suppose that $\mathcal{F}$ is a family of models of P. We say that M is an *optimal $\mathcal{F}$-model* of P w.r.t. $(\wp, \mathcal{A})$ if:

1. $M \in \mathcal{F}$, and

2. there is no model M' of P in $\mathcal{F}$ such that $\mathsf{HOF}(\wp, \mathcal{A})(M') < \mathsf{HOF}(\wp, \mathcal{A})(M)$.

We use the notation $\underline{\mathtt{Opt}}(P, \mathcal{F}, \wp, \mathcal{A})$ to denote the set of all optimal $\mathcal{F}$-models of P w.r.t. $(\wp, \mathcal{A})$.

We will often use the expressions *optimal model*, optimal *optimal minimal model*, and *optimal stable model* to denote models that are optimal w.r.t. the families $\mathsf{MOD}(P)$, $\mathsf{MM}(P)$ and $\mathsf{ST}(P)$, respectively.

Example 4.4 Consider again our "Dinner" example. The different possible choices for dishes lead to 16 stable models for the cooking logic program P, all listed in Appendix A.2. However, if we assign weights to dishes according to $\wp_1$, i.e. according to the price/cost of those dishes, we get a unique optimal stable model w.r.t. $(\mathtt{sum}, \wp_1)$:[5]

$$\begin{aligned}\underline{\mathtt{Opt}}(P, \mathsf{ST}(P), \wp_1, \mathtt{sum}) \;=\{ \;\; &\{dinner(caprese, idli, rasgulla), appetizer(caprese), entree(idli), \\ & dessert(rasgulla), \ldots\} \;\;\; \}.\end{aligned}$$

On the other hand, if we prefer to minimize cooking time, we can resort to the weight function $\wp_2$. In this case, we get the following optimal stable models w.r.t. $(\mathtt{sum}, \wp_2)$.
$\underline{\mathtt{Opt}}(P, \mathsf{ST}(P), \wp_2, \mathtt{sum}) = \{M_3, M_4, M_5, M_6\}$ where:

$$\begin{aligned}
M_3 &= \{dinner(caprese, spaghetti_carbonara, tiramisu), appetizer(caprese), \\
&\quad entree(spaghetti_carbonara), dessert(tiramisu), \ldots\}. \\
M_4 &= \{dinner(caprese, spaghetti_carbonara, rasgulla), appetizer(caprese), \\
&\quad entree(spaghetti_carbonara), dessert(rasgulla), \ldots\}. \\
M_5 &= \{dinner(caprese, matar_paneer, tiramisu), appetizer(caprese), \\
&\quad entree(matar_paneer), dessert(tiramisu), \ldots\}. \\
M_6 &= \{dinner(caprese, matar_paneer, rasgulla), appetizer(caprese), \\
&\quad entree(matar_paneer), dessert(rasgulla), \ldots\}.
\end{aligned}$$

Example 4.5 Consider the program $P_{\mathbf{bids}}$ in Example 3.2 and the weight assignment $\wp_3$ in Example 4.2. If we use the aggregation strategy $\mathtt{sum}$, the optimal models $\underline{\mathtt{Opt}}(P_{\mathbf{bids}}, \mathsf{ST}(P_{\mathbf{bids}}), \wp_3, \mathtt{sum})$ encode the potential winners that minimize the sum of the revenues for bids not accepted by the auctioneer, represented by the atoms with predicate *loses*. Therefore, these models encode the potential winners that maximize the revenue for the auctioneer, and thus exactly represent the best sets of bids to be accepted for the auction at hand.

5 Complexity Results

In this section we analyze the complexity of the main decision problems relating to optimal models. In particular, for different families $\mathcal{F}$ of models, P of logic programs, and assumptions on the monotonicity of the aggregation function used, we study the following problems:

Problem 1: (*Checking*) Given P, $\wp$, $\mathcal{A}$ and M as input, decide whether M is an optimal $\mathcal{F}$-model of P w.r.t. $(\wp, \mathcal{A})$.

[5]Note that, in this example, whenever we have to list a stable model of the program, we just list the atoms that characterize the model, namely, the atoms with predicates *dinner*, *appetizer*, *entree*, and *dessert* (in fact, just *dinner* would suffice, in this case). We do not list EDB atoms, which are true in every stable model, and auxiliary atoms, like *not_take*.

Problem 2: (*Cautious reasoning*) Given P, $\wp$, $\mathcal{A}$ and literal q as input, decide whether q is true in every optimal $\mathcal{F}$-model of P w.r.t. $(\wp, \mathcal{A})$, denoted by $P \models_c^{\mathcal{F}} q$.

Cautious reasoning is useful to single out the *necessary* consequences of the program. For example, the atom `appetizer(caprese)` is true in *all* models in $\underline{\mathsf{Opt}}(P, \mathsf{ST}(P), \wp_2, \mathtt{sum})$ as is easily seen in Example 4.4. This means that as long as the user wishes to minimize the time she spends on cooking, she is forced to choose `caprese` as the appetizer. On the other hand, `dessert(tiramisu)` is *not* true in all models in $\underline{\mathsf{Opt}}(P, \mathsf{ST}(P), \wp_2, \mathtt{sum})$.

Problem 3: (*Brave reasoning*) Given P, $\wp$, $\mathcal{A}$ and a ground literal q as input, decide whether there exists an optimal $\mathcal{F}$-model M of P w.r.t. $(\wp, \mathcal{A})$ such that q is true w.r.t. M, denoted by $P \models_b^{\mathcal{F}} q$.

Unlike cautious reasoning, *brave* reasoning finds out whether there exists an optimal model in which a literal is true. In the Dinner example (cf. example 4.4), we see that `dessert(tiramisu)` is a brave consequence of P, but not a cautious consequence.

Problem 4: ($Approx(P, \wp, \mathcal{A}, n, S)$). Given P, $\wp$, $\mathcal{A}$, a set of ground literals S, and a real number n as input, decide whether there exists a model M in $\mathcal{F}$ such that: (i) $\mathsf{HOF}(\wp, \mathcal{A})(M) \leq n$, and (ii) every literal in S is true w.r.t. M.

A successful solution to problem (4) allows the user to approximate optimal models.

Example 5.1 Problem (4) above has interesting consequences. Suppose the cook wishes to ask the following queries:

1. *"Is there some way for me to fix dinner at a price less than or equal to £ 6.50 her head?"* This corresponds to an instance of Problem 4 where $S = \emptyset$ and $n = 6.50$.
2. Suppose for some reason, the cook wants to make `samosa` as the appetizer. Then she may want to ask the query: *Is there some way for me to fix dinner including a samosa appetizer at a price less than or equal to £ 6.50 her head?"* This corresponds to an instance of Problem 4 where $S = \{appetizer(samosa)\}$ and $n = 6.50$.

We denote by $max\_hof(\wp, \mathcal{A}, P)$, the highest value that HOF may assume on P (given $\wp$ and $\mathcal{A}$), that is, $max\_hof(\wp, \mathcal{A}, P) = max(\{\mathsf{HOF}(\wp, \mathcal{A})(M) \mid M \subseteq B_P\})$. In the complexity analysis, we assume that the atomic weight assignment is part of the input and weights are (non-negative) integers. Moreover, we assume that, given $\wp$, $\mathcal{A}$, P, and M, both $\mathsf{HOF}(\wp, \mathcal{A})(M)$ and $max\_hof(\wp, \mathcal{A}, P)$ are polynomial-time computable (note that all sample strategies shown in Section 4 satisfy these assumptions). We analyze the complexity of the propositional case (that is, we assume that programs are ground); the results, however, can be easily extended to *data complexity* [31].

5.1 Overview of Results

We give a complete overview of the complexity results for optimal models that we formally prove in the following of the section, and we supply a brief discussion providing intuitive explainations of the results.

The complexity results are summarized in Table 1 and in Table 2. In particular, Table 1 shows the complexity for the case of general (disjunctive) programs where negation can appear in the rules' bodies; Table 2 collects the results for positive (disjunctive) programs where negation is disallowed. Its first column specifies

Models	Aggregation Strategy	Checking	Brave Reasoning	Cautious Reasoning	Approx
Stable	Monotonic	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
Stable	Arbitrary	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
Minimal	Monotonic	co-NP	Σ_2^P	Π_2^P	Σ_2^P
Minimal	Arbitrary	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
All	Monotonic	co-NP	Δ_2^P	Δ_2^P	NP
All	Arbitrary	co-NP	Δ_2^P	Δ_2^P	NP

Table 1: Disjunctive general programs

the family of models (Stable models, Minimal models, or All models); the second column specifies possible restrictions on the aggregation strategy (Monotonicity, Strict Monotonicity, Arbitrary = No restriction); the remaining columns (3 to 7) report the complexity results. Each of these columns refers to a specific task (Checking, Brave Reasoning, Cautious Reasoning, and Approx, in the specified order). All results we report are completeness results, i.e., when we say that a problem P has complexity C, then we have proved that P is C-complete under polynomial-time transformations.

Consider the results displayed in Table 1. Checking is Π_2^P-complete for stable models. Thus, checking if an interpretation I is an optimal stable model is located one level higher, in the polynomial hierarchy, than checking if it is an ordinary stable model (which is "only" co-NP-complete). Intuitively, this increase of complexity is because of two "orthogonal" sources of complexity: (i) the stability check (i.e., proving that the interpretation is a stable model); and (ii) the optimality check (i.e., proving that the value of the objective function on I is minimum in the family of the stable models of the program). Imposing monotonicity on the aggregation strategy does not help for stable models (the complexity remains Π_2^P), because there is no relationships between an interpretation I being stable and $\mathsf{HOF}(\wp, \mathcal{A})(I)$ being optimal. In this respect, the situation is different for Minimal Models. While Checking is Π_2^P-complete (as for stable models) under arbitrary aggregation strategies, its complexity decreases to co-NP (i.e., it becomes the same as the complexity of minimality checking) for monotonic aggregation strategies. Indeed, in this case, the two minimality criteria to be considered for the check (the minimality of the model and the optimality of its value under the objective function) work in parallel: if an interpretation A is smaller than B (i.e., it is preferable to B under the subset inclusion criterion determining minimal models), then $\mathsf{HOF}(\wp, \mathcal{A})(A) \leq \mathsf{HOF}(\wp, \mathcal{A})(B)$. Thus, the complexity of the two sources is not summed up in this case. For the family of All models, the complexity of Checking remains co-NP even if the aggregation strategy is arbitrary. The reason is that, compared to stable models or minimal models, one source of complexity is eliminated in this case, because checking membership in the family of all models is polynomial (while it is co-NP-complete for the families of stable models and minimal models). Thus, the only hard source of complexity is the optimality check which causes the co-NP-completeness of this problem.

Problem Approx is somehow complementary to Checking. To prove an instance of Approx, one has to find a member A of the models' family (e.g., a stable model or a minimal model), whose value of the objective function is lower than the given bound (and A contains the specified literals). Similarly, to disprove an instance of Checking, one has to do something similar: find a member A of the models' family, whose value of the objective function is lower than the value of the objective function on the model to be checked. According with this intuition, the complexity of Approx is complementary (Π_2^P/Σ_2^P, co-NP/NP) to the complexity of Checking, nearly always. The only exception is the case of Monotonic aggregation strategy for the family of Minimal models, where Approx cannot enjoy the "parallelism" among the two sources of complexity which mitigated the complexity of Checking for this case (see the comment above).

Reasoning (Brave and Cautious) brings an additional source of complexity, compared to the previous problems: the (possibly) exponential number of optimal models which should be analyzed (in the worst case) to decide whether the given literal is true or not. Nevertheless, in most cases, the applicability of smart techniques mitigates the complexity of these reasoning problems, allowing us to solve them deterministically by a polynomial number of calls to oracles solving Checking or Approx (an oracle for a problem P is like a subroutine taking an instance of P in input and returning its yes/no solution) – see, for instance, the proof of Theorem 5.7. Thus, compared to Checking, the complexity of Brave Reasoning and Cautious Reasoning increases only "mildly", it is located only half a level higher in the polynomial hierarchy, stepping from co-NP and Π_2^P to Δ_2^P and Δ_3^P, respectively. An interesting exception is reasoning with the family of Minimal models under monotonic aggregation strategies. In this case, the problem remains of the same complexity as standard minimal model reasoning (Σ_2^P and Π_2^P for brave and cautious reasoning, respectively).

Models	Aggregation Strategy	Checking	Brave Reasoning	Cautious Reasoning	Approx
Stable, Minimal	Monotonic	co-NP	Σ_2^P	Π_2^P	Σ_2^P
Stable, Minimal	Arbitrary	Π_2^P	Δ_3^P	Δ_3^P	Σ_2^P
All	Monotonic	co-NP	Δ_2^P	Δ_2^P	NP
All	Arbitrary	co-NP	Δ_2^P	Δ_2^P	NP

Table 2: Disjunctive positive programs

Table 2 which shows complexity results for positive programs contains no major surprises. As disjunction is allowed, disallowing negation does not make any difference as far as both Minimal model semantics and All model semantics are concerned because these semantics are syntax independent (unlike Stable model semantics, $a \leftarrow \mathbf{not}\ b$ is precisely the same as $a \vee b$ for Minimal models and All models semantics). Thus, the complexity results on positive programs for the families of Minimal models and All models is precisely the same as for general programs. Disallowing negation, however, makes a difference for Stable model semantics and lowers the complexity over monotonic aggregation strategies, which becomes the same as for the case of minimal models (indeed, Stable models degenerate to Minimal models when negation is disallowed).

5.2 General reduction among problems

We first show that, for any of *all, stable, minimal* family of models, brave and cautious reasoning problems have the same computational complexity, if we consider general aggregation strategies.

Theorem 5.1 *(Brave Reasoning vs Cautious Reasoning under arbitrary aggregation) Let P be a program, $\mathcal{F}$ a family of models in* $\{$all, stable, minimal$\}$*, $\wp$ a weight function, and $\mathcal{A}$ an aggregation strategy. Then there* *exists a program P', a weight function $\wp'$, and an arbitrary (possibly non monotonic) aggregation strategy $\mathcal{A}'$, all polynomially constructible, s.t.* Brave reasoning *for P w.r.t. $(\wp, \mathcal{A})$ reduces to* Cautious reasoning *for P' w.r.t. $(\wp', \mathcal{A}')$, and viceversa.*

PROOF. Let P be a logic program, q an atom in B_P, $\wp$ an atomic weight assignment, and $\mathcal{A}$ an aggregation strategy. Moreover, let q' be a fresh atom. Define a program P', an atomic weight assignment $\wp'$, and two aggregation strategies $\mathcal{A}'$ and $\mathcal{A}''$ as follows.

$$P' = P \cup \{q \leftarrow q';\ q' \leftarrow q\}$$

$$\begin{cases} \wp'(p) = \wp(p) & \forall p \neq q' \\ \wp'(q') = c_{q'} & \text{where } c_{q'} \neq \wp'(p)\ \forall p \neq q' \end{cases}$$

$$\mathcal{A}'(X) = \begin{cases} A(X) & \text{if } c_{q'} \notin X \\ A(X - \{c_{q'}\}) + 1 & \text{if } c_{q'} \in X \end{cases}$$

$$\mathcal{A}''(X) = \begin{cases} A(X - \{c_{q'}\}) & \text{if } c_{q'} \in X \\ A(X) + 1 & \text{if } c_{q'} \notin X \end{cases}$$

Then, it is easy to verify that the following claims hold.

CLAIM a). $P \models_c^{\mathcal{F}} q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_b^{\mathcal{F}} q'$ w.r.t. $(\wp', \mathcal{A}')$.

CLAIM b). $P \models_c^{\mathcal{F}} \neg q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_b^{\mathcal{F}} \neg q'$ w.r.t. $(\wp', \mathcal{A}'')$.

CLAIM c). $P \models_b^{\mathcal{F}} q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_c^{\mathcal{F}} q'$ w.r.t. $(\wp', \mathcal{A}'')$.

CLAIM d). $P \models_b^{\mathcal{F}} \neg q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_c^{\mathcal{F}} \neg q'$ w.r.t. $(\wp', \mathcal{A}')$. □

Note that the reduction above does not preserve the monotonicity of the aggregation strategy, i.e., even if $\mathcal{A}$ is a monotonic aggregation strategy, $\mathcal{A}'$ may, in general, not be monotonic. However, for stable model semantics and all model semantics, brave and cautious reasoning have the same complexity even w.r.t. monotonic aggregation strategies, as stated by the following theorem.

Theorem 5.2 *(Brave Reasoning vs Cautious Reasoning under monotonic aggregation) Let P be a general program, $\mathcal{F}$ a family of models in* {all, stable}*, $\wp$ a weight function, and $\mathcal{A}$ a monotonic aggregation strategy. Then, there exist a program P', a weight function $\wp'$, and a monotonic aggregation strategy $\mathcal{A}'$, all of them polynomially constructible, s.t.* Brave reasoning *for P w.r.t. $(\wp, \mathcal{A})$ reduces to* Cautious reasoning *for P' w.r.t. $(\wp', \mathcal{A}')$, and viceversa.*

There exists a very natural relation between traditional – i.e., non optimal – brave reasoning and the approximation problem. This is stated by the following proposition, whose proof is straightforward.

Proposition 5.1 *(Brave Reasoning vs Approx) Let P be a program, L a literal, and $\mathcal{F}$ a family of models. There exists an $\mathcal{F}$-model M for P s.t. L is true w.r.t. M if and only if $(P, \wp_0, sum, 0, \{L\})$ is a yes-instance of* Approx*, where $\wp_0$ denotes the weight function which assigns 0 to each atom in B_P.*

Thus, for any family of models, traditional brave reasoning reduces to *Approx*. Note that the aggregation strategy used in the above reduction is monotonic.

Moreover traditional brave reasoning (without objective functions) is clearly a special case of optimal brave reasoning. To see this, consider any trivial aggregation strategy which assigns some fixed value to every given set. Then the Herbrand objective function is a constant function and all the models in the given family play the same role in the reasoning task, as in the case of traditional reasoning. Clearly, the same argument applies to the relationship between traditional cautious reasoning and optimal cautious reasoning.

5.3 Results for Stable Models

In this section, we study the complexity of checking, cautious reasoning, brave reasoning, and approximate reasoning problems with respect to the stable model semantics.

Checking

Theorem 5.3 *(General/Positive Program, Arbitrary Aggregation) Given a disjunctive program P, $\wp$, $\mathcal{A}$ and M as input, deciding whether M is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Π_2^P-complete. Hardness holds even if P is a positive program and M is a stable model of P.*

PROOF. Π_2^P-*Membership*. We can verify that M is not an optimal stable model as follows. Guess $M_1 \subseteq B_P$, and check that: either (1) [M_1 is a stable model of P] and [$(\wp, \mathcal{A})(M_1) < \mathsf{HOF}(\wp, \mathcal{A})(M)$], or (2) M is not a stable model of P. Since both (1) and (2) can be done by a single call to an NP oracle [3], this problem is in Σ_2^P, and, as a consequence, Checking is in Π_2^P.

Π_2^P-*Hardness*. Given a disjunctive positive program P and a literal $\neg q$, deciding whether $\neg q$ is a cautious consequence of P under stable semantics (i.e., deciding whether q does not belong to any stable model) is Π_2^P-hard [3]. We reduce this problem to optimal stable model checking.

Let q' be a fresh atom and P' be the (positive) program obtained from P by: (i) inserting the atom q' in the head of every rule of P. It is easy to see that

$$\mathsf{MM}(P') = \{q'\} \cup \mathsf{MM}(P)$$

Since both P and P' are positive programs, $\mathsf{ST}(P) = \mathsf{MM}(P)$ and $\mathsf{ST}(P') = \mathsf{MM}(P')$. Therefore:

$$\mathsf{ST}(P') = \{q'\} \cup \mathsf{ST}(P)$$

Let $\wp$ be a one-to-one function from B_P into $[1...|B_P|]$ ($\wp$ assigns an identifier to each atom). Define $\mathcal{A}$ as follows: (i) $\mathcal{A}(K) = 0$ if K is the image (under $\wp$) of a set N of atoms containing q, (ii) $\mathcal{A}(K) = 1$ otherwise (i.e., N does not contain q). (Note that $\mathcal{A}$ is not monotonic.) Now, we have that: (i) $\mathsf{HOF}(\wp, \mathcal{A})(\{q'\}) = 1$, and, (ii) for each $M \in (\mathsf{ST}(P') - \{q'\})$ s.t. $q \in M$, $\mathsf{HOF}(\wp, \mathcal{A})(M) = 0$. Therefore, [$\{q'\}$ is an optimal stable model of P'] if and only if [no stable model of P contains q]. That is, $\{q'\}$ is an optimal stable model of P' iff $\neg q$ is a cautious consequence of P. Hence, Checking is Π_2^P-hard; moreover, hardness holds even for positive programs, as the used program P' is positive. Note that $\{q'\}$ is a stable model of P'. $\Box$

Hence, optimal stable model checking for disjunctive programs is Π_2^P-complete and the complexity of the problem does not decrease on positive programs. The aggregation strategy used in the proof of Π_2^P-hardness is arbitrary (not monotonic). One may thus wonder whether assuming the monotonicity of this strategy has an impact on the complexity of optimal stable model checking. The following result says that the answer to this question is "no."

Theorem 5.4 *(General Program, Monotonic Aggregation) Given a disjunctive program P, $\wp$, $\mathcal{A}$ and M as input, deciding whether M is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Π_2^P-complete even if $\mathcal{A}$ is a monotonic aggregation strategy. Hardness holds even if M is a stable model of P.*

PROOF. ¿From Theorem 5.3, it remains to prove only hardness. Given a (general) disjunctive program P and an atom q, deciding whether q is a cautious consequence of P under stable model semantics (i.e., deciding whether q belongs to all stable model) is Π_2^P-hard [3]. We reduce this problem to optimal stable model checking with monotonic aggregation functions.

Let P' be the program obtained from P by: (i) inserting the literal $\mathbf{not}(q)$ in the body of every rule of P, and, (ii) adding the rule $q \vee q' \leftarrow$, where q' is a fresh atom. It is easy to see that

$$\mathsf{ST}(P') = \{q\} \cup \{M \cup \{q'\} \mid M \in \mathsf{ST}(P) \wedge q \notin M\}$$

Let $\mathcal{A} = sum$ and $\wp$ be the following atomic weight assignment: (i) $\wp(q) = 1$, and (ii) $\wp(x) = 0$, for each $x \in B_{P'} - \{q\}$. Now, we have that: (i) $\mathsf{HOF}(\wp, sum)(\{q\}) = 1$, and, (ii) for each $M \in (\mathsf{ST}(P') - \{q\})$, $\mathsf{HOF}(\wp, \mathcal{A})(M) = 0$. Therefore, [$\{q\}$ is an optimal stable model of P'] if and only if [it is the only stable model of P'] if and only if [every stable model of P contains q]. That is, $\{q\}$ is an optimal stable model of P' iff q is a cautious consequence of P (we are done). Hence, Checking is Π_2^P-hard even if the monotonicity of aggregation strategy is imposed (as we have used the monotonic strategy sum). □

In order to decrease the complexity of optimal stable model checking, we have to assume both that the programs involved are positive and that the aggregation strategy involved is monotonic.

Theorem 5.5 *(Positive Program, Monotonic Aggregation) Given a positive disjunctive program P, $\wp$, M, and a monotonic aggregation strategy $\mathcal{A}$ as input, deciding whether M is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is* co-NP*-complete. Hardness holds even if M is known to be a stable model.*

Approx

We now show that the approximate reasoning problem is computationally intractable w.r.t. stable model semantics.

Theorem 5.6 *(General/Positive Program, Arbitrary Aggregation) Under stable model semantics, given a disjunctive general program P, $\wp$, $\mathcal{A}$, a set of ground literals S and a real number n as input, $Approx(P, \wp, \mathcal{A}, n, S)$ is Σ_2^P-complete. Hardness holds even if P is a positive program.*

PROOF. We can decide the problem as follows. Guess $M \subseteq B_P$, and check that: (1) M is a stable model of P, (2) $\mathsf{HOF}(\wp, \mathcal{A})(M) \leq n$, and (3) every literal in S is true w.r.t. M. Clearly, property (2) and (3) can be checked in polynomial time; while (1) can be decided by a single call to a NP oracle. The problem is therefore in Σ_2^P.

Hardness follows from Proposition 5.1 and from the Σ_2^P-hardness of (traditional) brave reasoning for disjunctive positive programs under stable model semantics [3]. □

Reasoning

In this section, we will study the complexity of brave and cautious reasoning under the optimal stable model semantics. Several hardness results will be proved by reductions from QBFs into problems related to optimal models; the disjunctive programs used will be appropriate adaptations and extensions of the disjunctive program reported below (which was first described in [3]).

Let Φ be a formula of the form $\forall Y\, E$, where E is a Boolean expression over propositional variables from $X \cup Y$, where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$. We assume that E is in 3DNF, i.e., $E = D_1 \vee \cdots \vee D_r$ and each $D_i = L_{i,1} \wedge L_{i,2} \wedge L_{i,3}$ is a conjunction of literals $L_{i,j}$. Define the following DLP $LP(\Phi)$:

$$
\begin{array}{llll}
x_i \vee x_i' \leftarrow & & & \text{for each } i = 1, \ldots, n \\
y_j \vee y_j' \leftarrow & y_j \leftarrow w & y_j' \leftarrow w & \text{for each } j = 1, \ldots, m \\
w \leftarrow y_j \wedge y_j' & & & \text{for each } j = 1, \ldots, m \\
w \leftarrow \sigma(L_{k,1}) \wedge \sigma(L_{k,2}) \wedge \sigma(L_{k,3}) & & & \text{for each } k = 1, \ldots, r
\end{array}
$$

where σ maps literals to atoms as follows:

$$
\sigma(L) = \begin{cases} x_i' & \text{if } L = \neg x_i \text{ for some } i = 1, \ldots, n \\ y_j' & \text{if } L = \neg y_j \text{ for some } j = 1, \ldots, m \\ L & \text{otherwise} \end{cases}
$$

Intuitively, x_i' corresponds to $\neg x_i$ and y_j' corresponds to $\neg y_j$. It is very important to note that $LP(\Phi)$ *is always positive*.

Given a truth assignment $\phi(X)$ to $X = \{x_1, \ldots, x_n\}$, we denote by $M_\phi \subseteq B_{LP(\Phi)}$ the following interpretation

$$
M_\phi = \{x_i \mid \phi(x_i) = true\} \cup \{x_i' \mid \phi(x_i) = false\} \cup \{w\} \cup \{y_1, \ldots, y_m\} \cup \{y_1', \ldots, y_m'\}.
$$

Moreover, given an interpretation M of $LP(\Phi)$, we denote by ϕ_M the truth assignment to $X = \{x_1, \ldots, x_n\}$:

$$
\phi_M(x_i) = true \text{ iff } x_i \in M.
$$

Let E be a Boolean expression and ϕ be a truth assignment for the variables in the set X. $E_{\phi(X)}$ denotes the Boolean expression E where each variable $x \in X$ is replaced by its truth value $\phi(x)$.

Lemma 5.1 *Let $\Phi = \forall Y\, E$ and $LP(\Phi)$ be the formula and the disjunctive logic program defined above. Consider the set A of the truth assignments $\phi(X)$ to $X = \{x_1, \ldots, x_n\}$ such that $\Phi_\phi = \forall Y\, E_{\phi(X)}$ is a valid formula, and let B be the set of the stable models of $LP(\Phi)$ which contain w. Then, there is a one-to-one correspondence between A and B. In particular:*

1. *if $\Phi_\phi = \forall Y\, E_{\phi(X)} \in QBF_{1,\forall}$, then $M_\phi \in \mathsf{ST}(LP(\Phi))$, and*
2. *if $M \in \mathsf{ST}(LP(\Phi))$ contains w, then $\Phi_{\phi_M} \in QBF_{1,\forall}$.*

PROOF. Follows immediately from Theorems 3.3 and 3.5 of [3], which precisely state the correspondence between such a formula Φ and its associated program $LP(\Phi)$. $\square$

Note that $LP(\Phi)$ is constructible from Φ in polynomial time and that $\mathsf{ST}(LP(\Phi)) = \mathsf{MM}(LP(\Phi))$, as $LP(\Phi)$ is a positive program.

We are now ready to derive the complexity of Brave reasoning with optimal models.

Theorem 5.7 *(General/Positive Program, Arbitrary Aggregation) Given a disjunctive program P, $\wp$, $\mathcal{A}$ and a ground literal q as input, deciding whether q is true in some optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Δ_3^P-complete. Hardness holds even if P is a positive program.*

PROOF. Δ_3^P-*Membership*. We first call a Σ_2^P oracle to verify that P admits stable models (otherwise, q cannot be a brave consequence). We compute then $k = max\_hof(\wp, \mathcal{A}, P)$ (this is done in polynomial time by assumption). After that, by binary search on $[0..k]$, we determine the cost Σ of the optimal stable models, by a polynomial number of calls to a (Σ_2^P) oracle deciding $approx(P, \wp, \mathcal{A}, n, \emptyset)$ ($n = k/2$ on the first call; then if the oracle answers yes", $n = k/4$; otherwise, n is set to $k/2 + k/4$, and so on, according with standard binary search). (Observe that the number of calls to the oracle is logarithmic in k, and, as a consequence, is polynomial in the size of the input.) Finally, a call to the oracle $approx(P, \wp, \mathcal{A}, \Sigma, \{q\})$ verifies that q is true in some optimal stable model of P.

Δ_3^P-*Hardness*. Let $\langle x_1, ..., x_n\rangle$ be in lexicographical order. Then, Δ_3^P-hardness is shown by a reduction from deciding whether the lexicographically minimum truth assignment $\phi(X)$, $X = \{x_1, \ldots, x_n\}$, such that $\Phi_\phi = \forall Y\ E_{\phi(X)}$ is valid, satisfies $\phi(x_n) = true$ (where such a ϕ is known to exists). W.l.o.g. we assume that E is in 3DNF of the form defined above.

Consider the (positive) program $LP(\Phi)$ defined above. Let $\mathsf{ST}_w(LP(\Phi))$ denote the set of the stable models of $LP(\Phi)$ which contain w. ¿From Lemma 5.1 we know that there is a one-to-one correspondence between $\mathsf{ST}_w(LP(\Phi))$ and the set of truth assignments ϕ which make Φ_ϕ valid.

Now, let $\wp$ be the atomic weight assignment such that: (i) $\wp(x_i) = 2^{n-i}$ $(1 \leq i \leq n)$, (ii) $\wp(w) = 2^n$, and (iii) $\wp(y) = 0$, if $y \notin \{w\} \cup \{x_1, \ldots, x_n\}$. Moreover, let $\mathcal{A}$ be the aggregation strategy defined as follows: (i) $\mathcal{A}(X) = 2^n$ if $2^n \notin X$, (ii) $\mathcal{A}(X) = sum(X) - 2^n$, if $2^n \in X$.

Then, the Herbrand objective function $\mathsf{HOF}(\wp, \mathcal{A})$ assigns 2^n to the stable models which do not contain w; while it assigns the sum of the weights of the x_is in M to each stable model M containing w (the models in $\mathsf{ST}_w(LP(\Phi))$ are thus the candidates for optimal models, as their values is less than 2^n). $\mathsf{HOF}(\wp, \mathcal{A})$ induces a total order on $\mathsf{ST}_w(LP(\Phi))$. In particular, given two stable models M and M' in $\mathsf{ST}_w(LP(\Phi))$, $\mathsf{HOF}(\wp, \mathcal{A})(M) > \mathsf{HOF}(\wp, \mathcal{A})(M')$ iff the truth assignment ϕ_M is greater than $\phi_{M'}$ in the lexicographically order. Therefore, $LP(\Phi)$ has a unique optimal stable model M (actually, M is in $\mathsf{ST}_w(LP(\Phi))$), corresponding to the lexicographically minimum truth assignment ϕ_{min} such that $\Phi_{\phi_{min}} = \forall Y\ E_{\phi_{min}(X)}$ is valid. Hence, the lexicographically minimum truth assignment $\phi_{min}(X)$ making $\Phi_{\phi_{min}}$ valid fulfills $\phi_{min}(x_n) = true$ if and only if x_n is true in the optimal stable model of $LP(\Phi)$ w.r.t. $(\wp, \mathcal{A})$ (that is, iff x_n is a brave consequence of $LP(\Phi)$ w.r.t. $(\wp, \mathcal{A})$). Therefore, brave reasoning is Δ_3^P-hard. Moreover, hardness holds even if the logic program is positive as the utilized program $LP(\Phi)$ positive. □

The following result says that even if we require the aggregation strategy to be monotonic, the hardness result presented above continues to persist. Thus, the complexity does not decrease in the case of monotonic aggregation strategies.

Theorem 5.8 *(General Program, Arbitrary/Monotonic Aggregation) Given a disjunctive program P, $\wp$, $\mathcal{A}$ and a ground literal q as input, deciding whether q is true in some optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Δ_3^P-complete even if $\mathcal{A}$ is a monotonic aggregation strategy.*

Remark. Note that if the weight assignments are required to be in unary (tally) notation, then $O(log\ n)$ oracle calls suffice to determine the cost Σ of the optimal model. In this case, the problem ends up being $\Delta_2^P[O(\log n)]$-complete.

If the aggregation strategy is monotonic *and* the program is a disjunctive positive program, then brave and cautious reasoning have different complexity (unless the polynomial hierarchy collapses at its second level). Indeed the following result shows that for positive programs with monotonic aggregation strategies the brave reasoning problem is Σ_2^P-complete and the cautious reasoning problem is Π_2^P-complete.

Theorem 5.9 *(Positive Program, Monotonic Aggregation, Reasoning) Given a disjunctive positive program P, a weight function $\wp$, a monotonic aggregation strategy $\mathcal{A}$, and a ground literal q as input, deciding whether q is true in some (resp., in every) optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Σ_2^P-complete (resp., Π_2^P-complete).*

5.4 Results for Minimal Models

Minimal model semantics is syntax independent. For any disjunctive general program P, there exists a corresponding disjunctive positive program having the same semantics. Define the *positive version* $PV(P)$ of P as the following disjunctive positive program:

$$\begin{aligned} PV(P) = \{ \quad & A_1 \vee \cdots \vee A_n \vee B_1 \vee \cdots \vee B_m \leftarrow D_1, \ldots, D_k \mid \\ & A_1 \vee \cdots \vee A_n \leftarrow D_1, \ldots, D_k, \mathbf{not}(B_1), \ldots, \mathbf{not}(B_m) \text{ is a rule of } P \ \} \end{aligned}$$

Note that $PV(P)$ is computable in linear-time from P. Clearly, P and $PV(P)$ have the same set of models, and hence the same set of minimal models. Thus, considering only disjunctive positive programs is no longer an actual restriction, because any general program is equivalent (w.r.t. minimal model semantics) to a disjunctive positive program. Moreover, for positive programs, stable and minimal models coincide. As a consequence, for any problem Π, all the complexity results we are interested in can be immediately derived from the corresponding results we proved for the problem Π for disjunctive positive programs under stable model semantics.

To save space and readability of the paper, we do not provide formal statements about the computational complexity of the considered problems (Checking, Brave reasoning, Cautious reasoning, $Approx$) under minimal model semantics. However, Table 1 and Table 2 show the computational complexity of these problems.

5.5 Results for All Models

If we consider all models, we get the lowest complexity results for all problems. First, we define a disjunctive positive program P_E which can be polynomially computed from a given Boolean formula E, and which is useful for all the reductions of this section.

Let E be a CNF Boolean formula, i.e., $E = C_1 \wedge C_2 \wedge \cdots C_r$ and each $C_i = L_{i,1} \vee \cdots \vee L_{i,t_i}$ is a disjunction of literals. Moreover, let $\{x_1, \ldots x_n\}$ be the set of Boolean variables occurring in E. P_E is the following disjunctive positive program:

$$
\begin{array}{ll}
x_i \vee x'_i \leftarrow; & \text{for each } i = 1, \ldots, n \\[6pt]
contr \leftarrow x_i \wedge x'_i; & \text{for each } i = 1, \ldots, n \\[6pt]
contr \leftarrow \sigma(L_{k,1}) \wedge \cdots \wedge \sigma(L_{k,t_k}) & \text{for each } k = 1, \ldots, r \\
x_i \leftarrow contr & \text{for each } i = 1, \ldots, n \\
x'_i \leftarrow contr & \text{for each } i = 1, \ldots, n
\end{array}
$$

where σ maps literals to atoms as follows:

$$
\sigma(L) = \begin{cases} x_i & \text{if } L = \neg x_i \text{ for some } i = 1, \ldots, n \\ x'_i & \text{if } L = x_i \text{ for some } i = 1, \ldots, n \end{cases}
$$

There is a one-to-one correspondence between the truth assignments to X satisfying E and the models of P_E which do not contain the atom $contr$. In particular, given a model M of P_E s.t. $contr \notin M$, the truth assignment ϕ_M satisfies E, where $\phi_M(x_i) = true$ if $x_i \in M$, and $\phi_M(x_i) = false$ if $x'_i \in M$. Viceversa, given a truth assignment ϕ satisfying E, the set of atoms I_ϕ is a model of P_E, where

$$
I_\phi = \{x_i \in \{x_1, \ldots, x_n\} \mid \phi(x_i) = true\} \cup \{x'_i \in \{x'_1, \ldots, x'_n\} \mid \phi(x_i) = false\}
$$

Note that $contr$ does not belong to I_ϕ.

On the other hand, a model for P_E containing the atom $contr$ encodes a contradictory truth assignment for E, i.e. a truth assignment which violates some conjunct C_j or assigns the value *true* to both x_i and x'_i (which encodes $\neg x_i$). Moreover, $contr$ forces each atom of B_{P_E} to be true. Hence, any unsatisfiable truth assignment for E maps to the model B_{P_E} of P_E.

Checking

Our first result is that determining whether an arbitrary interpretation M is an optimal model of a disjunctive logic program P is co-NP-complete. Note that in contrast, determining whether M is an optimal stable/minimal model is Π_2^P-complete.

Theorem 5.10 *(General/Positive Program, Arbitrary/Monotonic Aggregation) Given a program P, a weight function $\wp$, an aggregation strategy $\mathcal{A}$, and a set of atoms M as input, deciding whether M is an optimal model of P w.r.t. $(\wp, \mathcal{A})$ is* co-NP*-complete. Hardness holds even if P is a positive program, M is a model of P, and $\mathcal{A}$ is a monotonic aggregation strategy.*

PROOF. co-NP-*Membership*. We can verify that M is not an optimal model as follows. Guess $M_1 \subseteq B_P$, and check that: either (1) [M_1 is a model of P] and [$\mathsf{HOF}(\wp, \mathcal{A})(M_1) < \mathsf{HOF}(\wp, \mathcal{A})(M)$], or (2) M is not a model of P. Since both (1) and (2) are polynomial time tasks, this problem is in NP, and as a consequence, Checking is in co-NP.

co-NP-*Hardness*. Let E be a CNF Boolean expression, and P_E the disjunctive positive program defined above. We reduce the problem of deciding whether E is an unsatisfiable Boolean expression to the problem of checking whether the Herbrand base B_{P_E} is an optimal model for P_E w.r.t. to $(\wp_0, count)$, where $\wp_0(q) = 0$

for every atom $q \in B_{P_E}$. Note that the aggregation strategy $count$, which given a set S returns the number of elements in S, is monotonic.

Then, $\mathsf{HOF}(\wp_0, count)(B_{P_E}) = |B_{P_E}|$. On the other hand, for any model M of P_E corresponding to a satisfiable truth assignment to E, $\mathsf{HOF}(\wp_0, count)(M) < |B_{P_E}|$ holds. Thus, B_{P_E} is an optimal model for P_E w.r.t. $(\wp_0, count)$ iff E is not satisfiable. □

Approx

Theorem 5.11 states that the $Approx$ problem is NP-complete when all models are considered. Recall that, in contrast, it is Σ_2^P-complete for stable and minimal models.

Reasoning

As in the case of the *Checking* and *Approx* problems, the complexity of reasoning with "all" models is better than with stable/minimal models.

Theorem 5.12 *(Brave) (General/Positive Program, Arbitrary/Monotonic Aggregation, Reasoning) Given a program P, a weight function $\wp$, an aggregation strategy $\mathcal{A}$, and a ground literal q as input, deciding whether $P \models_b^{All} q$ w.r.t. $(\wp, \mathcal{A})$ (brave reasoning) as well as deciding whether $P \models_c^{All} q$ w.r.t. $(\wp, \mathcal{A})$ (cautious reasoning) are Δ_2^P-complete problems. Hardness holds even if P is positive and $\mathcal{A}$ is monotonic.*

PROOF. Δ_2^P*-Membership, brave reasoning*. We first compute $k = max\_hof(\wp, \mathcal{A}, P)$ (this is done in polynomial time by assumption). After that, by binary search on $[0..k]$, we determine the cost Σ of the optimal models, by a polynomial number of calls to an (NP) oracle deciding $approx(P, \wp, \mathcal{A}, n, \emptyset)$ ($n = k/2$ on the first call; then if the oracle answers "yes", $n = k/4$; otherwise, n is set to $k/2 + k/4$, and so on, according with standard binary search). (Observe that the number of calls to the oracle is logarithmic in k, and, as a consequence, is polynomial in the size of the input.) Finally, a call to the oracle $approx(P, \wp, \mathcal{A}, \Sigma, \{q\})$ verifies that q is true in some optimal model of P.

Δ_2^P*-Hardness, brave reasoning*. We show this by a polynomial time reduction of the following Δ_2^P-complete problem [8, 11]: Given a satisfiable CNF Boolean formula E on $X = \{x_1, \ldots, x_n\}$, decide whether the with respect to $\langle x_1, \ldots, x_n \rangle$ lexicographically minimum $\phi(X)$ satisfying E, which we denote by $\phi_{min}(X)$, fulfills $\phi_{min}(x_n) = true$.

Consider the disjunctive positive program P_E corresponding to E, as defined above. Recall that there is a one-to-one correspondence between the truth assignments to X satisfying E and the models of P_E which do not contain the atom $contr$. Now, consider the *sum* aggregation strategy, and the atomic weight assignment $\wp$ such that: (i) $\wp(x_i) = 2^{n-i}$ $(1 \leq i \leq n)$, (ii) $\wp(x_i') = 0$ $(1 \leq i \leq n)$, and (iii) $\wp(contr) = 1$. Note that $\mathsf{HOF}(\wp, sum)(B_{P_E}) = 2^n$, because of $contr$. Then, B_{P_E} is not an optimal model. Indeed, since E is satisfiable, $\mathsf{HOF}(\wp, sum)(M) \leq 2^n - 1$ for any other model M for P_E.

It is easy to see that the Herbrand objective function $\mathsf{HOF}(\wp, sum)$ induces a total order on the set of models of P_E. In particular, given two models M and M' of P_E, $\mathsf{HOF}(\wp, sum)(M) > \mathsf{HOF}(\wp, sum)(M')$ iff the truth assignment ϕ_M is greater than ϕ_M' in the lexicographical order. In particular, P_E has a unique optimal model M, corresponding to the lexicographically-minimum truth assignment satisfying E (i.e., $\phi_M(X) = \phi_{min}(X)$). Therefore, the lexicographically minimum truth assignment $\phi_{min}(X)$ satisfying E fulfills $\phi_{min}(x_n) =$

$true$ if and only if x_n is true in the optimal model of P_E w.r.t. $(\wp, sum)$ (that is, iff x_n is a brave consequence of P w.r.t. $(\wp, sum)$). Therefore, brave reasoning is Δ_2^P-hard. Moreover, hardness holds even if the program is positive and the aggregation strategy is monotonic, as P_E is positive and sum is monotonic, resp.

Cautious Reasoning. Both membership and hardness for Δ_2^P can be easily derived by the same arguments described in the first part of the proof for brave reasoning. □

6 Algorithms for Computing Optimal Models

In this section, we present algorithms for computing optimal models, considering the families of stable, minimal, and all models.

6.1 Definitions and Notations

We start by introducing some notation. We say that an interpretation I' for a program P *extends* an interpretation I for P if I is consistent and $I \subseteq I'$. A model M for P extends an interpretation I (also, I is extended to M) if there exists an interpretation I' such that I' extends I and $M = I'^{+}$.

$T_P : 2^{B_P \cup \neg.B_P} \rightarrow 2^{B_P}$ denotes the (skeptical version of the) immediate consequence operator. It is defined as:

$$T_P(I) = \{a \in B_P | \quad \exists r \in ground(P)\, s.t.\, a \in H(r),\ H(r) - \{a\} \subseteq \neg.I, and\ B(r) \subseteq I\}.$$

$\Phi_P : 2^{B_P \cup \neg.B_P} \rightarrow 2^{B_P}$ denotes an extension of Fitting's operator [5] to the disjunctive case, used for computing false atoms of P.

$$\Phi_P(I) \quad = \quad \{a \in B_P | \forall r \in ground(P)\ with\ a \in H(r) :\ B(r) \cap \neg.I \neq \emptyset\ or\ (H(r) - \{a\}) \cap I \neq \emptyset\}.$$

Definition 6.1 Let $\mathcal{P}$ be a program and I a set of literals. A *positive possibly-true literal* of $\mathcal{P}$ w.r.t. I is an atom a such that there exists a rule $r \in ground(\mathcal{P})$ for which all the following conditions hold:

1. $a \in H(r)$;
2. $H(r) \cap I = \emptyset$ (that is, the head is not true w.r.t. I);
3. $B(r) \subseteq I$ (that is, the body is true w.r.t. I).

A *negative possibly-true literal* of $\mathcal{P}$ w.r.t. I is a literal $\neg a$ s.t. $a \notin (I^+ \cup I^-)$, and there exists a rule $r \in ground(\mathcal{P})$ for which all the following conditions hold:

1. $\neg a \in B(r)$;
2. $H(r) \cap I = \emptyset$ (that is, the head is not true w.r.t. I);
3. $B^+(r) \subseteq I$ (that is, the positive atoms in the body are true w.r.t. I);
4. $B^-(r) \cap I \neq \emptyset$ (that is, no negative literal in the body is false w.r.t. I).

The set of all possibly-true literals of P w.r.t. I is denoted by $PT_P(I)$. □

Example 6.1 Consider the program $P = \{a \vee b \leftarrow c, d;\ \ e \vee d \leftarrow;\ \ g \vee h \leftarrow c, \neg f\}$ and let $I = \{c, d\}$ be an interpretation for P. Then, we have two positive possibly-true literals of P w.r.t. I, namely, a, and b; and the negative possibly-true literal $\neg f$. □

The following useful property of the set of possibly-true literals can be easily verified. The same property holds for the related notion of possibly-true conjunctions [17].

Proposition 6.1 *Let P be a program and I a consistent interpretation for P. Then, $PT_P(I) = \emptyset$ if and only if I^+ is a model for P.*

6.2 Computing an Optimal Stable Model of a Positive Program with a Monotonic Aggregation Strategy

Algorithm **Optimal_Stable_Model** shows how to compute optimal stable model of a disjunctive positive program P with a monotonic aggregation strategy $\mathcal{A}$.

Algorithm Optimal_Stable_Model
Input: A disjunctive positive program P, a weight function $\wp$,
  a monotonic aggregation strategy $\mathcal{A}$.
Output: An optimal stable model of P w.r.t. $(\wp, \mathcal{A})$.

Procedure $Compute\_Optimal(I : SetOfLiterals;$ **var** $best\_mod : SetOfLiterals)$;
var I': *SetOfLiterals*; L: Atom;
  repeat
    $I' := I$;
    $I := I' \cup T_P(I') \cup \neg.\Phi_P(I')$;
  until $I = I'$;
  if $I \cap \neg.I \neq \emptyset$ **or** $LB_P(I, \wp, \mathcal{A}) > \mathsf{HOF}(\wp, \mathcal{A})(best\_mod)$
    then return
  end_if
  if $PT_P(I) = \emptyset$ (* I^+ is a model *)
    then if $(I^+ \subset best\_mod)$ **or** $\mathsf{HOF}(\wp, \mathcal{A})(I^+) < \mathsf{HOF}(\wp, \mathcal{A})(best\_mod)$
      then $best\_mod := I^+$;
    end_if
    return
  end_if
  $L :=$ *choose*$(PT_P(I), \wp, \mathcal{A}, P, I)$;
  $Compute\_Optimal(I \cup \{L\}, best\_mod)$;
  $Compute\_Optimal(I \cup \{\neg L\}, best\_mod)$;
end_procedure

var *best_model*: *SetOfLiterals*;
begin (* Main *)
  $best\_model := B_P$;
  $Compute\_Optimal(\emptyset, best\_model)$;
  output $best\_model$; (* $best\_model$ is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ *)
end

The function $LB_P(I, \wp, \mathcal{A})$ is a polynomial-time function which returns a lower bound for the set

$$\{\mathsf{HOF}(\wp, \mathcal{A})(M) \mid M \text{ is a model of } P,\ I^+ \subseteq M,\ \text{and } I^- \cap M = \emptyset\}$$

of reals numbers. That is, no model of P extending I can get a Herbrand objective function value better than $LB_P(I, \wp, \mathcal{A})$.

Note that, for any consistent interpretation I, $\mathsf{HOF}(\wp, \mathcal{A})(I^+)$ is a trivial lower bound for P w.r.t. $(I, \wp, \mathcal{A})$, because $\mathcal{A}$ is a monotonic strategy, and every model of P including I, will clearly be a superset of I^+. Lower bounds are used to cut the search space. Consider a call to the procedure $Compute\_Optimal(I : SetOfLiterals;\ \mathbf{var}\ best\_mod : SetOfLiterals)$, where I is a set of literals, and *best_mod* is a model for the program, representing the model with the lowest Herbrand objective function we have computed so far. Intuitively, we are looking for some optimal stable model M "extending" the set of literals I. If P has a lower bound of n w.r.t. $(I, \wp, \mathcal{A})$, and $n > \mathsf{HOF}(\wp, \mathcal{A})(best\_mod)$, it clearly does not make sense to continue this way, because we cannot compute a model better than *best_mod*, by extending I. Thus, we can terminate the current call to the procedure, and go back to explore other possibilities.

The actual algorithm to be used for the computation of a lower bound for some set of literals I w.r.t. some $(\wp, \mathcal{A})$ should be chosen depending on the particular aggregation strategy $\mathcal{A}$, on the weight function $\wp$, and, possibly, on the particular class of programs P belongs to.

The function *choose*$(PT_P(I), \wp, \mathcal{A}, P, I)$ selects a possibly-true literal from $PT_P(I)$. We can use different strategies for this selection, which best fit different combinations of weight functions and aggregation strategies. However, we consider only choices that can be done in polynomial-time. If $\mathcal{A}$ is monotonic, the simplest strategy is a greedy choice: possibly true atoms which are assigned the lowest weights will be chosen first. However, more sophisticated methods can be easily designed. For instance, looking at the program P, we can choose atoms whose immediate logical consequences give the least increment, and so on.

The following example shows in detail how the algorithm works.

Example 6.2 Let P be the following disjunctive positive program:

$$\begin{array}{rcl} a \vee b & \leftarrow & \\ d & \leftarrow & \\ c \vee d & \leftarrow & a \\ e \vee f & \leftarrow & b \end{array}$$

We apply the algorithm **Optimal_Stable_Model** to compute a model belonging to $\underline{\mathtt{Opt}}(P, \mathsf{ST}(P), \wp, \mathtt{sum})$, i.e., an optimal stable model of P with the $\mathtt{sum}$ aggregation strategy, and a weight function $\wp$ such that $\wp(a) = 2$, $\wp(b) = 3$, $\wp(c) = 1$, $\wp(d) = 3$, $\wp(e) = 2$, and $\wp(f) = 2$. We use the lower-bound function LB_P that, applied to an interpretation I, just returns the sum of the weights associated with the atoms in I^+.

The procedure $Compute\_Optimal$ is first called with the empty interpretation, and $best\_mod$ is initialized with the Herbrand base of P. The value of *best_mod* is $\mathsf{HOF}(\wp, \mathtt{sum})(best\_mod) = 13$. After the execution of the **repeat** loop, we get the interpretation $I_1 = \{d, \neg c\}$, because $T_P(\emptyset) = \{d\}$, $\Phi_P(\{d\}) = \{c\}$, and no further atoms can be obtained by using these deterministic operators. The evaluation of the lower-bound function gives $LB_P(I_1, \wp, \mathtt{sum}) = \mathsf{HOF}(\wp, \mathtt{sum})(I_1^+) = 3$. This value is less than $\mathsf{HOF}(\wp, \mathtt{sum})(best\_mod)$, and hence the algorithm continues and computes the set of possibly-true atoms w.r.t. I_1. We get $PT_P(I_1) = \{a, b\}$. Since this set is not empty, we select from it a possibly-true atom, by using the function *choose*. Assume we choose the atom a, and let $I_2 = I_1 \cup \{a\} = \{a, d, \neg c\}$.

Then, the procedure $Compute\_Optimal$ is recursively called with the parameters I_2 and $best\_mod$, to compute possible optimal stable models starting from the interpretation I_2. The **repeat** loop ends with the new interpretation $I_3 = \{a, d, \neg b, \neg c, \neg e, \neg f\}$. This interpretation is consistent and the lower bound $LB_P(I_3, \wp, \mathtt{sum}) = 5$ is less than the value 13 of *best_mod*. Moreover, $PT_P(I_3) = \emptyset$. Indeed, the set $I_3^+ = \{a, d\}$ is a model of P. Since I_3^+ is a subset of $best\_mod = B_P$, it becomes the new best model, that is, we set $best\_mod = I_3^+$.

At this point, we come back to the previous execution of $Compute\_Optimal$, and try to compute optimal models assuming the previously selected atom be false. Therefore, we call $Compute\_Optimal(I_4, best\_mod)$, where $I_4 = I_1 \cup \{\neg a\} = \{d, \neg a, \neg c\}$. The **repeat** loop gives the interpretation $I_5 = \{b, d, \neg a, \neg c\}$, because b is now deterministically derivable from the first rule of P. We compute the lower bound for this interpretation: $LB_P(I_5, \wp, \mathtt{sum}) = 6$. This value is compared with the Herbrand objective function evaluated on the best model. Since $\mathsf{HOF}(\wp, \mathtt{sum})(best\_mod) = 5$, this execution of $Compute\_Optimal$ is stopped, because no stable model better than $best\_mod$ can be found starting from I_5.

Now, the first call of $Compute\_Optimal$ has been completed, and thus the algorithm stops and returns the optimal stable model $best\_mod = \{a, d\}$.

For completeness, note that I_5^+ is not a model for P, and in fact there were some possibly-true atoms w.r.t. I_5, as $PT_P(I_5) = \{e, f\}$. Indeed, P has other two stable models, namely, $\{b, d, e\}$ and $\{b, d, f\}$. However, these models are not optimal because their Herbrand objective function is equal to 8, while the optimal value is 5. Therefore, thanks to the lower-bound function, the algorithm is able to cut the search space, avoiding the generation of useless (i.e., not optimal) stable models.

Theorem 6.1 *Given a disjunctive positive program P, a weight function $\wp$, and a monotonic aggregation strategy $\mathcal{A}$, the algorithm* **Optimal_Stable_Model** *outputs an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$.*

PROOF. At each time of the computation of **Optimal_Stable_Model**, the variable $best\_model$ contains a model for P. Indeed, it is initialized with the Herbrand base B_P, which is clearly a model for P, and during the computation of the algorithm it can be replaced only by another model for P having a better Herbrand objective function (short: HOF) value. It is easy to verify that the algorithm terminates after a finite number of steps, because at each recursive call of $Compute\_Optimal$ we add to the (partial) interpretation I a new literal, and the Herbrand base of P is finite, as no function symbol occurs in P. Then, the algorithm returns a set of atoms, say M. We claim that M is a model for P. This clearly holds if $M = B_P$. Otherwise, by construction of the algorithm, M is the set of positive literals I^+ of an interpretation I such that $PT_P(I) = \emptyset$. In this case, the claim follows from Proposition 6.1.

Now, assume by contradiction that the algorithm **Optimal_Stable_Model** does not output an optimal stable model for P, and let M' be an optimal stable model for P. There are two possibilities:

1. the HOF value of M is strictly greater than M', i.e., $\mathsf{HOF}(\wp, \mathcal{A})(M) > \mathsf{HOF}(\wp, \mathcal{A})(M')$; or
2. the models M and M' have the same HOF value, but M is not stable and hence not minimal, as P is positive.

If the second condition holds, we may assume without loss of generality that $M' \subset M$, because in this case every minimal model included in M is an optimal stable model for P. Then it is easy to see that, for each call to $Compute\_Optimal(I, best\_mod)$ during the computation of **Optimal_Stable_Model**, either $\mathsf{HOF}(\wp, \mathcal{A})(best\_mod) > \mathsf{HOF}(\wp, \mathcal{A})(M')$ or $M' \subset best\_mod$ holds.

Let I be the largest interpretation for P that can be extended to M' and that has been used as a parameter for a call to the procedure $Compute\_Optimal$, during the execution of **Optimal_Stable_Model**. Note that such an interpretation must exist, because the algorithm starts with the empty interpretation, which can be extended to any model. Now, consider the evaluation of $Compute\_Optimal(I, best\_mod)$. We first evaluate the **repeat** loop, which extends I to a new interpretation, say $\bar{I}$. It is easy to verify that, for every model $\tilde{M}$ for P that extends I, $\tilde{M}$ extends $\bar{I}$, as well. In fact, we just add to I literals that are deterministic consequences of P given the interpretation I. It follows that M' extends $\bar{I}$. This in turn entails that $\bar{I}$ is a consistent interpretation. Moreover, $LB_P(\bar{I}, \wp, \mathcal{A})$ cannot be greater than $\mathsf{HOF}(\wp, \mathcal{A})(best\_mod)$ because we know that M' is an optimal stable model and extends $\bar{I}$.

Thus, the computation continues and we evaluate the set of possibly-true literals $PT_P(\bar{I})$. Assume that $PT_P(\bar{I}) = \emptyset$. ¿From Proposition 6.1, $\bar{I}^+$ is a model for P. Since $\bar{I}$ can be extended to the model M' then $PT_P(\bar{I}) = \emptyset$ entails $\bar{I}^+ = M'$. Thus, from the above relationships between M' and $best\_mod$, it follows that M' (i.e., $\bar{I}^+$) should replace $best\_mod$. However, if this happens, there is no way to replace M' by M during the algorithm. This is a contradiction, as we assumed M is the output of the computation of **Optimal_Stable_Model** on P.

Thus, assume $PT_P(\bar{I}) \neq \emptyset$ holds. Let L be the possibly-true literal selected by *choose*$(PT_P(\bar{I}), \wp, \mathcal{A}, P, \bar{I})$. We then call both $Compute\_Optimal(\bar{I} \cup \{L\}, best\_mod)$ and $Compute\_Optimal(\bar{I} \cup \{\neg L\}, best\_mod)$. However, if $L \in M'$, then $\bar{I} \cup \{L\}$ can be extended to M'; otherwise, $\bar{I} \cup \{\neg L\}$ can be extended to M'. In either case, our assumption on I is contradicted. $\Box$

Since even computing just a minimal model of a disjunctive positive program is an NP-hard task (actually, $P^{\mathrm{NP}[O(\log n)]}$-hard) [1], unless the polynomial hierarchy collapses, there exists no polynomial time algorithm which computes an optimal stable model for a disjunctive positive program. However, an optimal stable model can be computed by using polynomial space and single exponential time. Moreover, the algorithm runs efficiently on the tractable class of normal positive programs.

Remark. Given a positive disjunctive program P, the algorithm **Optimal_Stable_Model** outputs a model which is also optimal w.r.t. minimal model semantics. To see this, observe that for positive programs, stable and minimal models coincide. Now, consider the family of all models. For any optimal model M, there exists a minimal model $M' \subseteq M$ which is also an optimal model. Indeed, for any monotonic aggregation strategy $\mathcal{A}$ and any weight function $\wp$, $\mathsf{HOF}(\wp, \mathcal{A})(M') = \mathsf{HOF}(\wp, \mathcal{A})(M)$ must hold because M' is a subset of M. Thus, every optimal minimal model is also an optimal model w.r.t. the family of all models, hence algorithm **Optimal_Stable_Model** outputs an optimal model w.r.t. to the family of all models, as well.

6.3 Computing an Optimal Stable Model: General Case

Algorithm *Optimal_Model_General*, below computes an optimal stable model of a general disjunctive program with any aggregation strategy.

The main difference with the previous case, where the program was restricted to be positive and the aggregation strategy to be monotonic, is that we now have to explicitly check that the model at hand is stable. This is accomplished by the function *is_Stable*. Stable model checking can be effectively done by looking for some unfounded set possibly included in the model, as suggested in [17], and recently improved in [10]. We assume *is_Stable* is implemented as described in [10].

Algorithm Optimal_Model_General

Input: A disjunctive program P, a weight function $\wp$, an aggregation strategy $\mathcal{A}$.
Output: An optimal stable model of P w.r.t. $(\wp, \mathcal{A})$.

```
Procedure Compute_Optimal_G(I : SetOfLiterals;
                            var best_mod : SetOfLiterals; var best_val : Real);
var I': SetOfLiterals; L: Literal;
    repeat
        I' := I;
        I := I' ∪ T_P(I') ∪ ¬.Φ_P(I');
    until I = I';
    if I ∩ ¬.I ≠ ∅  or  LB_P(I, ℘, A) > best_val
        then return
    end_if
    if PT_P(I) = ∅ (* I+ is a model *)
        then if HOF(℘, A)(I+) < best_val  and  is_Stable(I+)
            then     best_mod := I+;
                     best_val := HOF(℘, A)(I+);
        end_if
        return
    end_if
    L := choose(PT_P(I), ℘, A, P, I);
    Compute_Optimal_G(I ∪ {L}, best_mod, best_val);
    Compute_Optimal_G(I ∪ {¬L}, best_mod, best_val);
end_procedure

var
    best_model: SetOfLiterals;
    best_value: Real;
begin   (* Main *)
    best_value := ∞;
    Compute_Optimal_G(∅, best_model, best_value);
    if best_value < ∞
        then output best_model;
        else output "P does not have stable models"
    end_if
end
```

The implementation of $LB_P(I, \wp, \mathcal{A})$ is an important issue. While the search space can be effectively reduced in the case of monotonic aggregations, there are general aggregation strategies where no "good" lower bounds can be computed in reasonable time. In this case, one can just return the trivial lower bound $-\infty$.

Moreover, for positive programs and monotonic aggregation strategies, we can always start with a model for the given program, whose HOF value represents an upper bound to the optimal HOF value. Indeed, the Herbrand base B_P is a model for any program P, and there always exists a model $M \subseteq B_P$ which is a stable model for P and has a HOF value not greater than B_P. In the general case – with negation and general aggregation strategies –, this initialization would be meaningless, because logic programs with negation may have no stable models at all, and because larger models may have better HOF values than smaller models. Thus, we have added the new parameter *best_value* to the recursive procedure. *best_value* is initialized with the special value ∞. By inspecting the value of this variable at the end of the algorithm, we can determine whether P has stable models or not. In the former case, P has also optimal stable models, and the variable *best_model* holds an optimal stable model for P.

The correctness of *Optimal_Model_General* is stated by the following theorem, whose proof is very similar to the proof of Theorem 6.1, and will be thus omitted.

Theorem 6.2 *Given a disjunctive program P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$, the algorithm* Optimal_Stable_Model_General *outputs an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$.*

Even in the general case, an optimal stable model can be computed by using polynomial space and single exponential time. Moreover, the algorithm runs efficiently on the tractable class of normal positive programs.

6.4 Computing an Optimal Minimal Model

In this section, we describe how to compute optimal minimal models. Let P be a disjunctive general program and let $PV(P)$ be the positive version of P, defined in Section 5.4.

Recall that the set of models of P and $PV(P)$ are identical, and thus the set of minimal models are identical, too. Moreover, for any positive program P', $\mathsf{ST}(P') = \mathsf{MM}(P')$ holds. It follows that the algorithms developed for computing optimal stable models can be used for computing optimal minimal models, too.

Proposition 6.2 *Let P be a disjunctive general program. Given the positive version $PV(P)$ of P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$, the algorithm* Optimal_Stable_Model_General *outputs an optimal minimal model of P w.r.t. $(\wp, \mathcal{A})$.*

Moreover, if $\mathcal{A}$ is monotonic, then, given $PV(P)$, $\wp$, and $\mathcal{A}$, the algorithm Optimal_Stable_Model *outputs an optimal minimal model of P w.r.t. $(\wp, \mathcal{A})$.*

Thus, given a general program P, we can first compute its positive version and then compute an optimal minimal model for P by using either *Optimal_Stable_Model* or *Optimal_Stable_Model_General*, depending on the aggregation strategy, i.e., whether it is monotonic or not. Since $PV(P)$ can be computed from P in linear time, the overall cost of computing an optimal minimal model (by using this procedure) is the same as the complexity of these algorithms, stated by Theorems 8.1 and 6.2, respectively.

6.5 Computing an Optimal Model

Computing an optimal model with respect to the family of all models is a quite different task, because every model is a candidate to be the best model. Thus, algorithm designed for computing just minimal and stable models are no longer useful, in general. However, if the aggregation strategy is monotonic, for each optimal model M, there is an optimal minimal model $M' \subseteq M$ having the same HOF value, and hence optimal as well. Therefore, in this case, we can safely use the algorithm *Optimal_Stable_Model* applied on the positive version of the program.

Proposition 6.3 *Let P be a disjunctive general program. Given the positive version $PV(P)$ of P, a weight function $\wp$, and a monotonic aggregation strategy $\mathcal{A}$, the algorithm* Optimal_Stable_Model *outputs an optimal model of P w.r.t. $(\wp, \mathcal{A})$.*

However, if we consider aggregation strategies that are not monotonic, by using the algorithms from previous sections, we can miss optimal models. We next describe a new algorithm for the general case.

First, we define a new operator for deriving literals that should be false in every model extending a given interpretation. Let P be a program and I an interpretation for P. The operator $\Phi_P : 2^{B_P \cup \neg.B_P} \rightarrow 2^{B_P \cup \neg.B_P}$ is defined as follows:

$$\Phi_P^a(I) = \{\ell \mid \exists r \in ground(P)\ with\ \ell \in B(r) : \ (B(r) - \ell) \subseteq I \text{ and } \neg.H(r) \subseteq I\}.$$

That is, if the head of r is false and all the literals in its body but ℓ are true with respect to I, then ℓ must be false in every model extending the interpretation I. Note that, if $\ell \in \Phi_P^a(I)$ is a negative literal, say $\neg p$, then we derive that p is true in every model (if any) extending I.

Algorithm *Optimal_Model* for the computation of an optimal model of a disjunctive program P with a general aggregation strategy $\mathcal{A}$. For an interpretation I, we denote by I^u the set of atoms that are undefined with respect to I, i.e., the set $B_P - (I \cap \neg.I)$.

The computation starts with the empty interpretation. At each call of the recursive procedure *Compute_Optimal_G*, we try to extend a partial interpretation I to a model of P. In particular, the **repeat** loop extends I by adding the literals (either positive or negative) that are immediate consequences of the program P starting from the interpretation I. Let I' this new interpretation. If I' is consistent and, according to the lower bound estimation $LB_P(I', \wp, \mathcal{A})$, can lead to some optimal model of P, then the procedure continues; otherwise, it ends and we backtrack to a previous call of *Compute_Optimal_G*. Then, if I' is consistent and complete, it is in fact a model for P (see the proof below). In this case, we compare it with the previous best model and possibly replace this model by I', if the HOF value of I' is better. Otherwise, i.e., if I' is consistent but not complete, we select an undefined literal L with the function *choose*, and then try to extend recursively first $I' \cup \{L\}$ and then $I' \cup \{\neg L\}$.

Algorithm Optimal_Model
Input: A disjunctive program P, a weight function $\wp$, an aggregation strategy $\mathcal{A}$.
Output: An optimal model of P w.r.t. $(\wp, \mathcal{A})$.

Procedure $Compute\_Optimal\_G(I : SetOfLiterals;$
    var $best\_mod : SetOfLiterals;$ **var** $best\_val : Real);$
var I': *SetOfLiterals*; L: Literal;
  repeat
    $I' := I;$
    $I := I' \cup T_P(I') \cup \neg.\Phi_P^a(I');$
  until $I = I'$;
  if $I \cap \neg.I \neq \emptyset$ **or** $LB_P(I, \wp, \mathcal{A}) > best\_val$
    then return
  end_if
  if $I^u = \emptyset$ (* I^+ is a model *)
    then if $\mathsf{HOF}(\wp, \mathcal{A})(I^+) < best\_val$
      then $best\_mod := I^+;$
        $best\_val := \mathsf{HOF}(\wp, \mathcal{A})(I^+);$
    end_if
    return
  end_if
  $L :=$ *choose*$(I^u, \wp, \mathcal{A}, P, I);$
  $Compute\_Optimal\_G(I \cup \{L\}, best\_mod, best\_val);$

$Compute\_Optimal\_G(I \cup \{\neg L\}, best\_mod, best\_val)$;
end_procedure

var
best_model: *SetOfLiterals*;
best_value: Real;
begin (* Main *)
$best\_value := \infty$;
$Compute\_Optimal\_G(\emptyset, best\_model, best\_value)$;
output $best\_model$;
end

We next briefly discuss the differences of the algorithm *Optimal_Model* with the previous algorithms for computing optimal minimal and optimal stable models. For the All-models semantics, we have to consider every model for the program. Since the possibly-true atoms have been defined for generating only stable models, in this case the function $choose$ may select any undefined atom in I^u, rather than restricting the selection just to $PT_P(I)$; otherwise, we can miss some (possibly optimal) model. Literals that should be false in every model extending I are computed using the operator Φ_P^a, rather than Φ_P, which declare false any atom no longer supported by the rules of P. The latter would not be safe for the family of all models.

Theorem 6.3 *Given a disjunctive program P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$, the algorithm* Optimal_Model_G *outputs an optimal model of P w.r.t. $(\wp, \mathcal{A})$.*

PROOF. Note that we consider logic programs without integrity constraints.[6] It follows that P has always models and hence optimal models. In fact, algorithm *Optimal_Model_G* always outputs a set of atoms, say M. We next prove that M is an optimal model for P.

First observe that M is model. Indeed, M is the set of positive literals I^+ of a total consistent interpretation I computed by the **repeat** loop. Assume by contradiction that M is not a model. Then, there is a rule $r \in P$ such that $B(r) \subseteq I$ and $H(r) \cap I = \emptyset$ or, equivalently, $\neg.H(r) \subseteq I$, because I is total. This entails that, for any literal $\ell \in B(r)$, $\ell \in \Phi_P^a(I)$. Since at each step of the loop we add $\neg.\Phi_P^a(I)$ to the current interpretation and we assumed $\ell \in B(r)$ and $B(r) \subseteq I$, we get that both ℓ and $\neg\ell$ belong to I, a contradiction.

Now, assume by contradiction that the algorithm *Optimal_Model_G* does not output an optimal model for P, and let M' be an optimal model for P. This means that neither M', nor any other optimal model for P, is generated during the computation of *Optimal_Model_G*, otherwise the variable $best\_mod$ should hold such a model and the algorithm would eventually output it. Therefore, $best\_val > \mathsf{HOF}(\wp, \mathcal{A})(M')$ holds at each step of the computation. Let I be the largest interpretation for P that can be extended to M' and that has been used as a parameter for a call to the procedure $Compute\_Optimal$, during the execution of *Optimal_Model_G*. Note that such an interpretation must exist, because the algorithm starts with the empty interpretation, which can be extended to any model. Let us now consider the evaluation of the function $Compute\_Optimal(I, best\_mod, best\_val)$. We first evaluate the **repeat** loop, which extends I to a new interpretation, say $\bar{I}$. It is easy to verify that, for every model $\tilde{M}$ for P that extends I, $\tilde{M}$ extends $\bar{I}$, as well. It follows that M' extends $\bar{I}$. This in turn entails that $\bar{I}$ is a consistent interpretation. Moreover, $LB_P(\bar{I}, \wp, \mathcal{A})$ cannot be greater than $best\_val$ because we know that M' is an optimal model and extends $\bar{I}$. Note that

[6] Recall that we used integrity constraints in our examples as a shorthand for a normal rule that behaves as a constraint under the stable model semantics, as described in Section 3.1.

$\bar{I}^u \neq \emptyset$, otherwise from our assumption on I and hence on $\bar{I}$, $M' = \bar{I}^+$ would immediately follow. Then, we select a literal L by using the function $choose(\bar{I}^u, \wp, \mathcal{A}, P, \bar{I})$, and call both $Compute\_Optimal(\bar{I} \cup \{L\}, best\_mod, best\_val)$, and $Compute\_Optimal(\bar{I} \cup \{\neg L\}, best\_mod, best\_val)$. However, if $L \in M'$, then $\bar{I} \cup \{L\}$ can be extended to M'; otherwise, $\bar{I} \cup \{\neg L\}$ can be extended to M'. In either case, our assumption on I is contradicted. □

Even in the case of all models, an optimal model can be computed by using polynomial space and single exponential time. The proof is very similar to the proof of Theorem 8.1 and thus will be omitted.

Theorem 6.4 *Given a disjunctive program P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$,* Optimal_Model *runs in polynomial space and in single exponential time, provided that $\wp$ and $\mathcal{A}$ are polynomial time computable.*

7 Related Work

We split related work into two categories - work on logic numeric values are assigned to models. The first is in logic programming with uncertainty, and work on "weighted logic programs."

7.1 Relationship with Logic Programs for Uncertainty

Logic programs to handle uncertainty were first introduced by Shapiro[26] and later studied by van Emden [32]. In both cases, the authors wrote normal, positive rules and associated a number between 0 and 1 (inclusive) with that rule. The model theory of such programs was simple. Interpretations assigned real numbers in the $[0, 1]$ interval to ground atoms. Interpretation I assigns $min\{I(A_1, \ldots, I(A_n)\}$ to the conjunction $A_1 \,\&\, \ldots \,\&\, A_n$. A ground rule $A \leftarrow A_1 \,\&\, \ldots \,\&\, A_n$ with associated real number v is satisfied by I iff $I(A) \geq v \times I(A_1 \,\&\, \ldots \,\&\, A_n)$. Later, this work was extended by Kifer and Subrahmanian [9] who introduced generalized annotated programs (GAPs for short). GAPs generalized such rules to use an arbitrary complete lattice of truth values, and furthermore, instead of associating numbers with rules as a whole, GAPs provided a syntax to annotate atoms with lattice values as well as expressions that would evaluate to lattice values. The resulting syntax allowed both Shapiro's and van Emden's framework to be special cases on GAPs. Furthermore, numerous other interesting kinds of problems (such as temporal reasoning problems) could be expressed in GAPs. Later, [20] adapted the GAP framework to handle purely probabilistic data. Lakshmanan and his colleagues [14, 13, 15, 27] developed an elegant parametric framework to represent varied probabilistic strategies in logic programs and developed important query optimization results.

However, all these approaches are not directly related to the problem of associating weights with models. We note that, in [20, 14], a probabilistic interpretation consists of a set of (ordinary) Herbrand interpretations together with a probability distribution over them. Thus, if we have a very simple probabilistic logic program with 2 ground atoms a, b occurring in it (and no others), we would have four Herbrand models — $\emptyset, \{a\}, \{b\}, \{a, b\}$. A probabilistic interpretation may associate a probability with each of them such as $0.2, 0.3, 0.1, 0.4$, respectively. Such an association of probabilities may be viewed as associating a "cost like" value with the models.

However, there are many differences between this paper and such approaches. First and foremost, a user of a logic program cannot specify his own costs — they are implicitly embedded already in the probabilistic logic program and cannot be changed. Second, the probabilistic LP approaches do not have any notion of "optimality" and hence, none of the complexity results of this paper or the algorithms to compute optimal (stable, minimal, all) models are described there.

7.2 Relationship with Weighted Logic Programs

Marek and Trusczynski [18] have introduced a notion of "weighted logic programs" and Niemela's group has separately introduced weighted logic programs [21, 22, 30]. These two frameworks are quite different, even though they have the same name, so we consider them separately.

Marek and Trusczynski [18] have rules that are *identical* in syntax to the rules above of Shapiro and van Emden with one difference — the numbers associated with rules do not have to lie in the $[0, 1]$ interval. A paraphrased version of the informal reading of the rule given in [18, p.7] $A \leftarrow A_1 \,\&\, \ldots \,\&\, A_n$ with number v is: If $A_1, \ldots, A_n$ are true and the amount of available resources equals or exceeds v, then we can derive a, and to do so, we must decrease the available resources by v. One can now give a straightforward fixpoint semantics for a weighted logic program (which is a set of such rules). For instance, consider the weighted logic program:

$$\begin{array}{lll} a & \leftarrow^3 & \\ b & \leftarrow^4 & \\ c & \leftarrow^2 & a \,\&\, b \\ d & \leftarrow^5 & a. \end{array}$$

One semantics for this program would allow us to infer, in one step, that a and b can be established at costs 3 and 4, respectively, and in the second step, that c and d can be established at costs 9 and 8, respectively. An alternative semantics is also proposed by the authors. The authors note that their results are similar to those of van Emden [32]. However, there are many differences between our work and theirs.

Niemela's group has also introduced a notion of a weighted logic program. In their framework, they have rules of the form $C_0 \leftarrow C_1, \ldots, C_n$ where each C_i has the form

$$L \leq \{\ell_1 = w_1, \ldots, \ell_m = w_m\} \leq U.$$

The ℓ_i's are literals (for the sake of brevity, we will assume they are ground though this is not required by [21, 22, 30]). An Herbrand interpretation I satisfies the above (ground) rule if:

- Each literal $\ell_i \in I$ (we say ℓ_i is true in this case), and
- $L \leq \sum_{\ell_i \in I} w_i \leq U$.

Satisfaction of non-ground rules is defined in the usual way.

There are many differences between our work and the above two pieces of work. First, it is the logic programmer who encodes the numbers. The end user has no say in the matter — this causes problems in examples like the cooking example, where different cooks may have different costs associated with planning a dinner. Second, there is no notion of an optimal cost — Marek and Trusczynski [18] have an unique least model that captures the least cost of each atom. However, there is no notion of associating costs with models. Niemela et. al [21] only ensure that the above constraints (expressed via their rules) are satisfied — there is no notion of a model being optimal. Third, neither approach deals with disjunctive logic programs. Niemela's work deals with normal programs, while the Marek and Truszczynski deals only with positive, normal programs. Hence, the complexity results of this paper and the algorithms to compute optimal (stable, minimal, all) models described here are novel.

An important contribution that Niemela's group makes that we do not is that they have implemented their framework and they have developed product configuration applications. These are important contributions that complement ours.

8 Conclusions

Logic programs and disjunctive logic programs form a powerful reasoning paradigm that can be used to elegantly and declaratively encode a variety of problems. Given a (possibly disjunctive) logic program P, there has been an immense amount of work on characterizing the declarative semantics of P. Many of these semantics (such as the stable model semantics [6], the minimal model semantics [19, 2], and the "all-models" semantics) identify a class of models of the program as being epistemically acceptable.

In this paper, we take the point of view that a user of a logic program (which is presumably written by a logic programming professional) may wish to add criteria that she cares about, in the selection of a model from the family of models identified by the semantics. We argue that once the logic programmer encodes an application using some semantics, the user should be able to use objective functions to specify which of the models conforming to the selected semantics should be picked. We have provided a formal definition of optimal models and illustrated their utility through a simple "cooking" example, as well as a combinatorial auction example. Subsequently, we have conducted a complexity-theoretic investigation of various important problems relevant to this optimal models notion. Specifically, we have developed results on the complexity of checking whether a model is optimal (w.r.t stable model semantics, minimal model semantics, and all-models semantics), on determining whether a ground atom is true in all or one of the optimal models w.r.t. one of these aforementioned semantics, and on checking whether there is a model w.r.t one of these semantics such that it contains certain ground atoms and is guaranteed to have a cost below a certain amount. We have also developed an exhaustive set of algorithms to compute optimal models w.r.t stable, minimal and all model semantics. Our results apply to disjunctive logic programs with negation, not just to normal logic programs.

The following questions still need to be studied. First, we need to develop efficient implementations of these algorithms, and conduct experiments in order to determine how much they scale to large programs and data sets. Second, we need to develop efficient methods to answer "brave" and "cautious" queries which ask whether a query is true in some/all optimal models. Third, we need to develop methods to optimize such queries. Preliminary starting points for these algorithms are already contained in the proofs of the complexity results, but extending them to algorithms that scale remains a challenge.

References

[1] Marco Cadoli. On the complexity of model finding for nonmonotonic propositional logics. In M. Venturini Zilli A. Marchetti Spaccamela, P. Mentrasti, editors. In *Proceedings of the 4th Italian Conference on Theoretical Computer Science*, page 125139. World Scientific, Singapore, 1992.

[2] J. Dix, F. Stolzenburg. Computation of Non-Ground Disjunctive Well-Founded Semantics with Constraint Logic Programming. Proceedings NMELP 1996, pages 202-224.

[3] T. Eiter and G. Gottlob. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[4] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[5] M. C. Fitting. (1988) *Logic Programming on a Topological Bilattice*, Fundamenta Informatica, 11, pps 209–218.

[6] M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[7] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[8] Jim Kadin. $\mathrm{P}^{(\mathrm{NP}[O(\log n)])}$ and sparse turing-complete sets for NP. *Journal of Computer and System Sciences*, 39(3):282–298, 1989.

[9] M. Kifer and V. S. Subrahmanian. (1992) Theory of Generalized Annotated Logic Programming and its Applications, *Journal of Logic Programming*, 12(4), pages 335–368, 1992.

[10] Christoph Koch and Nicola Leone. Stable model checking made easy. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 70–75, Stockholm, Sweden, August 1999. Morgan Kaufmann Publishers.

[11] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[12] Mark W. Krentel. Generalizations of opt p to the polynomial hierarchy. *Theoretical Computer Science*, 97(2):183–198, 1992.

[13] V.S. Lakshmanan and F. Sadri. (1994) *Modeling Uncertainty in Deductive Databases*, Proc. Int. Conf. on Database Expert Systems and Applications, (DEXA'94), September 7-9, 1994, Athens, Greece, Lecture Notes in Computer Science, Vol. 856, Springer (1994), pp. 724-733.

[14] V.S. Lakshmanan and F. Sadri. (1994) *Probabilistic Deductive Databases*, Proc. Int. Logic Programming Symp., (ILPS'94), November 1994, Ithaca, NY, MIT Press.

[15] V.S. Lakshmanan and N. Shiri. (1997) *A Parametric Approach with Deductive Databases with Uncertainty*, accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[16] N. Leone, F. Scarcello and V.S. Subrahmanian. (2001). Optimal Models of Disjunctive Logic Programs: Semantics, Complexity, and Computation. Univ. of Maryland Tech. Report CS-TR-4298. `www.cs.umd.edu/Library/TRs/CS-TR-4298.ps.Z`.

[17] Nicola Leone, Pasquale Rullo, and Francesco Scarcello. Disjunctive stable models: Unfounded sets, fixpoint semantics and computation. *Information and Computation*, 135(2):69–112, June 1997.

[18] V.W. Marek and M. Truszczynski. (1999) *Logic Programming with Costs*, unpublished manuscript, 1999.

[19] Jack Minker. On Indefinite Data Bases and the Closed World Assumption. In D.W. Loveland, editor, *Proceedings* 6^{th} *Conference on Automated Deduction (CADE '82)*, number 138 in Lecture Notes in Computer Science, pages 292–308, New York, 1982. Springer.

[20] R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, INFORMATION AND COMPUTATION, 101, 2, pps 150–201, 1993.

[21] I. Niemela, P. Simons and T. Soininen. *Stable Model Semantics of Weight Constraint Rules*, Proc. 5th International Conference on Logic Programming and Nonmonotonic Reasoning, El Paso, Texas, Dec. 1999.

[22] E. Niemela and P. Simons. *Extending the Smodels System with Cardinality and Weight Constraints*, in "Logic-based Artificial Intelligence" (ed. J. Minker), pps 491–521, Kluwer, 2000.

[23] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[24] Domenico Saccà and Carlo Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proceedings of the Ninth Symposium on Principles of Database Systems (PODS '90)*, pages 205–217, 1990.

[25] T. Sandholm. *Approaches to Winner Determination in Combinatorial Auctions*, Decision Support Systems, 28, 1-2, pps 165–176.

[26] E. Shapiro. (1983) *Logic Programs with Uncertainties: A Tool for Implementing Expert Systems*, Proc. IJCAI '83, pps 529–532, William Kauffman.

[27] N. Shiri. (1997) *On a Generalized Theory of Deductive Databases*, Ph.D. Dissertation, Concordia University, Montreal, Canada, August 1997.

[28] L. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[29] L. Stockmeyer and A. Meyer. Word problems requiring exponential time. In *Proceedings of the 5th ACM Symposium on Theory of Computing (STOC '73)*, pages 1–9. ACM Press, 1973.

[30] T. Soininen, I. Niemela, J. Tiihonen and R. Sulonen. *Representing Configuration Knowledge with Weight Constraint Rules*, Proc. AAAI Spring Symp. on Answer Set Programming: Towards Efficient and Scalable Knowledge, March 2001.

[31] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th ACM Symposium on Theory of Computing (STOC'82)*, pages 137–146. ACM, 1982.

[32] M.H. van Emden. (1986) *Quantitative Deduction and its Fixpoint Theory*, Journal of Logic Programming, 4, 1, pps 37-53.

[33] Klaus W. Wagner. Bounded query classes. *SIAM Journal of Computing*, 19(5):833–846, 1990.

Appendix A.1

In this appendix, we show three possible extended database relations for the predicates *dish*, *dislikes*, and *guest*, respectively, in the cooking program of Example 3.1.

Name	Type	Cost	Time
caprese	appetizer	0.55	5
samosa	appetizer	0.75	25
spaghetti_carbonara	entree	2.75	15
lasagna	entree	3.25	25
malai_kofta	entree	2.25	40
matar_paneer	entree	2.40	15
masala_dosa	entree	3.15	30
idli	entree	0.75	30
tiramisu	dessert	3.00	30
rasgulla	dessert	2.00	30

Person	Name of Dish
nicola	malai_kofta
vs	masala_dosa
francesco	masala_dosa
gb	caprese
simona	tiramisu
simona	idli
peppe	rasgulla
peppe	lasagna
tina	matar_paneer
tina	samosa

Guest
nicola
vs
francesco

Appendix A.2

The logic program in Example 3.1 has 16 stable models altogether, depending on what is chosen as an appetizer (2 choices), what is chosen as an entree (4 choices) and what is chosen for dessert (2 choices). Note that the choices are restricted according to the given group of guests; in our example, Francesco, Nicola, and VS. We list below, for all these stable models, just the predicate $dinner$, which characterizes each menu:

$$
\begin{aligned}
M_1 &= \{dinner(caprese, lasagna, tiramisu), \ldots\},\\
M_2 &= \{dinner(samosa, matar\_paneer, rasgulla), \ldots\},\\
M_3 &= \{dinner(caprese, spaghetti\_carbonara, tiramisu), \ldots\},\\
M_4 &= \{dinner(caprese, spaghetti\_carbonara, rasgulla), \ldots\},\\
M_5 &= \{dinner(caprese, matar\_paneer, tiramisu), \ldots\},\\
M_6 &= \{dinner(caprese, matar\_paneer, rasgulla), \ldots\},\\
M_7 &= \{dinner(caprese, lasagna, rasgulla), \ldots\},\\
M_8 &= \{dinner(samosa, lasagna, rasgulla), \ldots\},\\
M_9 &= \{dinner(samosa, matar\_paneer, tiramisu), \ldots\},\\
M_{10} &= \{dinner(samosa, lasagna, tiramisu), \ldots\},\\
M_{11} &= \{dinner(samosa, spaghetti\_carbonara, rasgulla), \ldots\},\\
M_{12} &= \{dinner(samosa, spaghetti\_carbonara, tiramisu), \ldots\},\\
M_{13} &= \{dinner(caprese, idli, rasgulla), \ldots\},\\
M_{14} &= \{dinner(samosa, idli, rasgulla), \ldots\},\\
M_{15} &= \{dinner(caprese, idli, tiramisu), \ldots\},\\
M_{16} &= \{dinner(samosa, idli, tiramisu), \ldots\}.
\end{aligned}
$$

Appendix A.3

In this appendix, we give some formal proofs not included in Section 3.2 for space reasons.

Theorem 5.2 *(Brave Reasoning vs Cautious Reasoning under monotonic aggregation) Let P be a general program, $\mathcal{F}$ a family of models in* {all, stable}*, $\wp$ a weight function, and $\mathcal{A}$ a monotonic aggregation strategy. Then, there exist a program P', a weight function $\wp'$, and a monotonic aggregation strategy $\mathcal{A}'$, all of them polynomially constructible, s.t.* Brave reasoning *for P w.r.t. $(\wp, \mathcal{A})$ reduces to* Cautious reasoning *for P' w.r.t. $(\wp', \mathcal{A}')$, and viceversa.* PROOF. Let P be a general program, q an atom in B_P, $\wp$ an atomic weight assignment, and $\mathcal{A}$ a monotonic aggregation strategy. Define a program P', an atomic weight assignment $\wp'$, and two aggregation strategies $\mathcal{A}'$ and $\mathcal{A}''$ as follows.

$$P' = P \cup \{q \leftarrow q';\ q' \leftarrow q;\ q' \leftarrow \neg q'';\ q'' \leftarrow \neg q'\}$$

where q' and q'' are fresh atoms.

$$\begin{cases} \wp'(p) = \wp(p) & \forall p \notin \{q', q''\} \\ \wp'(q') = c_{q'} & \text{where } c_{q'} \neq \wp'(p)\ \forall p \neq q' \\ \wp'(q'') = c_{q''} & \text{where } c_{q''} \neq \wp'(p)\ \forall p \neq q'' \end{cases}$$

$$\mathcal{A}'(X) = \begin{cases} \mathit{max\_hof} & \text{if } \{c_{q'}, c_{q''}\} \subseteq X \\ A(X - \{c_{q'}, c_{q''}\}) & \text{if } c_{q''} \in X \\ A(X - \{c_{q'}, c_{q''}\}) + 1 & \text{if } c_{q'} \in X \end{cases}$$

$$\mathcal{A}''(X) = \begin{cases} \mathit{max\_hof} & \text{if } \{c_{q'}, c_{q''}\} \subseteq X \\ A(X - \{c_{q'}, c_{q''}\}) + 1 & \text{if } c_{q''} \in X \\ A(X - \{c_{q'}, c_{q''}\}) & \text{if } c_{q'} \in X \end{cases}$$

Note that if $\mathcal{A}$ is monotonic, then both $\mathcal{A}'$ and $\mathcal{A}''$ are monotonic, as well. It is easy to see that the following claims hold.

CLAIM a). $P \models_c^{\mathcal{F}} q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_b^{\mathcal{F}} q'$ w.r.t. $(\wp', \mathcal{A}')$.

CLAIM b). $P \models_c^{\mathcal{F}} \neg q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_b^{\mathcal{F}} q''$ w.r.t. $(\wp', \mathcal{A}'')$.

CLAIM c). $P \models_b^{\mathcal{F}} q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_c^{\mathcal{F}} q'$ w.r.t. $(\wp', \mathcal{A}'')$.

CLAIM d). $P \models_b^{\mathcal{F}} \neg q$ w.r.t. $(\wp, \mathcal{A})$ iff $P' \models_c^{\mathcal{F}} q''$ w.r.t. $(\wp', \mathcal{A}')$. □

Theorem 5.5 *(Positive Program, Monotonic Aggregation) Given a positive disjunctive program P, $\wp$, M, and a monotonic aggregation strategy $\mathcal{A}$ as input, deciding whether M is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is* co-NP*-complete. Hardness holds even if M is known to be a stable model.* PROOF. co-NP-*Membership*. On positive programs, minimal and stable models coincide. As a consequence, also optimal stable models and optimal minimal models coincide. We can thus verify that M is not an optimal stable model by checking that it is not an optimal minimal model. That is, guess $M_1 \subseteq B_P$, and check that: either (1) [M_1 is a model of P] and [$\mathsf{HOF}(\wp, \mathcal{A})(M_1) < \mathsf{HOF}(\wp, \mathcal{A})(M)$], or [$M_1$ is a model of P] and [$M_1 \subseteq M$], or (3) M is not a model of P. Since (1), (2), and (3) are polynomial time tasks, this problem is in NP, and as a consequence, Checking optimal stable models is in co-NP.

co-NP-*Hardness* Given a disjunctive positive program P and an atom q, it is well-known and easy to see that deciding whether $P \models q$ is co-NP-hard. We reduce this problem to optimal stable model checking.

Let q' be a fresh atom and P' be the (positive) program obtained from P by: (i) inserting the atom q' in the head of every rule of P. It is easy to see that

$$\mathsf{MM}(P') = \{q'\} \cup \mathsf{MM}(P)$$

Since both P and P' are positive programs, $\mathsf{ST}(P) = \mathsf{MM}(P)$ and $\mathsf{ST}(P') = \mathsf{MM}(P')$. Therefore:

$$\mathsf{ST}(P') = \{q'\} \cup \mathsf{ST}(P)$$

Let $\wp$ be the following weight function: $\wp(p) = 1$ for any atom $p \in B_P$ s.t. $p \neq q$; $\wp(q) = \wp(q') = |B_P|$. The aggregation function $\mathcal{A}$ is *sum*. Note that *sum* is monotonic.

Let M be a model of P s.t. $q \notin M$. By definition of $\wp$ and $\mathcal{A}$, we have that $\mathcal{A}(M) < |B_P|$. Now, consider the set $\{q'\}$. $\{q'\}$ is a stable model for P' and $\mathcal{A}(\{q'\}) = |B_P|$. Hence, {q'} is an optimal stable model of P' w.r.t. $(\wp, \mathcal{A})$ iff [every model of P contains q], i.e., iff [$P \models q$]. As a consequence, Checking is co-NP-hard. □

Theorem 5.8 *(General Program, Arbitrary/Monotonic Aggregation) Given a disjunctive program P, $\wp$, $\mathcal{A}$ and a ground literal q as input, deciding whether q is true in some optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Δ_3^P-complete even if $\mathcal{A}$ is a monotonic aggregation strategy.* PROOF. ¿From Theorem 5.7, it remains to prove only hardness. We provide a reduction from the same problem used in the proof of that theorem, and show that we can impose monotonicity on the aggregation strategy if we allow negation in the logic program.

Consider the program $LP'(\Phi) = LP(\Phi) \cup \{w \leftarrow \mathbf{not}(w)\}$. We have now that every stable model of $LP'(\Phi)$ must contain w, that is, $\mathsf{ST}(LP'(\Phi)) = \mathsf{ST}_w(LP(\Phi))$. Therefore, there is a one-to-one correspondence between $\mathsf{ST}(LP'(\Phi))$ and the set of truth assignments ϕ which make Φ_ϕ valid.

Now, let $\wp$ be the atomic weight assignment $\wp$ such that: (i) $\wp(x_i) = 2^{n-i}$ $(1 \leq i \leq n)$, and, (ii) $\wp(y) = 0$ if $y \notin \{x_1, \ldots, x_n\}$. Take the monotonic aggregation strategy sum.

Then, the Herbrand objective function $\mathsf{HOF}(\wp, sum)$ induces a total order on $\mathsf{ST}(LP'(\Phi))$. In particular, given two stable models M and M' in $\mathsf{ST}(LP'(\Phi))$, $\mathsf{HOF}(\wp, \mathcal{A})(M) > \mathsf{HOF}(\wp, \mathcal{A})(M')$ if and only if the truth assignment ϕ_M is greater than $\phi_{M'}$ in the lexicographically order. Therefore, $LP'(\Phi)$ has a unique optimal stable model M, corresponding to the lexicographically minimum truth assignment ϕ_{min} such that $\Phi_{\phi_{min}} = \forall Y \ E_{\phi_{min}(X)}$ is valid. Hence, the lexicographically minimum truth assignment $\phi_{min}(X)$ making $\Phi_{\phi_{min}}$ valid fulfills $\phi_{min}(x_n) = true$ if and only if x_n is true in the optimal stable model of $LP'(\Phi)$ w.r.t. $(\wp, \mathcal{A})$ (that is, iff x_n is a brave consequence of $LP(\Phi)$ w.r.t. $(\wp, \mathcal{A})$). Therefore, brave reasoning is Δ_3^P-hard, even if we require the monotonicity of the aggregation strategy (as the used strategy sum is monotonic). □

Theorem 5.9 *(Positive Program, Monotonic Aggregation, Reasoning) Given a disjunctive positive program P, a weight function $\wp$, a monotonic aggregation strategy $\mathcal{A}$, and a ground literal q as input, deciding whether q is true in some (resp., in every) optimal stable model of P w.r.t. $(\wp, \mathcal{A})$ is Σ_2^P-complete (resp., Π_2^P-complete).* PROOF. We can decide the brave reasoning problem as follows. Guess $M \subseteq B_P$, and check that: (1) M is an optimal stable model of P w.r.t. $(\wp, \mathcal{A})$, and (2) q is true w.r.t. M. Clearly, property (2) can be checked in polynomial time; while (1) can be decided by a single call to a co-NP oracle, by virtue of Theorem 5.5. The problem is therefore in Σ_2^P.

Hardness for the brave reasoning problem follows from the Σ_2^P-hardness of (traditional) brave reasoning for disjunctive positive programs under stable model semantics [3].

The computational complexity of cautious reasoning follows immediately because it can be regarded as the complementary problem of brave reasoning. □

Theorem 5.11 *(General/Positive Program, Arbitrary/Monotonic Aggregation) Suppose we consider the "all" models semantics. Given a program P, a weight function $\wp$, an aggregation strategy $\mathcal{A}$, a set of ground literals S and a real number n as input, $Approx(P, \wp, \mathcal{A}, n, S)$ is* NP*-complete. Hardness holds even if P is a positive program and $\mathcal{A}$ is monotonic.* PROOF. We can decide the problem as follows. Guess $M \subseteq B_P$, and check that: (1) M is a model of P, (2) $\mathsf{HOF}(\wp, \mathcal{A})(M) \leq n$, and (3) every literal in S is true w.r.t. M. Clearly, (1), (2), and (3) can be checked in polynomial time, hence the problem is in NP.

Let E be a CNF formula and P_E the corresponding disjunctive positive program. E is satisfiable if and only if $P_E \models_b^{All} \neg contr$. Hence, brave reasoning w.r.t. *All models* is NP-hard, and we conclude by virtue of Proposition 5.1. □

Appendix A.4

In this appendix, we give some formal proofs not included in Section 6 for space reasons.

Theorem 8.1 *Given a disjunctive positive program P, a weight function $\wp$, and a monotonic aggregation strategy $\mathcal{A}$,* Optimal_Stable_Model *runs in polynomial space and in single exponential time, provided that $\wp$ and $\mathcal{A}$ are polynomial time computable.*

PROOF. Since at each call of $Compute\_Optimal$ we add a literal to the partial interpretation I, the maximum depth of the chain of recursive calls is $|B_P|$, in the worst case. It follows that the algorithm uses at most polynomial space, because we assumed all the functions ($\wp$, $\mathcal{A}$, LB_P, and *choose*) are polynomial time computable, and hence they work in polynomial space, too.

For time complexity, note that the algorithm uses calls to $Compute\_Optimal$ to perform a guided search of the interpretations for P. In particular, any interpretation of P is expected at most once. Clearly, we do not generate every interpretation. Indeed, if we detect that an interpretation is inconsistent, or cannot be extended to any optimal model, the recursive call exits and no further extension of such an interpretation is tried. Since the number of interpretations for P is single exponential in $|B_P \cup \neg B_P|$, and every (non recursive) step of $Compute\_Optimal$ is feasible in polynomial time, it follows that *Optimal_Stable_Model* runs in single exponential time, in the worst case. □

Theorem 8.2 *Given a normal positive program P, a weight function $\wp$, and a monotonic aggregation strategy $\mathcal{A}$, the Algorithm* Optimal_Stable_Model *runs in polynomial time, provided that $\wp$ and $\mathcal{A}$ are polynomial time computable.*

PROOF. Let P be a normal positive program. We have to compute an optimal stable model for P w.r.t. some $(\wp, \mathcal{A})$, where $\mathcal{A}$ is monotonic. Let I be the interpretation computed at the end of the **repeat** loop of the first call to the procedure *Compute_Optimal*. It is easy to see that I^+ is the minimum model of the normal positive program P, and hence its unique stable model and its unique optimal stable model, too. Since the set of possibly-true atoms $PT_P(I)$ is empty, either $I^+ = B_P$ or $I^+ \subset B_P$. In either case, the procedure ends and the algorithm outputs the optimal stable model. Note that both the loop and the evaluation of the **if** statement are feasible in polynomial time, since we assumed that $\wp$ and $\mathcal{A}$, and hence HOF, are polynomial-time computable. $\square$

Theorem 8.3 *Given a disjunctive program P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$, the Algorithm* Optimal_Stable_Model_General *runs in polynomial space and in single exponential time, provided that $\wp$ and $\mathcal{A}$ are polynomial time computable.*

PROOF. As observed for the proof of Theorem 8.1, the maximum depth of the chain of recursive calls is $|B_P|$, in the worst case. Moreover, the function *is_Stable* works in polynomial space and single exponential time [10]. It follows that the algorithm uses at most polynomial space.

For the time complexity, note that, as for the previous algorithm, each interpretation of P is generated at most once. Therefore, the number of calls to $Compute\_Optimal\_G$ is single exponential in $|B_P \cup \neg B_P|$. Moreover, the function *is_Stable* can be evaluated in single exponential time, and the other functions ($\wp$, $\mathcal{A}$, LB_P, and *choose*) are polynomial time computable. It follows that *Optimal_Stable_Model_General* runs in single exponential time, in the worst case. $\square$

Theorem 8.4 *Given a normal positive program P, a weight function $\wp$, and an aggregation strategy $\mathcal{A}$, the Algorithm* Optimal_Stable_Model_General *runs in polynomial time, provided that $\wp$ and $\mathcal{A}$ are polynomial-time computable.*

PROOF. Let P be a normal positive program. We have to compute an optimal stable model for P with respect to $(\wp, \mathcal{A})$. ¿From the same reasoning as in the proof of Theorem 8.2, it follows that, at the end of the **repeat** loop of the first call to the procedure *Compute_Optimal_G*, we get the unique stable model of P, which is also the unique optimal stable model of P. Here we just observe that, even in this case, both the loop and the evaluation of the **if** statement are feasible in polynomial time, since *is_Stable* runs in polynomial time for normal logic programs [10], and we assumed that $\wp$ and $\mathcal{A}$, and hence HOF, are polynomial time computable. $\square$