Expanding queries to incomplete databases by interpolating general logic programs.[1]

Chitta Baral, Michael Gelfond and Olga Kosheleva
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, U.S.A.
{chitta,mgelfond,olga}@cs.utep.edu, http://cs.utep.edu/krgroup.html
Phone: 915-747-5480, Fax: 915-747-5030

Abstract

In databases, queries are usually defined on complete databases. In this paper we introduce and motivate the notion of extended queries that are defined on incomplete databases. We argue that the language of extended logic program is appropriate for representing extended queries. We show through examples that given a query, a particular extension of it has important characteristics which corresponds to removal of the CWA from the original specification of the query. We refer to this particular extension as the *expansion* of the original query.

Normally queries are expressed as general logic programs. We develop an algorithm that given a general logic program (satisfying certain syntactic properties) expressing a query constructs an extended logic program that expresses the expanded query. The extended logic program is referred to as the *interpolation* of the given general logic program.

1 Introduction and Motivation

In this paper we introduce the concept of *interpolating* general logic programs. We will also motivate interpolation of general logic programs from the point of view of expanding 'queries' [2] from a domain consisting of only complete databases to a domain that allows incomplete databases of a particular kind.

1.1 Expressing transitive closure of incomplete information?

Consider a domain $\mathcal{U}$ of individuals, Sam, John, Peter, Mary, and Alberto. Now consider a database instance D consisting of the facts

$par(sam, john)$[3], $par(john, peter)$, and $par(mary, alberto)$;

where $par(X, Y)$ means that Y is a parent of X.

Now if we would like to ask a *query Q* about the ancestor and non-ancestor pairs in D, we can express it using the general logic program Π_0:

[1]This paper is based on the paper [BGK93], where the term *approximation* was used instead of the term *interpolation.* As pointed out by the anonymous referees and several other colleagues we agree with the inappropriateness of the term 'approximation' and use the term 'interpolation' instead.

[2]Our use of the term 'query' is based on the standard definition of a query in the area of Databases [AHV95]. We will formally define the term 'query' in Section2.1.

[3]In the atom $par(sam, john)$ we use 'sam' instead of 'Sam' to follow the usual convention of variables staring with a capital letter and constants starting with small letters.

$$\left.\begin{array}{l} anc(X,Y) \leftarrow par(X,Y) \\ anc(X,Y) \leftarrow par(X,Z), anc(Z,Y) \end{array}\right\} \Pi_0$$

The above representation of the *query* assumes the Close World Assumption (CWA) [Rei78] about the database[4] D, which says that an atom f will be inferred to be $false$ w.r.t. D if it is not entailed by D. We also refer to D as a *complete database*. By $CWA(D)$ we denote the set $\{f \ : \ f$ is an atom in the language and $f \notin D\}$, and the meaning of a complete database D is expressed by the set $D \cup \neg CWA(D)$, where for any set S of atoms $\neg S$ denotes the set $\{\neg f \ : \ f \in S\}$.

The *query* Q represented by the general logic program Π_0 can be considered as a function from instances of the relation par to instances of the relation anc. For example,

$Q(\{par(sam, john), par(john, peter), par(mary, alberto)\}) =$
$\quad \{anc(sam, john), anc(john, peter), anc(sam, peter), anc(mary, alberto)\}$.

The function Q can be expressed by the general logic program Π_0 in the following way:

$$Q(D) = \{anc(X,Y) \ : \ anc(X,Y) \in least\ model^5\ of\ (\Pi_0 \cup D) \tag{1}$$

Now, let us consider the domain $\{a, b, c, d, e\}$, where a, b, c, d, and e denote remains of five different individuals found from an archaeological site. Using sophisticated tests scientists are able to determine that b is a parent of a and c is a parent of b, and neither e is parent of any of a, b, c and d nor any of them is the parent of e. This information can be represented by the following set S of literals:

$S = \{par(a, b), par(b, c), \neg par(a, e), \neg par(b, e), \neg par(c, e), \neg par(d, e), \neg par(e, a),$

$\quad \neg par(e, b), \neg par(e, c), \neg par(e, d), \neg par(a, d)\}$.

The set S is not a database instance because it contains negative literals. Sets with negative literals will be called *incomplete database instances* or simply incomplete databases. More formally, an incomplete database is of the form $D \cup \neg S$, where $S \subset CWA(D)$. There has been several different approaches to represent incomplete informations which includes null values[Gra77, Zan84], Codd tables, and conditional tables [Gra91, IL84a, IL84b] in Databases, use of disjunctions, classical negations and abductions in extensions of logic programming(For a survey on this see [BG94]).

As before we are interested in ancestor and non-ancestor pairs. But this time with respect to the incomplete database S. The function Q and our earlier concept of *query* are no longer appropriate as they require database instances as input. To define ancestor and non-ancestor pairs using the information in S, we need to expand the notion of *query* to that of *extended query* which allows input to be incomplete databases.

Although, the approaches in databases, particularly conditional tables allow us to represent the set S it is not clear how to express extended queries (such as the query about the ancestor and non-ancestor relationship with respect to the incomplete database S) using them.

Neither can we ignore the negative literals in S and compute Q with respect to the database consisting of the positive literals in S? If we do that we will be able to infer $\neg anc(a, d)$. This is

[4]We will often use the term 'database' to mean 'database instance'.
[5]Since $\Pi_0 \cup D$ is a Horn logic program it has a least model [vEK76].

not correct intuitively, because it is possible that with further tests the scientist may determine $par(c,d)$, which will be consistent with the current determination and which will force us to infer $anc(a,d)$.

Since we can not ignore the negative literals in S, and general logic programs such as $\Pi_0 \cup D$ do not allow rules with negative literals in their head we need to extend the standard language of general logic programs to allow convenient expression of incomplete knowledge. We will use the language of extended logic programs (ELP's)[6] under the answer set semantics from [GL91] (see also [PW89]) containing two types of negation: negation as failure *not* and classical negation[7] $\neg$.

An extended logic program allows the head of a rule and the atoms in the body of the rule to be literals, where literals are either atoms or atoms preceded by $\neg$. An extended logic program T can be viewed as a specification of sets of literals called answer (belief) sets of T which represent possible theories about the world that can be built by a rational reasoner based on the information from T. For any literal l, T entails l if l belongs to all answer sets of T. If T neither entails l nor entails $\bar{l}$, the literal opposite[8] of l we say l is *unknown* w.r.t. T.

Recall that we were trying to represent the query that determines ancestor and non-ancestor pairs from S and we now believe that extended logic programs may be the language to express such a query. Let us denote this query by Q'. The next question is how are Q and Q' related?

While Q' allows incomplete databases as inputs, Q only allows complete databases. But they must coincide when the inputs are complete databases. Moreover, for any incomplete database X', $Q'(X')$ must not disagree[9] with $Q(X)$, for any complete database X that extends X'. The intuition behind it is that if currently we have the incomplete database X' then our partial conclusion about ancestors and non-ancestors should be such that in presence of additional information we do not retract our earlier conclusion. Also, given an incomplete database X', the extended query Q' should be such that $Q'(X')$ contains all information about ancestors and non-ancestors that 'critically depend' on X'. In other words for any X', if for all complete database X that agree with X' an ancestor pair is true in $Q(X)$, then that ancestor pair must be true in $Q'(X')$. Similarly, about non-ancestor pairs. In general, given a query Q an extended query Q' that satisfies the above properties is referred to as its *expansion*. We formally define *expansion* of a query by an extended query and relate it to the above described properties in Definition 3.4 and Proposition 3.1 respectively.

Now that we have an intuitive idea about Q', the next question is how to express it using an extended logic program.

Let us start with the general logic program Π that expresses the query Q. Since, we would like to allow incomplete databases as inputs, let us consider Π as an extended logic program. But then we no longer have the CWA about ancestor. The next step would be to consider the extended logic program obtained by adding explicit CWA about ancestors to the general logic program Π representing Q. The resultant extended logic program is given as follows:

$\neg anc(X,Y) \leftarrow not\ anc(X,Y)$

[6] The usefulness of extended logic programs to represent incomplete information has been discussed in [GL91, BG94] from the point of view of knowledge representation. A thorough treatise with respect to databases is beyond the scope of this paper and is one of our next goals.

[7] This negation is called "classical", "strong", and "explicit" by various authors who associate different meanings to it. Pearce and Wagner [PW89] show that this negation has close ties with Nelson's constructive negation [Nel49]. In this paper we refer to it as "classical" negation.

[8] More specifically, by $\bar{l}$ we will denote the literal $\neg l$ and we will simplify $\neg\neg A$ to A.

[9] Two sets of literals S_1 and S_2 are said to disagree if $S_1 \cap \neg S_2 \neq \emptyset$. Otherwise we say that they agree.

$anc(X,Y) \leftarrow par(x,Y)$

$anc(X,Y) \leftarrow par(X,Z), anc(Z,Y)$

But the resultant extended logic program is not an adequate expression of Q'. It only works when the input database instance is complete and does not work for S, as it incorrectly infers $\neg anc(a,d)$.

It is not obvious how to express this extended query Q' and to our knowledge there has not been any logic programming representation of this extended query in the literature.

In this paper we express this extended query through an extended logic program and prove that it expands the query expressed by Π. *The more general question we answer in this paper is how to expand queries from complete databases that are expressed by general logic programs so that they are applicable to incomplete databases.* We use the term 'expand' because our goal is to expand the domain of the queries from complete databases only to databases that may be incomplete.

For convenience, we also introduce the term *interpolation* which is defined as follows. Given a general logic program Π that represents a query Q, we refer to the extended logic program T that expands Q as the *interpolation* of Π. In other words we say that T *interpolates* Π. The intuition behind the term 'interpolation' is that T agrees with Π on all inputs where Π is defined, and for inputs where Π is not defined T interpolates to a value based on the mappings of Π on the neighboring (complete) inputs.

1.2 The 'Birds normally Fly' example

Before we get to the details of our formalization, let us consider another motivating example from the literature of knowledge representation and common-sense reasoning. Through this example we will show the applicability of the notion of 'expansion' in the context of knowledge representation and common-sense reasoning.

This example (originally from [McC86]) formalizes the sentence 'birds normally fly' and some associated information about birds, and abnormal birds. Unlike the ancestor example, logic programs that express both the original query (about flying ability of birds, when the input is a complete database) and the expanded query corresponding to this example appears in the literature [BG94], albeit without any formal discussion about 'expansion' or 'interpolation'. We will now analyze those programs using the notion of 'expansion'.

Example 1.1 Suppose that we are told that penguins are birds that do not fly, that birds normally fly, and that Tweety is a bird and Sam is a penguin. Let us also assume that this information is *complete*.

We need to represent this information in a way which will allow us to

(a) conveniently reason about it,

(b) incorporate new information in the database without removing anything from it.

The example served as a testing ground for various nonmonotonic formalisms. Assuming that the information given is complete, we can easily represent knowledge from the example by the general logic program Π_1 consisting of the rules:

$$\left.\begin{array}{ll} r1. & f(X) \leftarrow b(X), not \;\; ab(X) \\ r2. & b(X) \leftarrow p(X) \\ r3. & ab(X) \leftarrow p(X) \end{array}\right\} \Pi_1$$

used in conjunction with the facts

f1. $b(t) \leftarrow$
f2. $p(s) \leftarrow$

Here f, b, p stand for flies, bird, penguin relations, ab denotes special predicate abnormal, t stands for Tweety and s for Sam. The completeness assumption is captured by the CWA embedded in the semantics of general logic programs. □

It is easy to see that, according to this representation, Sam does not fly while Tweety is a flying bird and not a penguin. If we learn that Tweety is a penguin we will update the collection of facts by adding to it $p(t) \leftarrow$. This will lead to the revision of our beliefs about flying abilities of Tweety but leaves the closed world assumption, incorporated in the semantics of general logic programs, unchanged.

Let us now consider the case where we do not automatically assume completeness about the information on birds and penguins. i.e. We would like to remove the CWA about birds and penguins. Now let us continue with Example 1.1.

Example 1.2 Suppose we are given the same general information as in Example 1.1, except for the closed world assumption. Let us also assume that Tweety , Opus and Sam are birds, Sam is a penguin, Tweety is not and about Opus we do not know. Notice that since Opus may be a penguin we can not conclude that it flies. How can we represent this information?

Notice the correspondence between removing CWA about the input relations and 'expanding' the query. To expand the query we will first try to add rules that use incomplete information about birds and penguins and make conclusion about non-flying birds. Since we use extended logic programs and ELPs do not have implicit CWA (unlike general logic programs) we need explicit rules that can make conclusion about non-flying birds.

The first natural attempt of finding an extended logic program representing knowledge from Example 1.2 seems to lead to an extended program T' consisting of Π_1 from Example 1.1 and the rules $r4$ and $r5$

$$\left.\begin{array}{ll} r1. & f(X) \leftarrow b(X), not \;\; ab(X) \\ r2. & b(X) \leftarrow p(X) \\ r3. & ab(X) \leftarrow p(X) \\ r4. & \neg f(X) \leftarrow p(X) \\ r5. & \neg f(X) \leftarrow \neg b(X) \end{array}\right\} T'$$

Unlike Π_1 the program T' will be used in conjunction with both, positive and negatives facts. such as the facts (g1) – (g5) below, where o stands for Opus.

g1. $b(t) \leftarrow$
g2. $\neg p(t) \leftarrow$
g3. $b(o) \leftarrow$
g4. $p(s) \leftarrow$
g5. $b(s) \leftarrow$

The resulting representation works correctly w.r.t. queries about Tweety and Sam (about whom we have complete information in terms of the predicates bird and penguin). For these queries the same answers are produced by Π_1 and T', when used in conjunction with f1-f2 and g1-g5 respectively. The situation changes however if we are interested in queries about Opus. Notice that we do not know if Opus is a penguin or not. i.e. Neither $p(o)$ nor $\neg p(o)$ is entailed by our representation. This means that the answer that T' (in conjunction with g1-g5) will give to query $p(o)$ is *unknown* which is of course correct (though different from the *no* answer given to this query by $\Pi_1 \cup \{f1, f2\}$) and reflects our refusal to use the closed world assumption for penguins. Unfortunately, T' (in conjunction with g1-g5) does not exhibit similar behavior w.r.t. the flying property of birds. In particular it concludes that Opus is a flying bird which contradicts our intuition.

To correct the problem let us replace rule (r3) of T' by a new rule similar to the rules used in [GL92].

$r3'.\ ab(X) \leftarrow not\ \neg p(X)$

and have the program T_1 consisting of the following rules.

$$\left.\begin{array}{ll} r1. & f(X) \leftarrow b(X), not\ \ ab(X) \\ r2. & b(X) \leftarrow p(X) \\ r3'. & ab(X) \leftarrow not\ \neg p(X) \\ r4. & \neg f(X) \leftarrow p(X) \\ r5. & \neg f(X) \leftarrow \neg b(X) \end{array}\right\} T_1$$

The program T_1 is more cautious than T'. It agrees with T' on queries about Tweety and Sam but all inquires about properties of Opus (except him being a bird) are (correctly) answered as *unknown*. □

We now informally argue that this is no accident and that T_1 exhibits the 'correct behavior' if used in conjunction with any collection of facts formed by literals with predicate symbols b and p.

Let us consider the examples in this subsection (Example 1.1 and 1.2) using the terminology of databases and queries. In Example 1.1 we are given a complete database about birds and penguins expressed by the set $\{f1, f2\}$ and the program Π_1 expresses the query about flying and non-flying birds. In Example 1.2 we are looking for an extended logic program that can reason about flying and non-flying birds given an incomplete database about birds and penguins expressed by the set $\{g1, g2, g3, g4, g5\}$. In other words we are looking for the expansion of the query represented by Π_1. Hence, to prove that T_1 exhibits the correct behavior we need to show that T_1 expands Π_1. We formally state this in Section 3 and give a proof.

The rest of the paper is organized as follows: In Section 2 we review the notion of queries and extended logic programs. In Section 3 we formalize the notions of query expansion and interpolation of a general program Π and illustrate it by way of examples. In Section 4 we discuss an algorithm that constructs query expansions and interpolations for a large class of general logic programs.

2 Preliminaries

This section contains formal definition of a query and also information about extended logic programs. Some results about general and extended logic programs necessary to make the paper self-contained are presented in the Appendix.

2.1 Databases and Queries

A *relation schema* R_i has a name N_i and a finite list of attributes $L_i = \langle A_1, \ldots, A_{l_i} \rangle$, where l_i is the arity of the relation schema R_i. It will sometimes be denoted as $R_i(A_1, \ldots, A_{l_i})$. A *database schema* [Ull88] R is a finite set $\{R_1, \ldots, R_n\}$ of relation schemata. $\mathcal{U}$ is an arbitrarily large but finite set of objects that can be used in the relations and is referred to as the *domain*. Given a relation schema R_i, a *relation instance* is a set of tuples of the form $\langle a_1, \ldots, a_{l_i} \rangle$, where $\{a_1, \ldots, a_{l_i}\} \subset \mathcal{U}$. This tuple may also be denoted by the atom $R_i(a_1, \ldots, a_{l_i})$. A *database instance* W is a set of relation instances.

Traditionally, the intuitive meaning of a database instance W is based on the CWA. i.e. if $R_i(a_1, \ldots, a_{l_i}) \in W$ then we say that $R_i(a_1, \ldots, a_{l_i})$ is $true$ w.r.t. W, otherwise we say that $R_i(a_1, \ldots, a_{l_i})$ is $false$ w.r.t. W.

In presence of incompleteness we extend the notion of relation instance to *incomplete relation instances* which may consist of positive literals (or atoms) of the form $R_i(a_1, \ldots, a_{l_i})$ and negative literals of the form $\neg R_j(a_1, \ldots, a_{l_j})$. An *incomplete database instance* is a set of incomplete relation instances. When dealing with incomplete database instance we no longer use the CWA. Given a relation schema R_k and an incomplete database instance W, we say $R_k(a_1, \ldots, a_{l_k})$ is $true$ w.r.t. W if $R_k(a_1, \ldots, a_{l_k}) \in W$, we say $R_k(a_1, \ldots, a_{l_k})$ is $false$ w.r.t. W if $\neg R_k(a_1, \ldots, a_{l_k}) \in W$; otherwise we say $R_k(a_1, \ldots, a_{l_k})$ is $unknown$ w.r.t. W.

A *query* from a database schema R to a database schema S is a partial[10] mapping from instances of R to (incomplete)[11] instances of S. An *extended query* from a database schema R to a database schema S is a partial mapping from incomplete instances of R to incomplete instances of S. In this paper we consider queries that are represented by general logic programs and extended queries that are represented by extended logic programs.

It should be noted that viewing a general logic program (or an extended logic program) as a specification of a theory about the world does not contradict with viewing it as a query (or an extended query). In the first case it is a function with no inputs, similar to the view where constants in a first-order language are considered as functions of arity zero.

2.2 Extended Logic Programs

Before we define the syntax and semantics of extended logic programs let us introduce some necessary notations.

Let *Lit* be the set of ground literals in a first-order language L. If $p_1, \ldots, p_n$ are predicate symbols from L by $Lit(p_1, \ldots, p_n)$ we will denote the collection of ground literals whose predicates are from $\{p_1, \ldots, p_n\}$. For a set of ground literals I, $atoms(I)$ denotes the set of ground atoms in I.

An extended logic program is a set of rules of the form:

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{2}$$

where l_i's are literals and *not* denotes negation-as-failure.

When l_i's are atoms the program is called a *general* logic program. (For any extended logic program T, T^+ represents the general logic program obtained from T by replacing $\neg l$ by l' in all rules of

[10] Partial mapping is allowed to take into account programs that may not always terminate, or programs that do not have a semantics. For example the program $p \leftarrow \neg p$ does not have a stable model [GL88].

[11] This is to take into account the fact that a program may have multiple stable models.

T. A detailed construction of T^+ from T, and the relation between stable models of T and answer sets of T^+ is presented in [GL90].) General logic programs that do not have *not* (i.e. those with m = n) are called *definite* logic programs. The answer set semantics of extended programs [GL91] treats a rule with variables as shorthand for the set of its ground instances. It is sufficient then to define answer sets for extended programs without variables.

To define the answer set semantics of extended logic programs let us first consider programs without negation-as-failure.

The *answer set* of such a program Π is the smallest subset S of *Lit* such that

(i) for any rule $l_0 \leftarrow l_1, \ldots, l_m$ from Π, if $l_1, \ldots, l_m \in S$, then $l_0 \in S$;

(ii) if S contains a pair of complementary literals, then $S = Lit$.

The answer set of a program Π that does not contain negation-as-failure is denoted by $\alpha(\Pi)$.

Now let Π be *any* extended program without variables. For any set $S \subset Lit$, let Π^S be the extended program obtained from Π by deleting

(i) each rule that has a formula *not* l in its body with $l \in S$, and

(ii) all formulas of the form *not* l in the bodies of the remaining rules.

Clearly, Π^S does not contain *not* , so that its answer set is already defined. If this answer set coincides with S, then we say that S is an *answer set* of Π. In other words, the answer sets of Π are characterized by the equation $S = \alpha(\Pi^S)$. In the absence of $\neg$ in the program, answer sets coincide with stable models.

For a general program Π and an atom f we say $\Pi \models f$ iff f is $true$ in all stable models [GL88] of Π. Similarly, we say $\Pi \models \neg f$ iff f is $false$ in all stable models of Π. For an extended logic program T and a literal f we say $T \models f$ iff f belongs to all answer sets of T. An extended logic program is called *consistent* if it has at least one consistent answer set. An extended logic program is called *contradictory*, if it has an inconsistent answer set (that is, an answer set containing a pair of complementary literals).

A general logic program is said to be *stratified* [ABW88] [VG88] if there is no recursion through negation-as-failure *not* . A stratified general logic program has a unique stable model [GL88].

A general logic program is said to be signed [Kun89] if there is a set S (called signing) of ground atoms such that, for any ground instance of a rule of the type (1), either
$\{l_1, \ldots, l_m\} \subset S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$
or
$\{l_0, l_1, \ldots, l_m\} \cap S = \emptyset$ and $\{l_{m+1}, \ldots, l_n\} \subset S$.

A general logic program is said to be call-consistent[Kun89] if its dependency graph does not have a cycle with odd number of negative edges. A general logic program is said to be acyclic if its dependency graph does not have a cycle with only positive edges.

A logic program (general or extended) is called *coherent*, if it has at least one stable model. It is called *categorical*, if it has exactly one stable model.

In the appendix we give several theorems about general and extended logic programs from the literature that will be used in proving the various results of the paper.

3 Query Expansion and Interpolation

Let us revisit the definition of *query* in Section 2.1. It is defined as a partial mapping from database instances to (incomplete) database instances. We do not require that the input and output relations of the query to be disjoint. Since the terms 'input' and 'output' connotate disjointness we would refer to them as *parameters* and *values* of the query instead, and for a query Q, we will denoted them by $\mathcal{P}_Q$ and $\mathcal{V}_Q$, respectively. For generality, we allow parameters and values to be arbitrary sets of literals, without requiring them to be literals of some particular sets of predicates. For a program Π over a language $\mathcal{L}$, the parameters and values of the query it represents will be denoted by $\mathcal{P}_\Pi$ and $\mathcal{V}_\Pi$. (The index Π will be omitted whenever possible.) We will often refer to $\mathcal{P}_\Pi$ and $\mathcal{V}_\Pi$ as the parameters and values of Π. To specify the mapping represented by a general logic program and an extended logic program we need the following definitions.

Definition 3.1 Let R and S be sets of ground literals over a language $\mathcal{L}$ of an extended logic program Π. $\Pi_R \mid S$ the S-consequences of Π and R, is defined as follows:
$\Pi_R \mid S = \{s : s \in S \; and \; \Pi \cup R \models s\}$ □

Using the above notation we now precisely define a general logic program that represents a query Q to be a (partial) function from complete sets of literals from the parameter (of Q) to sets of literals from the value (of Q). But, since both parameter and values may have literals, we will have to be careful about the domain of the query specified by a general logic program. The first requirement is that the elements of the domain must be *complete* w.r.t. $\mathcal{P}$. (Formally, a set of literals S is said to be *complete* w.r.t. a set of literals P if for any atom in P either the atom or its negation is in S.) But, not all complete sets of literals from the parameter will be in the domain. We will additionally require that each element X of the domain should be a valid input in the sense that when added to Π the resulting general logic program should not entail different literals from the parameter than that is in X. The following definition makes this precise.

Definition 3.2 Let X be an arbitrary set from $2^{\mathcal{P}}$ and let $Y = \Pi_{atoms(X)}|\mathcal{V}$. We will say that X is a *valid input* of Π ($X \in Dom(\Pi)$) and Y is the value of Π at X (i.e. $Y = \Pi(X)$) if the following hold:

1. $X = \Pi_{atoms(X)} \mid \mathcal{P}$
2. X is complete w.r.t. $\mathcal{P}$.

Consider the general logic program Π_1 from Example 1.1 with $\mathcal{P} = Lit(p, b)$ and $\mathcal{V} = Lit(f)$ and assume that t is the only constant in the language of Π_1. It is easy to see that $X_1 = \{b(t), \neg p(t)\}$ and $X_2 = \{b(t), p(t)\}$ belong to the domain of Π_1 and that $\Pi_1(X_1) = \{f(t)\}$ and $\Pi_1(X_2) = \{\neg f(t)\}$.

However, $X = \{p(t), \neg b(t)\}$ does not belong to the domain of Π_1, since $\Pi_1 \cup \{p(t) \leftarrow\} \not\models \neg b(t)$ and, hence, $X \neq \Pi_{1\{p(t)\}} \mid \mathcal{P}$. The intuition behind not allowing X to be in the domain of Π_1 is that when X is added to Π_1 the resulting program does not entail $\neg b(t)$ which is entailed by X and which is an element of $\mathcal{P}$.

Definition 3.3 For any set $X \in 2^{\mathcal{P}}$, a superset $\hat{X}$ of X is said to be a Π-*extension of* X if $\hat{X} \in Dom(\Pi)$. We denote the set of all Π-extension of X by $S_\Pi(X)$. We omit Π from $S_\Pi(X)$ when it is clear from context.

Intuitively, given a set $X \in 2^{\mathcal{P}}$, $S_\Pi(X)$ denotes the different ways X can be completed with additional consistent information, and still be a valid input for Π.

We are now almost ready to precisely define the interpolation of a general logic program. In this paper we specify the interpolation through an extended logic program. We view an extended logic program T to be a function from $2^{\mathcal{P}}$ to $2^{\mathcal{V}}$ such that $T(X) = T_X \mid \mathcal{V} = \{s \ : s \in \mathcal{V} \ and \ T \cup X \models s\}$. Since we are only interested in extended logic programs that are interpolations, we do not restrict the domain of T.

Let T_1 be the extended program from Example 1.2 with $\mathcal{P} = Lit(p, b)$ and $\mathcal{V} = Lit(f)$ and assume that t is the only constant in the language of T_1. It is easy to see that $T_1(\{b(t)\}) = \emptyset$ while $T_1(\{b(t), p(t)\}) = \{\neg f(t)\}$ and $T_1(\{b(t), \neg p(t)\}) = \{f(t)\}$. Notice that both sets $\{b(t), p(t)\}$ and $\{b(t), \neg p(t)\}$ are Π_1-extensions of $\{b(t)\}$ and that T_1 coincides with Π_1 on these extensions. (I.e., the value obtained by applying T_1 to these extensions is same as the value obtained by applying Π_1 to them.)

Now we are ready for the main definition of this paper:

Definition 3.4 (Interpolation) Let Π be a general logic program representing a query Q, with parameters $\mathcal{P}$ and values $\mathcal{V}$. We say that an extended logic program T *interpolates* Π, w.r.t $\mathcal{P}$ and $\mathcal{V}$ if for every $X \in 2^{\mathcal{P}}$

$$T(X) = \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X}) \tag{3}$$

Moreover, we also say that the extended query represented by (the extended logic program) T expands the query represented by (the general logic program) Π. □

For convenience, we will just say that T interpolates Π without mentioning the query Q, the parameters $\mathcal{P}$ and the values $\mathcal{V}$, whenever they are clear from the context. But it should be noted that T interpolates Π w.r.t. $\mathcal{P}$ and $\mathcal{V}$ does not necessarily mean that T will also be an interpolation of Π for a different query with a different pair of parameter and value. [12] The choice of Q, $\mathcal{P}$ and $\mathcal{V}$ is an integral part of the program. The programmer designs the program with that choice in mind. This is similar to the choices and assumptions a Prolog programmer makes about whether a particular attribute of a predicate in the head of a rule will be ground or not when that predicate is invoked.

The following proposition breaks down the definition of interpolation to three different intuitive properties: equivalence, monotonicity and maximal informativeness. The equivalence property states that Π and T must be equivalent w.r.t. complete inputs. The monotonicity property states that T which accepts incomplete inputs should be monotonic, i.e. in the presence of additional consistent information it should not retract any of its earlier conclusions. The maximal informativeness property states that given an incomplete input X, the interpolation T should entail all literals that are entailed by Π w.r.t. all the complete extensions of X that are in the domain of Π. Intuitively, it means that if Π entails l regardless of what complete extension of X is given to Π as an input then X has enough information to make a decision on l and hence the interpolation T should entail l with X as the input.

[12] It is interesting to study how the meaning of a general or an extended logic program changes if other parameter and/or values are chosen while keeping an identical set of rules. Some answers to this question can be found in [GP96].

Proposition 3.1 An extended logic program T interpolates a general logic program Π iff the following conditions are satisfied:

1. (**Equivalence**) For every $X \in Dom(\Pi)$, $\Pi(X) = T(X)$.
2. (**Monotonicity**) T is monotonic, i.e. for every $X_1, X_2 \subseteq \mathcal{P}$, if $X_1 \subseteq X_2$ then $T(X_1) \subseteq T(X_2)$.
3. (**Maximal Informativeness**) For every $v \in \mathcal{V}$ and every $X \subseteq \mathcal{P}$, if for all $\hat{X} \in S(X)$ $v \in \Pi(\hat{X})$ then $v \in T(X)$. □

Proof:
"$\Longrightarrow$"
Since every $X \in Dom(\Pi)$ is complete $S(X) = X$. Hence, condition 1 holds. It is obvious that condition 3 holds. Condition 2 holds because $X_2 \subseteq X_1 \Rightarrow S(X_1) \subseteq S(X_2)$

"$\Longleftarrow$"
Condition 3 implies $T(X) \supseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$.
We now only need to show that $\forall X \subseteq \mathcal{P}$, $T(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$.
(case 1) $S(X) = \emptyset$
$\Rightarrow \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X}) = Lit(\mathcal{P})$
$\Rightarrow T(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$
(case 2) $S(X) \neq \emptyset$
Let $s \in T(X)$. Since, for all $\hat{X} \in S(X)$, $X \subseteq \hat{X}$, by monotonicity (2.) we have $s \in T(\hat{X})$. But using equivalence (1.) we have $s \in \Pi(\hat{X})$. Hence,

$$\forall X \subseteq \mathcal{P}, T(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$$

□

Let us go back to Π_1 and T_1 in Section 1. We are now in a position to rigorously state and prove the following proposition:

Proposition 3.2 Let T_1 and Π_1 be the logic programs defined in Example 1.2. T_1 is an interpolation of Π_1, for $\mathcal{P} = Lit(p, b)$ and $\mathcal{V} = Lit(f)$. □

The proof of the above proposition is presented in the appendix. It should be noted that it does not follow from the main result of the paper given in the next section.

The following example shows why T' from Example 1.2 is not an interpolation of Π.

Example 3.1 Consider the extended logic program T' from Example 1.2. It is easy to see that $T'(\{b(t)\})$ contains $f(t)$ while, $T'(\{b(t), p(t)\})$ contains $\neg f(t)$. I.e, $T'(\{b(t)\}) \not\subseteq T'(\{b(t), p(t)\})$ and hence T' can not be an interpolation of Π_1. □

The proof of Proposition 3.2 substantially relies on the stratifiability of Π_1. In the next example we give an interpolation of a non-stratified program.

Example 3.2 Consider the following general logic program Π_2:

$$\left.\begin{array}{l} c(X) \leftarrow p(X) \\ c(X) \leftarrow q(X) \\ p(X) \leftarrow not \;\; q(X), r(X) \\ q(X) \leftarrow not \;\; p(X), r(X) \end{array}\right\} \Pi_2$$

Here $\mathcal{P} = Lit(r)$ and $\mathcal{V} = Lit(c)$. It is easy to see that for every $X \in 2^{\mathcal{P}}$, $\Pi_2 \cup X$ is call-consistent [Kun89]. Using results from [LT94] it is easy to check that Π_2 is defined for any complete and consistent X, $X \subseteq \mathcal{P}$. (The program $\Pi_2 \cup X$ can then be split into three layers, the bottom layer consisting of X, the next layer consisting of the rules with either p or q in the head and the top layer consisting of the rules with c in the head. The stable models of $\Pi_2 \cup X$ can then be computed bottom-up starting from the bottom layer.)

Consider T_2 obtained from Π_2 by adding to it the rule

$\neg c(X) \leftarrow \neg r(X)$. □

Proposition 3.3 T_2 is an interpolation of Π_2. □

So far in this section we have made precise the notion of interpolation (and query expansion) and proved the interpolation results for two particular general logic programs. But we still do not know which program interpolates the program Π_0 that represents the ancestor query. Our goal now is to come up with a precise algorithm that constructs interpolations of general logic programs. Such an algorithm is very important. Currently there are a large number of queries that are expressed by general logic programs. When there is a need to expand these queries so that they accept incomplete database instances, such an algorithm will be very useful. It should be noted that lack of similar techniques in databases has caused many a problems when moving from older database models (such as hierarchical and network models) to relational model. Our algorithm interpolates a large class of general logic programs, but is not able to interpolate all them. We introduce the notions of *weak interpolation* and *sound interpolation*, two weaker notions of interpolation, and show that the program obtained by our algorithm is a weak interpolation for stratified programs and is a sound interpolation for all programs.

Definition 3.5 (Weak and Sound Interpolation) Let Π be a general logic program, with parameter $\mathcal{P}$ and value $\mathcal{V}$. We say an extended logic program T is a *weak interpolation* of Π w.r.t. $\mathcal{P}$ and $\mathcal{V}$ if the following three conditions are satisfied.

1. (**Monotonicity**) For every $X_1, X_2 \subseteq \mathcal{P}$, if $X_1 \subseteq X_2$ then $T(X_1) \subseteq T(X_2)$.
2. (**Equivalence**) For every $X \in Dom(\Pi)$, $\Pi(X) = T(X)$.
3. For any $X \subseteq \mathcal{P}$
$$T(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$$

If only the first and the third conditions are satisfied then we say T to be a *sound interpolation* of Π w.r.t. $\mathcal{P}$ and $\mathcal{V}$. □

From the third condition above it is clear that if T is a sound interpolation of Π then it satisfies the weak equivalence property which is:

For every $X \in Dom(\Pi)$, $T(X) \subseteq \Pi(X)$.

4 Interpolating General Logic Programs

In this section we present an algorithm which constructs an interpolation of a large class of general logic programs with some restrictions on its parameter and value. We now make these restrictions precise.

Let Π be a general logic program in language $\mathcal{L}$. We only consider the query Q with values $\mathcal{V}$ consisting of all ground literals formed with predicates in the heads of the rules in Π (called IDB predicates), and parameters $\mathcal{P}$ consisting of all other ground literals in *Lit* (called EDB predicates). A general logic program Π representing a query Q that satisfies the above property is called a *natural representation* of Q.

Before we give the algorithm we demonstrate the intuition behind the algorithm by using the ancestor query as an example.

4.1 Interpolation of the Transitive Closure Program

Recall the ancestor query represented as general logic program was:

$$\left.\begin{array}{l} anc(X,Y) \leftarrow par(X,Y) \\ anc(X,Y) \leftarrow par(X,Z), anc(Z,Y) \end{array}\right\} \Pi_0$$

where *par* is the EDB predicate and *anc* is the IDB predicate.

Our goal is to construct an interpolation of this program. We refer to our interpolation program as $\mathcal{A}(\Pi_0)$.

The main idea behind our construction is to have a program which coincides with Π_0 on atoms of the form $anc(a,b)$ and derives $\neg anc(a,b)$ when no consistent extension of the original EDB may derive $anc(a,b)$. To achieve this we introduce a new predicate m_{anc}, where $m_{anc}(c,d)$ intuitively means that " c may bean ancestor of d". Our information about m_{anc} is complete hence $\neg m_{anc}$ coincides with *not* m_{anc} and hence we define $\neg anc$ using the rule:

$\neg anc(X,Y) \leftarrow not\ m_{anc}(X,Y)$.

Next we modify the original rules to define m_{anc} and use another new predicate m_{par}, where $m_{par}(c,d)$ means that "c may be a parent of d". We now define m_{anc} using the rules:

$m_{anc}(X,Y) \leftarrow m_{par}(X,Y)$
$m_{anc}(X,Y) \leftarrow m_{par}(X,Z), m_{anc}(Z,Y)$

We now need to define m_{par}. Intuitively, in presence of incomplete information about parents we can say "e may be a parent of f if we do not know for sure that e is not a parent of f." This can be written as the rule:

$m_{par}(X,Y) \leftarrow not\ \neg par(X,Y)$

Putting all the above rules together we have an interpolation of Π_0 denoted by $\mathcal{A}(\Pi_0)$.

$$\left.\begin{array}{l} m_{par}(X,Y) \leftarrow not\ \neg par(X,Y) \\ m_{anc}(X,Y) \leftarrow m_{par}(X,Y) \\ m_{anc}(X,Y) \leftarrow m_{par}(X,Z), m_{anc}(Z,Y) \\ \neg anc(X,Y) \leftarrow not\ m_{anc}(X,Y) \\ anc(X,Y) \leftarrow par(X,Y) \\ anc(X,Y) \leftarrow par(X,Z), anc(Z,Y) \end{array}\right\} \mathcal{A}(\Pi_0)$$

where $\mathcal{P}$ and $\mathcal{V}$ are the same as for the program Π_0.

Theorem 4.1 (Interpolation) $\mathcal{A}(\Pi_0)$ interpolates Π_0 with
$\mathcal{P} = Lit(par)$ and $\mathcal{V} = Lit(anc)$. □

Since we were using the ancestor query as an example, the above theorem is a special case of a more general theorem (Theorem 4.4) to be stated later. Nevertheless, a separate proof of this theorem is important as it will illustrate proof techniques that can be used for proving interpolation results for more restricted class of general logic programs such as, the class of Horn logic programs. Hence, a separate proof of this theorem is given in the Appendix.

4.2 The Interpolation Algorithm

Now we apply the idea from section 4.1 to arbitrary general logic programs. We expand the language of such a program Π by introducing intermediate predicate m_p for every predicate p in Π. Intuitively $m_p(\bar{t})$ means that $p(\bar{t})$ may be true.

To define these predicates we consider two cases. Since negative information about EDB predicates is explicitly given, this intuition is captured by the rule:

$$m_q(\bar{t}) \leftarrow not\ \neg q(\bar{t}) \tag{v1}$$

The definition of m_a for an IDB predicate a can not rely on negative information about a. Instead it uses the corresponding definition of a from Π. More precisely, $\mathcal{A}(\Pi)$ contains the rule

$$m_a(\bar{t}) \leftarrow m_{b_1}(\bar{t}), \ldots, m_{b_n}(\bar{t}), not\ c_1(\bar{t}), \ldots, not\ c_n(\bar{t}) \tag{v2}$$

for every rule

$$a(\bar{t}) \leftarrow b_1(\bar{t}), \ldots, b_m(\bar{t}), not\ c_1(\bar{t}), \ldots, not\ c_n(\bar{t}) \tag{v3}$$

in Π.

For any program Π, the program $\mathcal{A}(\Pi)$ contains the rule

$$a(\bar{t}) \leftarrow b_1(\bar{t}), \ldots, b_m(\bar{t}), \neg c_1(\bar{t}), \ldots, \neg c_n(\bar{t}) \tag{v4}$$

for every rule (v3) in Π. Rule (v4) is intended to be a monotonic and weakened version of rule (v2). The intuition is that we no longer want to make hasty (revisable) conclusion using the negation as failure operator (*not*) in the body. The negation as failure operator is therefore replaced by $\neg$ in (v4). On the other hand we would like $m_a(\bar{t})$ to be true if there is any possibility that $a(\bar{t})$ is true. Hence, in the body of (v2) we still have the negation as failure operator. Once the truth of $a(\bar{t})$ is established by $\mathcal{A}(\Pi)$ through the rule (v4) it will never be retracted in presence of additional consistent information about the EDB predicates.

The above definitions ensure that the interpolation does not entail $m_p(\bar{t})$ for an input X iff it does not entail $p(\bar{t})$ for any consistent extension Y of X. Hence it is safe to infer $\neg p(\bar{t})$ when the interpolation does not entail $m_p(\bar{t})$. This observation leads to the following definition of falsity of IDB predicates

$$\neg p(\bar{t}) \leftarrow not\ m_p(\bar{t}). \tag{v5}$$

The above intuitions can be summarized into the following algorithm that constructs the interpolations of a large class of general logic programs.

Algorithm 4.1 *For any general logic programs* Π *the extended logic program* $\mathcal{A}(\Pi)$ *contains the following rules:*
1. If q *is an EDB in the program* Π, $\mathcal{A}(\Pi)$ *contains the rule:*

$m_q(\bar{t}) \leftarrow not\ \neg q(\bar{t})$.

2. *For any rule* $a(\bar{t}) \leftarrow b_1(\bar{t}), \ldots, b_m(\bar{t}), not\ c_1(\bar{t}), \ldots, not\ c_n(\bar{t})$ *in* Π, $\mathcal{A}(\Pi)$ *contains the rules:*
   (a) $m_a(\bar{t}) \leftarrow m_{b_1}(\bar{t}), \ldots, m_{b_m}(\bar{t}), not\ c_1(\bar{t}), \ldots, not\ c_n(\bar{t})$,
   (b) $a(\bar{t}) \leftarrow b_1(\bar{t}), \ldots, b_m(\bar{t}), \neg c_1(\bar{t}), \ldots, \neg c_n(\bar{t})$.
3. *If* p *is an IDB in the program* Π, $\mathcal{A}(\Pi)$ *contains the rule:*
   $\neg p(\bar{t}) \leftarrow not\ m_p(\bar{t})$.
4. *Nothing else is in* $\mathcal{A}(\Pi)$. □

Example 4.1 (Transitive Closure) Consider Π_0 from the previous subsection:
$anc(X, Y) \leftarrow par(X, Y)$
$anc(X, Y) \leftarrow par(X, Z), anc(Z, Y)$
Then the transformation $\mathcal{A}(\Pi)$ based on the above algorithm is exactly the extended logic program $\mathcal{A}(\Pi_0)$ obtained in Section 4.1. □

4.3 Properties of the transformation $\mathcal{A}$

In this section we will formalize (and later prove them in the Appendix) several properties of the transformation $\mathcal{A}$. We will show that for a large class of general logic programs it constructs an interpolation and for all general logic programs it constructs a sound interpolation.

But before that we would like to point out that our algorithm will not generate an interpolation for the program Π_1 in Section 1 about birds and flying, for the given parameter and value. In Section 1 for the program Π we are interested in $\mathcal{P} = Lit(b, p)$ and $\mathcal{V} = Lit(f)$. But Algorithm 4.1 requires that $\mathcal{P} = Lit(p)$ and $\mathcal{V} = Lit(ab, b, f)$.

Also, Algorithm 4.1 requires that $\mathcal{P}$ and $\mathcal{V}$ do not intersect and $\mathcal{P} \cup \mathcal{V} = Lit$. At first glance this seems to be very restrictive. In reality, the restriction corresponds to the requirement about EDB and IDB predicates in databases [Ull89], that EDB and IDB predicates be disjoint, and EDB predicates not appear in the head of rules with non-empty bodies. Also, it is well known that in most cases if an EDB predicate appears in the head of the rule with non-empty body then it can be transformed to an intuitively equivalent program with new predicates.

So to be able to apply our transformation algorithm to Π_1 we consider $\mathcal{P} = Lit(b, p)$ and $\mathcal{V} = Lit(f, ab)$. The only change we have made so far is adding $Lit(ab)$ to $\mathcal{V}$. From the definition of interpolation it is clear that if we construct an interpolation w.r.t. $\mathcal{P} = Lit(b, p)$ and $\mathcal{V} = Lit(f, ab)$, we will have an interpolation w.r.t. $\mathcal{P} = Lit(b, p)$ and $\mathcal{V} = Lit(f)$. Now to satisfy the requirement that predicate names in $\mathcal{P}$ not appear in the head of rules with non empty bodies, we consider the following transformation of Π_1, where we introduce a new predicate b_{new}, replace b by b_{new} in all rules Π_1 and then add $b_{new}(X) \leftarrow b(X)$ to Π_1. The resultant program is:

$$\left.\begin{array}{l} f(X) \leftarrow b_{new}(X), not\ \ ab(X) \\ b_{new}(X) \leftarrow p(X) \\ ab(X) \leftarrow p(X) \\ b_{new}(X) \leftarrow b(X) \end{array}\right\} \Pi_{1*}$$

We can now use Algorithm 4.1 to transform (Note that we have not said if the algorithm creates an interpolation or not.) Π_{1*} w.r.t. $\mathcal{P} = Lit(b, p)$, and $\mathcal{V} = Lit(b_{new}, f, ab)$

The results about the transformation $\mathcal{A}$ that we will be proving are intuitively as follows:

(a) If Π is a signed [Kun89] general logic program, then

1. extended logic program $\mathcal{A}(\Pi)$ interpolates program Π (Theorem 4.4), and
2. for any $X \subseteq \mathcal{P}$ program $\Pi \cup atoms(X)$ has a unique stable model if and only if program $\mathcal{A}(\Pi) \cup X$ has a unique answer set. (Theorem 4.5)

(b) We will give an example of a non-signed general logic program Π such that $\mathcal{A}(\Pi)$ does not interpolate program Π. (Example 4.2)

(c) If Π is a stratified program, then both Equivalence and Monotonicity (see Proposition 3.1) hold and $\mathcal{A}(\Pi)$ is a weak interpolation of Π. (Theorem 4.3)

(d) If Π is any general logic program, then $\mathcal{A}(\Pi)$ is a sound interpolation of Π. (Theorem 4.2)

Theorem 4.2 (Properties of $\mathcal{A}(\Pi)$ for an arbitrary general program Π) Let Π be a general logic program and a natural representation of a query Q, then $\mathcal{A}(\Pi)$ is a sound interpolation of Π. □

Theorem 4.3 (Properties of $\mathcal{A}(\Pi)$ for a stratified general program Π) Let Π be a stratified general logic program and a natural representation of a query Q, then $\mathcal{A}(\Pi)$ is a weak interpolation of Π. □

We now present the result where we state the class of programs for which $\mathcal{A}(\Pi)$ is an interpolation of Π. This class is referred to as signed [Kun89]. Intuitively, a signed general program is a program whose predicates can be partitioned to two sets such that for any rule the predicate in the head and the predicates in the body that are not preceded by *not* belong to the same partition and the predicate in the head and the predicates in the body that are preceded by *not* belong to the opposite partitions.

Theorem 4.4 (Properties of $\mathcal{A}(\Pi)$ for a signed program Π) For any signed general logic program Π that is a natural representation of a query Q, $\mathcal{A}(\Pi)$ is an interpolation of Π. □

The programs Π_1 and Π_{1*} are not signed but are stratified. Hence, $\mathcal{A}(\Pi_{1*})$ may not be an interpolation of Π_{1*}. But it is a weak interpolation of Π_{1*}. We believe that it is an interpolation but are unable to prove it. Neither do we have interpolation result about a more general class than the class of signed general logic programs. Nevertheless, the following example shows an unsigned but stratified general logic program for which Algorithm 4.1 does not construct an interpolation.

Example 4.2 Consider the program Π_6:

$$\left.\begin{array}{l} p \leftarrow q \\ p \leftarrow not\ q \end{array}\right\} \Pi_6$$

Here $\mathcal{V} = \{p, \neg p\}, \mathcal{P} = \{q, \neg q\}$. It is easy to see that for any $X \subseteq \mathcal{P}$ $\quad p \in \Pi(X)$.
Program $\mathcal{A}(\Pi_6)$ consists of the following rules:

$$\left.\begin{array}{l} m_q \leftarrow not\ \neg q \\ m_p \leftarrow m_q \\ m_p \leftarrow not\ q \\ p \leftarrow q \\ p \leftarrow \neg q \\ \neg p \leftarrow not\ m_p \end{array}\right\} \mathcal{A}(\Pi_6) = T$$

Consider $X = \emptyset$ and the extended logic program $T \cup X = T$. It is easy to see that T^+ is a stratified program, therefore T has only one answer set A. Evidently, $A = \{m_p, m_q\}$ and $p \notin A$. Therefore

$$\mathcal{A}(\Pi)(X) \neq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$$

□

The following theorem proves that the transformation $\mathcal{A}$ preserves categoricity for signed general logic programs.

Theorem 4.5 (Categoricity) Let Π be a signed general logic program with signing R, a natural representation of a query Q, and $X \subseteq \mathcal{P}$. Program $\Pi \cup atoms(X)$ is categorical if and only if program $\mathcal{A}(\Pi) \cup X$ is categorical. □

In summary, the transformation $\mathcal{A}$ constructs interpolations for signed general logic programs, weak interpolations for stratified general logic programs and sound interpolations for all programs. Moreover, the transformation preserved categoricity for signed programs and from Lemma 6.7 in the Appendix preserves stratification (i.e. If Π is stratified then $\mathcal{A}(\Pi)^+$ is stratified) for stratified programs.

5 Conclusion and Future Work

Reasoning with incomplete information has been looked at by researchers from many different perspectives: from closed world assumption where information not known to be true was automatically assumed to false [Cla78], to various kinds of null values in databases [Gra77, Zan84], to reasoning with disjunctive information in disjunctive logic programming [LMR92], to extended logic programs and epistemic specifications [GL91, Gel91]. In this paper we concentrated on a particular kind of incompleteness where we know a set of tuples about certain relations that are true, a set of tuples that are false, and do not know about the rest of the tuples. Our goal has been to reason with this in such a manner that our conclusions are not made false in presence of additional information about the previously unknown tuples.

We motivated our goal through two examples, one from database and another from knowledge representation. Another common example that comes to our mind is from the field of 'reasoning about actions'. Early work on reasoning about actions such as STRIPS [FN71] assumed complete information about the initial state. In recent years some researchers have considered the case when the initial state is incomplete from the perspective of this paper. i.e. We know certain fluents to be true in the initial situation, certain fluents to be false in the situation, and about the rest we have no idea. Now if we would like to reason with such an incomplete information, we better be careful and make conclusions that does not have to be withdrawn in presence of additional information about the fluents whose truth value in the initial state was previously unknown. For this reason Gelfond and Lifschitz [GL93] translate statements of the form

a **causes** f **if** $p_1, \ldots, p_n$

in their action description language $\mathcal{A}$ to the rule

$noninertial(f, a, S) \leftarrow not\ \neg holds(p_1, S), \ldots, not\ \neg holds(p_n, S)$

instead of the more straightforward rule:

$noninertial(f, a, S) \leftarrow holds(p_1, S), \ldots, holds(p_n, S)$

so as to be able have $noinertial(f, a, S)$ to be $true$ if there is a chance that all the preconditions may be $true$, even if their truth might not be known. The first rule is necessary when we assume incomplete information about the *holds* predicate.

The flying birds example and the action domain above were previously represented as extended logic programs [GL91]. Although the representation had some common features such as the *not* $\neg$ in the body of some rules, the exact mechanism behind them was not studied.

In this paper we took a significant step towards studying this mechanism.

• We introduced the concept of an extended query and argued the appropriateness of using extended logic programs to express extended queries.

• We defined what it means to expand the query represented by a general logic program so as to allow incomplete database instances.

• We related (through an example) query expansion to removal of CWA.

• We presented an algorithm that takes a general logic program and constructs an extended logic program that expands the query represented by the logic program, for the class of signed programs. This is important because it provides an automatic way to interpolate a large number of existing programs in various domains.

But a lot remains to be done. Our algorithm although produces meaningful (sound interpolations for arbitrary programs and weak interpolations for stratified programs) extended logic programs for all programs, it is not able to (provably) construct interpolations for the flying bird example or the action domains. Overcoming this is an important future work.

With the use of extensions of logic programming in knowledge representation, reasoning about actions and related fields it has become necessary to find lemmas that can be used as building blocks for proving deeper results. Some such lemmas were presented in [GP91, Tur93]. The proofs in this paper are important from this point of view. They demonstrate the usefulness of the results in [GP91, Tur93], and hopefully will help in formalizing and proving other results.

So far in this paper we have used the terms 'query expansion' and 'interpolation' almost synonymously. Interpolating a general logic program Π by an extended logic program T is the same as expanding the query represented by Π by the program T. Our earlier use of the term 'approximation' [BGK93] was unfortunate and caused much confusion. We also earlier used the term 'open counterpart' and have been suggested the term 'open consensus' by one of the anonymous referees. There are at least two other ways of looking at our formalization. One (as suggested by another anonymous referee) is to consider our interpolation as an 'interpolation of a class of general logic programs'; the class obtained by adding any subset of the input atoms to the program. This way of looking at it is very much similar to the 'query expansion' formulation. The other way to look at it is based on the concept of 'elaboration tolerance' [GP96]. Intuitively, an elaboration tolerant representation of a specification requires only a small change in the representation when the specification is modified slightly. With respect to this paper the change in specification that happens during interpolation is the removal of the Close World Assumption with respect to the input, and the change in representation corresponds to the transformation done by the Algorithm 4.1. This notion of elaboration tolerance perhaps explains the difference in representation Π_1 and Π_{1*}.

For the kind of reasoning with incomplete information we are concerned with, besides extended logic programs, abductive logic programs may also be a candidate formalism. An abductive logic program is a triple $< \Pi, A, I >$, where Π is an abductive logic program, A is a set of atoms called *abducibles*, and I is a first-order formula (or a collection of it) referred to as integrity constraints. In [KM90], a semantics of abductive logic programs is given via the notion of *generalized stable model*, which is defined as a stable model of $\Pi \cup E$, which satisfies I, for any subset E of A. An abductive logic program T is said to entail a literal f if f is true in all generalized stable models of T.

In [DDS93] Denecker and De Schreye present an abductive logic program for the action domain that 'correctly' reasons w.r.t. incomplete information about the initial state in the sense that their program is sound and complete w.r.t. the semantics of the action language $\mathcal{A}$. This suggests that their program could possibly be an interpolation of the general logic program that correctly reasons in presence of complete information. This is topic of further study and is an immediate priority with us.

In this paper we studied interpolations of programs viewed as functions from complete databases to complete databases. In many cases, we are interested in queries from complete databases to incomplete databases. One such example [BG94] is the query of 'flying birds where we may also have wounded birds. In this case even when the input is complete the output may be incomplete because for wounded birds the intuitive conclusion is to neither infer that they fly nor infer that they do not fly. Such queries can be represented by extended logic programs. Our notion of interpolation can be generalized to take into account such queries. We are currently working on such a generalization. Another related question that is important is to find features that guarantee that an extended logic program is its own interpolation.

References

[ABW88] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA., 1988.

[AHV95] S. Abiteboul, R. Hall, and V. Vianu, editors. *Foundations of Databases*. Addison Wesley, 1995.

[BG94] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[BGK93] C. Baral, M. Gelfond, and O. Kosheleva. Approximating general logic programs. In *Proceedings of International Logic Programming Symposium*, pages 181–198, 1993.

[Cla78] K. Clark. Negation as failure. In Herve Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[DDS93] M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. In *Proceedings of ILPS 93, Vancouver*, pages 147–164, 1993.

[Dun92] P. Dung. On the relations between stable and well-founded semantics of logic programs. *Theoretical Computer Science*, 105:7–25, 1992.

[Fag90] F. Fages. Consistency of Clark's completion and existence of stable models. Technical Report 90-15, Ecole Normale Superieure, 1990.

[FN71] R. Fikes and N. Nilson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[Gel91] M. Gelfond. Strong introspection. In *Proc. AAAI-91*, pages 386–391, 1991.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. Bowen, editors, *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 579–597, 1990.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In *Joint International Conference and Symposium on Logic Programming.*, pages 559–573, 1992.

[GL93] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17(2,3,4):301–323, 1993.

[GP91] M. Gelfond and H. Przymusińska. Definitions in epistemic specifications. In A. Nerode, W. Marek, and Subramanian V. S., editors, *Logic Programming and Non–monotonic Reasoning: Proc. of the First Int'l Workshop*, pages 245–259, 1991.

[GP96] M. Gelfond and H. Przymusinska. Towards a theory of elaboration tolerance: Logic programming approach. To appear in *Journal of Software Engineering and Knowledge Engineering*, 1995

[Gra77] J. Grant. Null values in a relational database. *Information Processing Letters, 6 (5)*, pages 156–157, October 1977.

[Gra91] G. Grahne. *The problem of incomplete information in relational databases.* Springer Verlag, 1991.

[IL84a] T. Imielinski and W. Lipski. The relational model of data and cylindric algebras. *J. Comput. System Sci.*, 1(28):80–102, 1984.

[IL84b] T. Imielinski and W Lipski. Incomplete information in relational databases. *JACM 31, 4*, pages 761–791, Oct 1984.

[KM90] A. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proc. of ECAI-90*, pages 385–391, 1990.

[Kun89] K. Kunen. Signed data dependencies in logic programs. *Journal of Logic Programming*, 7(3):231–245, 1989.

[LMR92] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of disjunctive logic programming.* The MIT Press, 1992.

[LT94] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proc. of the Eleventh Int'l Conf. on Logic Programming*, pages 23–38, 1994.

[McC86] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.

[MS89] W. Marek and V.S. Subrahmanian. The Relationship Between Stable, Supported, Default and Auto-Epistemic Semantics for General Logic Programs. In G. Levi and M. Martelli, editors, *Proceedings of Sixth International Conference in Logic Programming*, pages 600–620, 1989.

[Nel49] D. Nelson. Constructible falsity. *Journal of Symbolic Logic*, 14:16–26, 1949.

[PW89] D. Pearce and G. Wagner. Reasoning with negative information 1 – strong negation in logic programming. Technical report, Gruppe fur Logic, Wissentheorie and Information, Freie Universitat Berlin, 1989.

[Rei78] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 119–140. Plenum Press, New York, 1978.

[Tur93] H. Turner. A monotonicity theorem for extended logic programs. In D. S. Warren, editor, *Proc. of 10th International Conference on Logic Programming*, pages 567–585, 1993.

[Ull88] J. Ullman. *Principles of database and knowledge-base systems*, volume 1. Computer Science Press, 1988.

[Ull89] J. Ullman. *Principles of database and knowledge-base systems*, volume 2. Computer Science Press, 1989.

[vEK76] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM.*, 23(4):733–742, 1976.

[VG88] A. Van Gelder. Negation as failure using tight derivations for general logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 149–176. Morgan Kaufmann, San Mateo, CA., 1988.

[Zan84] C. Zaniolo. Database relations with null values. *Journal of Computer System Sciences*, 28:142–166, 1984.

6 Appendix

6.1 Results about general and extended logic programs

In this section we state some theorems from the literature that will be used in the proofs later.

Theorem 6.1 *(Gelfond and Przymusinska) [GP91]*
Let $\mathcal{L}_0$ be a language and $\mathcal{L}_1$ be its extension by a set of new predicates. Let T and D be extended logic programs in $\mathcal{L}_0$ and $\mathcal{L}_1$ respectively. If for any rule of the type (2) from D, $l_0 \in \mathcal{L}_1 - \mathcal{L}_0$ and $\forall i, i > 0 \;\; l_i \in \mathcal{L}_0$ then
if no answer set of T satisfies the premises of contrary rules (i.e. rules with contrary literals in the head) from D then for all $l \in \mathcal{L}_0$, $T \models l$ iff $T \cup D \models l$ $\square$

Theorem 6.2 *(Marek and Subrahmanian) [MS89]* For any answer set S of a general logic program Π:

(a) For any ground instance of rule of the type (1) from Π, if
$\{l_1, \ldots, l_m\} \subseteq S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$
then $l_0 \in S$.
(b) If S is a consistent answer set of Π and $l_0 \in S$ then there exist a ground instance of a rule from Π such that
$\{l_1, \ldots, l_m\} \subseteq S$ and $\{l_{m+1}, \ldots, l_n\} \cap S = \emptyset$. □

For a general logic program Π with signing S, let Π_S be the set of all rules whose heads belong to S, and let $\Pi_{\bar{S}}$ be the set of all rules whose heads belong to $\bar{S}$.
Consider rules r and r' of the form (1) without classical negation. We say that $r \preceq r'$ iff $l_0 = l_0{}'$, $\{l_1, \ldots, l_m\} \subseteq \{l_1{}', \ldots, l_{m'}{}'\}$ and $\{l_{m'+1}{}', \ldots, l_{n'}{}'\} \subseteq \{l_{m+1}, \ldots, l_n\}$.
Now we can define partial order $\preceq$ on programs. Given programs P and Q, we say that $P \preceq Q$ iff for every rule r in program P there is a rule r' in program Q such that $r \preceq r'$.

Theorem 6.3 *(Turner) [Tur93]* For extended logic programs P and Q with common signing S, if $P_{\bar{S}} \preceq Q_{\bar{S}}$ and $Q_S \preceq P_S$ and $literals(P) \cap \bar{S} \subset literals(Q) \cap \bar{S}$, then program Q entails every literal in $\bar{S}$ that is entailed by program P. □

For any predicate p occurring in an extended logic program Π, let p' be a new predicate of the same arity. The atom $p'(X_1, \ldots, X_n)$ will be called the *positive form* of the negative literal $\neg p(X_1, \ldots, X_n)$. Every positive literal is, by definition, its own positive form. The positive form of a literal L will be denoted by L^+. Π^+ stands for the general logic program obtained from Π by replacing each rule (2) by

$$l_0^+ \leftarrow l_1^+, \ldots, l_m^+, not\ \ l_{m+1}^+, \ldots, not\ \ l_n^+$$

For any set $S \subseteq Lit$, S^+ stands for the set of the positive forms of the elements of S.

Theorem 6.4 *[GL90]* A consistent set $S \subseteq Lit$ is an answer set of Π if and only if S^+ is a stable model of Π^+. □

Theorem 6.5 *[GL90]* An extended logic program Π is categorical if

(a) Π^+ is stratified, and

(b) The answer set of Π^+ does not contain atoms of the form $p(t)$, $p'(t)$. □

6.2 Proofs of Theorems 3.2 and 3.3

Proposition 3.2 Let T_1 and Π_1 be the logic programs defined in Example 1.2. T_1 is an interpolation of Π_1, for $\mathcal{P} = Lit(p, b)$ and $\mathcal{V} = Lit(f)$. □

Proof:
The proof consist of three parts.
(1) Let Π_1^0 be the general program consisting of rule
(r2) : $b(X) \leftarrow p(X)$
and T_1^0 be the extended program consisting of rules (r2) and
(r2') : $\neg p(X) \leftarrow \neg b(X)$

We show that T_1^0 is an interpolation of Π_1^0 with $\mathcal{P} = \mathcal{V} = Lit(p, b)$.

(2) Let $\tilde{T}_1$ be the program obtained from T_1 by adding $r2'$, removing $r3'$, and replacing $r1$ by $r1'$ with $\mathcal{P} = Lit(p, b)$ and $\mathcal{V} = Lit(f)$.

$$\left.\begin{array}{l} r1'.\ \ f(X) \leftarrow b(X), \neg p(X) \\ r2.\ \ b(X) \leftarrow p(X) \\ r4.\ \ \neg f(X) \leftarrow p(X) \\ r5.\ \ \neg f(X) \leftarrow \neg b(X) \\ r2'.\ \ \neg p(X) \leftarrow \neg b(X) \end{array}\right\} \tilde{T}_1$$

We show that $\tilde{T}_1$ interpolates Π_1.

(3) We show that T_1 and $\tilde{T}_1$ are equivalent w.r.t. inputs from $\mathcal{P}$, i.e. for every $X \subseteq \mathcal{P}$ $T_1(X) = \tilde{T}_1(X)$.

(2) and (3) imply that T_1 interpolates Π_1.

Proof of part (1):

To prove that T_1^0 is an interpolation of Π_1^0 we need to show

$$T_1^0(X) = \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X}). \tag{4}$$

Consider two cases (a) $T_1^0(X)$ is inconsistent and (b) $T_1^0(X)$ is consistent.

(case a) We show that if $T_1^0(X)$ is inconsistent then $S_{\Pi_1^0}(X) = \emptyset$. It is easy to see that $T_1^0(X)$ is inconsistent iff
(a1) $\exists c\ p(c), \neg b(c) \in X$ or
(a2) X is inconsistent.
Consider case (a1). Then for any Y such that $X \subseteq Y$, $b(c) \in \Pi(atoms(Y))$ and hence $Y \neq \Pi(atoms(Y)$, i.e. $Y \notin Dom(\Pi)$. Hence, $S_{\Pi_1^0}(X) = \emptyset$. Similarly, $S_{\Pi_1^0}(X) = \emptyset$ for case (a2).

(case b: $T_1^0(X)$ is consistent.) We will prove that
(b1) $T_1^0(X) \subseteq \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$ and (b2) $T_1^0(X) \supseteq \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$

(b1) Obviously for any atom s, $s \in T_1^0(X)$ implies $s \in \Pi_1^0(X)$ and hence by monotonicity (w.r.t. atoms) of Π_1^0, we have that $s \in \Pi_1^0(\hat{X})$ for any $\hat{X} \in S_{\Pi_1^0}(X)$.
For negative literals the conclusion follows immediately from consistency of $T_1^0(X)$ and Theorem 6.2. Hence, $T_1^0(X) \subseteq \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$.

(b2) Let $l \in \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$. Assume that $l \notin T_1^0(X)$. Consider $X_0 = X \cup \{\bar{l}\}$ (where $\bar{l}$ is the literal contrary to l) and $X^* = {\Pi_1^0}_{atoms(X_0)} \mid \mathcal{P}$.

First we will show that $X^* \in Dom(\Pi_1^0)$.

Since $\Pi_1^0 \cup atoms(X_0)$ has a unique stable model M, X^* is complete. By definition of X^*, $M = atoms(X^*)$. It is easy to show that M is the stable model of $\Pi_1^0 \cup M$, hence $X^* = \Pi^0_{1\,atoms(X^*)} \mid \mathcal{P}$, and therefore
$X^* \in Dom(\Pi_1^0)$. (t1)

Now we will show that $X_0 \subseteq X^*$. Suppose this is not true. Then, there exists a negative literal $\neg s$ such that $\neg s \in X_0$ and $\neg s \notin X^*$. Since, X^* is complete, this means $s \in X^*$. Since Π_1^0 and T_1^0 have the same rules for positive atoms, $s \in T_1^0(X_0)$. Since $\neg s \in X_0$ $T_1^0(X_0)$ is inconsistent.
On the other hand let us recall that we assumed consistency of $T_1^0(X)$. It is not difficult to show that consistency of $T_1^0(X)$ implies consistency of $T_1^0(X_0)$ which leads to a contradiction. Hence,
$X_0 \subseteq X^*$. (t2)

From (t1) and (t2) and since $X \subseteq X_0$, we have $X^* \in S_{\Pi_1^0}(X)$ and hence $l \in \Pi_1^0(X^*)$. Since $\mathcal{P} = \mathcal{V}$, we have $l \in X^*$. But, $\bar{l} \in X_0$ (by definition of X_0) and by (t2), $\bar{l} \in X^*$. Hence, X^* contains both l and $\bar{l}$ which is impossible. Hence, $l \in T_1^0(X)$.

This completes the proof of part (1). □

Proof of part (2)
We need to prove
$$\tilde{T}_1(X) = \bigcap_{\hat{X} \in S(X)} \Pi_1(\hat{X}). \tag{5}$$

(case a: $\tilde{T}_1(X)$ is inconsistent.)
From Theorem 6.1 it follows that $\tilde{T}_1(X)$ is inconsistent iff $T_1^0(X)$ is inconsistent.
It is easy to see that by Theorem 6.1 $Dom(\Pi_1) = Dom(\Pi_1^0)$ and hence
for every $X \subseteq \mathcal{P}$ $\quad S_{\Pi_1}(X) = S_{\Pi_1^0}(X)$. (s)
In case (a) of part (1) we showed that if $T_1^0(X)$ is inconsistent then $S_{\Pi_1^0}(X) = \emptyset$. Hence we can conclude that $S_{\Pi_1}(X) = \emptyset$ and (4) holds.

(case b: $\tilde{T}_1(X)$ is consistent.)

Notice that Π_1^0 and Π_1 are stratified programs and for any $X \subseteq \mathcal{P}$,
$\Pi_1^0 \cup atoms(X)$ and $\Pi_1 \cup atoms(X)$ have unique stable models. (s0)

Using theorems 6.2 and 6.1 we have
$f(c) \in \tilde{T}_1(X)$ iff $b(c) \in T_1^0(X)$ $\;$ *and* $\;$ $\neg p(c) \in T_1^0(X)$. (s1)
$\neg f(c) \in \tilde{T}_1(X)$ iff $\neg b(c) \in T_1^0(X)$ $\;$ *or* $\;$ $p(c) \in T_1^0(X)$ (s2)
$f(c) \in \Pi_1(X)$ iff $b(c) \in \Pi_1^0(X)$ $\;$ *and* $\;$ $\neg p(c) \in \Pi_1^0(X)$ (s3)

From (3) we have
$b(c) \in T^0{}_1(X)$ iff $b(c) \in \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$ (s4)
$\neg p(c) \in T^0{}_1(X)$ iff $\neg p(c) \in \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$ (s5)
$\neg b(c) \in T_1^0(X)$ iff $\neg b(c) \in \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$ (s6)

$p(c) \in T_1^0(X)$ iff $p(c) \in \bigcap_{\hat{X} \in S_{\Pi_1^0}(X)} \Pi_1^0(\hat{X})$ (s7)

We first prove (4) for atoms.
$f(c) \in \tilde{T}_1(X)$
$\Leftrightarrow$ for every $\hat{X} \in S_{\Pi_1^0}(X)$ $b(c) \in \Pi_1^0(X)$ *and* $\neg p(c) \in \Pi_1^0(X)$ (from (s1), (s4) and (s5))
$\Leftrightarrow$ for every $\hat{X} \in S_{\Pi_1}(X)$ $f(c) \in \Pi_1(X)$ (from (s3) and (s))

Now we prove (4) for negative literals.
$\neg f(c) \in \tilde{T}_1(X)$
$\Leftrightarrow$ for every $\hat{X} \in S_{\Pi_1^0}(X)$ $\neg b(c) \in \Pi_1^0(X)$ *or* $p(c) \in \Pi_1^0(X)$ (from (s2), (s6) and (s7))
$\Leftrightarrow$ for every $\hat{X} \in S_{\Pi_1^0}(X)$ $b(c) \notin \Pi_1^0(X)$ *or* $\neg p(c) \notin \Pi_1^0(X)$ (from (s0))
$\Leftrightarrow$ for every $\hat{X} \in S_{\Pi_1}(X)$ $\neg f(c) \in \Pi_1(X)$ (from (s3), (s0) and (s))

This completes the proof of (4). □

Proof of part (3)

(case a) When both $T_1(X)$ and $\tilde{T}_1(X)$ are consistent.

When $T_1(X)$ is consistent it is easy to show (by first transforming it to a general logic program $T_1^+(X)$, and then showing that $T_1^+(X)$ is stratified) that it has an unique answer set. Similarly, for $\tilde{T}_1(X)$.

From Theorem 6.2 we have that for any $X \subseteq \mathcal{P}$ and any constant c

$f(c) \in T_1(X)$ iff $b(c) \in T_1(X)$ and $ab(c) \notin T_1(X)$
iff ($b(c) \in X$ or $p(c) \in X$) and $\neg p(c) \in T_1(X)$
iff ($b(c) \in X$ or $p(c) \in X$) and $\neg p(c) \in X$
iff $b(c) \in X$ and $\neg p(c) \in X$ (Since X must be consistent) (s8)

Similarly, $f(c) \in \tilde{T}_1(X)$ iff $b(c) \in \tilde{T}_1(X)$ and $\neg p(c) \in \tilde{T}_1(X)$
iff ($b(c) \in X$ or $p(c) \in X$) and ($\neg p(c) \in X$ or $\neg b(c) \in X$)
iff ($b(c) \in X$ and $\neg p(c) \in X$) or ($p(c) \in X$ and $\neg b(c) \in X$) (Since X must be consistent)
iff ($b(c) \in X$ and $\neg p(c) \in X$) (Since $p(c) \in X$ and $\neg b(c) \in X$ makes $\tilde{T}_1(X)$ inconsistent. (s9)

Also, $\neg f(c) \in T_1(X)$ iff $p(c) \in T_1(X)$ or $\neg b(c) \in T_1(X)$
iff $p(c) \in X$ or $\neg b(c) \in X$
iff $p(c) \in \tilde{T}_1(X)$ or $\neg b(c) \in \tilde{T}_1(X)$
iff $\neg f(c) \in \tilde{T}_1(X)$ (s10)

(case b) We now show that $T_1(X)$ is inconsistent iff $\tilde{T}_1(X)$ is inconsistent.

From Theorem 6.1 it follows that $\tilde{T}_1(X)$ is inconsistent iff $T_1^0(X)$ is inconsistent.
Recall from proof of part (1) that $T_1^0(X)$ is inconsistent iff
(a1) $\exists c\ p(c), \neg b(c) \in X$ or
(a2) X is inconsistent.

Consider $(T_1(X))^+$. Following derivations similar to in (s8) and (s10) it is easy to see that $f(c)$ and $f(c)'$ are in $(T_1(X))^+$ iff X is inconsistent. T_1 does not have rules with either p or $\neg p$ in the head, and the predicate ab is not included in $\mathcal{P}$. Hence, when X is consistent the only way $(T_1(X))^+$ will contain opposite literals is when for some c, $p(c), \neg b(c) \in X$.

Hence, $T_1(X)$ is inconsistent iff $T_1^0(X)$ is inconsistent iff $\tilde{T}_1(X)$ is inconsistent.

From (case a) and (case b) we have $\forall X$ If $X \subseteq \mathcal{P}$ then $T_1(X) = \tilde{T}_1(X)$ $\Box$

This completes the proof of the proposition. $\Box$

Proposition 3.3 T_2 is an interpolation of Π_2. $\Box$

Proof

We will prove that

$$\forall X \subseteq \mathcal{P}, T_2(X) = \bigcap_{\hat{X} \in S_{\Pi_2}(X)} \Pi_2(\hat{X}) \tag{6}$$

Let us denote the right hand side of the above equation by N.

It is easy to show that $T_2(X)$ is consistent for any consistent $X \subseteq \mathcal{P}$. This together with the fact that $\Pi_2 \cup atoms(Y)$ is consistent for any Y implies that if $T_2(X)$ is inconsistent then $S_{\Pi_2}(X)$ is empty and hence (5) holds for inconsistent $T_2(X)$.

Suppose $T_2(X)$ is consistent. Let us first show that for any constant a,
$c(a) \in N$ iff $r(a) \in X$ iff $c(a) \in T_2(X)$. (t1)
Suppose $c(a) \in N$ and $r(a) \notin X$. Let $\hat{X} = atoms(X) \cup \{\neg r(c) : r(c) \notin X\}$. Obviously, $\hat{X} \in Dom(\Pi_2)$ and hence $\hat{X} \in S_{\Pi_2}(X)$. Since $\Pi_2 \cup atoms(\hat{X})$ is consistent, by Theorem 6.2 we have $c(a) \notin \Pi_2(\hat{X})$, which contradicts our assumption. Therefore, $r(a) \in X$. The rest of (t1) follows from the consistency of our programs and Theorem 6.2.

We now show that for any constant a,
$\neg c(a) \in N$ iff $\neg r(a) \in X$ iff $\neg c(a) \in T_2(X)$ (t2)
Suppose $\neg c(a) \in N$ and $\neg r(a) \notin X$. Let $\hat{X} = atoms(X) \cup \{r(a)\} \cup \{\neg r(c) : r(c) \notin \{X \cup r(a)\}\}$. It is easy to show that $\hat{X} \in S_{\Pi_2}(X)$. Clearly $\neg c(a) \notin \Pi_2(\hat{X})$, which contradicts our assumption. Therefore, $\neg r(a) \in X$. The rest of (t2) again follows from the consistency of our programs and Theorem 6.2.

This completes the proof of Proposition 3.3. $\Box$

Theorem 4.1 (Interpolation) $\mathcal{A}(\Pi_0)$ interpolates Π_0 with
$\mathcal{P} = Lit(par)$ and $\mathcal{V} = Lit(anc)$. $\Box$

Proof:

Let us denote $\mathcal{A}(\Pi_0)$ by T_0 and show that

$$T_0(X) = \bigcap_{\hat{X} \in S(X)} \Pi_0(\hat{X}). \tag{7}$$

Like before we consider two cases. (a) $T_0(X)$ is inconsistent and (b) $T_0(X)$ is consistent.

It is easy to show that $T_0(X)$ is consistent for any consistent X. This together with the fact that $\Pi_0 \cup atoms(Y)$ is consistent for any $Y \subseteq \mathcal{P}$ implies that if $T_0(X)$ is inconsistent then $S_{\Pi_0}(X)$ is empty and hence (7) holds for inconsistent $T_0(X)$.

Suppose $T_0(X)$ is consistent.
"$\Longrightarrow$"
Since, T_0 and Π_0 have the same rules defining anc, it is easy to show that $anc(a,b) \in T_0(X)$ implies that $anc(a,b) \in \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$.
Now, let $\neg anc(a,b) \in T_0(X)$ and $\neg anc(a,b) \notin \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$.
Consider $X^* \in S_{\Pi_0}(X)$ such that, $\neg anc(a,b) \notin \Pi_0(X^*)$ (u1)
$\Pi_0 \cup\ atoms(X^*) \models par(a,b) \Rightarrow par(a,b) \in X^* \Rightarrow \neg par(a,b) \notin X^* \Rightarrow$
$\neg par(a,b) \notin X \Rightarrow T_0 \cup X \models m_{par}(a,b)$ (u2)
Now, from (u1) we have $\neg anc(a,b) \notin \Pi_0(X^*)$
$\Rightarrow anc(a,b) \in \Pi_0(X^*)$
$\Rightarrow\ \exists c_1, \ldots, c_n$ such that $\Pi_0 \cup\ atoms(X^*) \models \{par(a,c_1), par(c_1,c_2), \ldots, par(c_n,b)\}$ (can be shown using fixpoint construction from [vEK76])
$\Rightarrow T_0 \cup X \models \{m_{par}(a,c_1), m_{par}(c_1,c_2), \ldots, m_{par}(c_n,b)\}$ (from (u2))
$\Rightarrow T_0 \cup X \models m_{anc}(a,b)$
$\Rightarrow \neg anc(a,b) \notin T_0(X)$
This contradicts our original assumption, and hence
$\neg anc(a,b) \in \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$.

"$\Longleftarrow$"
Let $anc(a,b) \in \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$ and $anc(a,b) \notin T_0(X)$.
Let $X^* = X \cup \{\neg par(a,b) : par(a,b) \notin X\}$. It is easy to show that $X^* \in S_{\Pi_0}(X)$. Hence, by our initial assumption $anc(a,b) \in \Pi_0(X^*)$
$\Rightarrow \Pi_0 \cup\ atoms(X^*) \models anc(a,b)$
$\Rightarrow T_0 \cup\ atoms(X^*) \models anc(a,b)$
$\Rightarrow T_0 \cup\ atoms(X) \models anc(a,b)$ (Since, $atoms(X) = atoms(X^*)$
$\Rightarrow anc(a,b) \in T_0(X)$.
This contradicts our original assumption, and hence $anc(a,b) \in T_0(X)$.

Let $\neg anc(a,b) \in \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$ and $\neg anc(a,b) \notin T_0(X)$.
$\neg anc(a,b) \notin T_0(X) \Rightarrow T_0 \cup X \models m_{anc}(a,b)$ (from Theorem 6.2)
$\Rightarrow \exists c_1, \ldots, c_n$ such that $T_0 \cup X \models \{m_{par}(a,c_1), m_{par}(c_1,c_2), \ldots, m_{par}(c_n,b)\}$
$\Rightarrow\ \{\neg par(a,c_1), \neg par(c_1,c_2), \ldots, \neg par(c_n,b)\} \cap X = \emptyset$ (from Theorem 6.2)
Let $X^* = X \cup \{par(a,b) : \neg par(a,b) \notin X\}$. It is easy to see that $X^* \in S_{\Pi_0}(X)$ and $\{par(a,c_1), par(c_1,c_2), \ldots, par(c_n,b)\} \subseteq X^*$. Hence, $anc(a,b) \in \Pi_0(X^*)$. This implies that $\neg anc(a,b) \notin \bigcap_{\hat{X} \in S_{\Pi_0}(X)} \Pi_0(\hat{X})$, which contradicts our original assumption and hence, $\neg anc(a,b) \in T_0(X)$.

This completes the proof of Theorem 4.1. □

4.3 Proof of the main result

In the proofs we will use the following notations:
for a general logic program Π

$s(\Pi)$ will denote the set of all stable models of Π;

Π_X will denote the program $\Pi \cup atoms(X)$

for an extended logic program T

$a(T)$ will denote the set of all answer sets of T;

T_X will denote the program $T \cup X$.

(The confusion between the set of literals Π_X, and T_X and the programs Π_X and T_X respectively will be clear from the context.)

For a set of atoms U we denote the set $\{\neg p : p \notin U\}$ by $\widetilde{U}$.
The answer set semantics treats a rule with variables as a shorthand for the set of its ground instances. Therefore everywhere in the proofs we will assume that all rules have only ground literals in them, i.e. we will consider propositional programs.

In the proofs we will use the following definitions.

Lemma 6.1 For any general logic program Π and any $X \subseteq \mathcal{P}$,

(1) if program Π_X is coherent, then $X \in Dom(\Pi)$ if and only if X is consistent and complete w.r.t. $\mathcal{P}$;

(2) if program Π_X is non-coherent, then $X \in Dom(\Pi)$ if and only if $X = Lit(\mathcal{P})$.

□

Proof: (1) Consider the case when program Π_X is coherent.
By definition

$$\Pi_{atoms(X)} \mid \mathcal{P} = \bigcap_{M \in S(\Pi_X)} (M \cap \mathcal{P}) \cup \bigcap_{M \in S(\Pi_X)} (\widetilde{M} \cap \mathcal{P}).$$

Since program Π does not have rule with the head from $\mathcal{P}$, for any $M \in S(\Pi_X)$ $M \cap \mathcal{P} = atoms(X)$, $\widetilde{M} \cap \mathcal{P} = \widetilde{atoms}(X)$ and, hence, $\Pi_{atoms(X)} \mid \mathcal{P}) = atoms(X) \cup \widetilde{atoms}(X)$.

Therefore $X \in Dom(\Pi)$
iff X is complete w.r.t. $\mathcal{P}$ and $\Pi_{atoms(X)} \mid \mathcal{P} = X$
iff X is complete w.r.t. $\mathcal{P}$ and $X = atoms(X) \cup \widetilde{atoms}(X)$
iff X is consistent and complete w.r.t. $\mathcal{P}$. □

(2) Consider the case when program Π_X is non-coherent.
Since $S(\Pi_X) = \emptyset$

$$\Pi_{atoms(X)} \mid \mathcal{P}) = \bigcap_{M \in S(\Pi_X)} (M \cap \mathcal{P}) \cup \bigcap_{M \in S(\Pi_X)} (\widetilde{M} \cap \mathcal{P}) = Lit.$$

Therefore $X \in Dom(\Pi)$
iff X is complete w.r.t. $\mathcal{P}$ and $\Pi_{atoms(X)} \mid \mathcal{P} = X$
iff $X = Lit$. □

Lemma 6.2 Let Π be a general logic program, $X \subseteq \mathcal{P}$ and A is a consistent answer set of $\mathcal{A}(\Pi)_X$. Then for any atom p

(a) $\neg p \in A$ iff $m_p \notin A$.

(b) if $p \in A$ then $m_p \in A$;

$\square$

Proof: Let $T = \mathcal{A}(\Pi)$.
First, we will prove (a). Let $p \in \mathcal{P}$. Since any rule from T has a head from $\mathcal{V}$ $A \cap \mathcal{P} = X$. Therefore $\neg p \in A$ iff $\neg p \in X$.
Consider program T^+. A^+ is a stable model of $(T_X)^+$. Since for every $p \in \mathcal{P}$ there is a rule

$$m_p \leftarrow not \;\; p'$$

in T^+ and there are no other rules with m_p in the head, from Theorem 2.3 (b) follows that for $p \in \mathcal{P}$, $p' \in A^+$ iff $m_p \notin A^+$. Since A is a consistent set we can conclude that $\neg\, p \;\in\; A$ iff $m_p \;\notin\; A$.

Consider, now, $p \in \mathcal{V}$. Since for every $p \in \mathcal{V}$ there is a rule

$$p' \leftarrow not \;\; m_p$$

in T^+ and there are no other rules with p' in the head, from Theorem 2.3 (b) follows that for $p \in \mathcal{V}$, $p' \in A^+$ iff $m_p \notin A^+$. Since A is a consistent set we can conclude that $\neg\, p \;\in\; A$ iff $m_p \;\notin\; A$.

Now let us prove (b). Since A is consistent, if $p \in A$, then $\neg p \notin A$. Using (a) we conclude that $m_p \in A$. $\square$

Lemma 6.3 For any general logic program Π and any $X \in Dom(\Pi)$, if M is a stable model of Π_X then the set $A \;=\; M \;\cup\; \widetilde{M} \;\cup\; \{m_c : c \in M\}$ is an answer set of $\mathcal{A}(\Pi) \cup X$. $\square$

Proof: Let us denote $\mathcal{A}(\Pi)$ by T.
We will now show that A is the unique answer set of $(T \cup X)^A$.
We divide all the rules from the program $T \cup X$ into three groups:

T_p - all the rules from $T \cup X$, whose heads are positive atoms from the language of Π. These rules come

1. either from step (2.b) of the Algorithm 4.1
2. or from X, as a rule $p \leftarrow$, where $p \in atoms(X)$;

T_n - all the rules from $T \cup X$, whose heads are negative literals from the language of Π. These rules come

1. either from step (3.) of the Algorithm 4.1
2. or from X, as a rule $\neg p \leftarrow$, where $\neg p \in X$;

T_m - all the rules from $T \cup X$, whose heads are of the form m_p. These rules come

1. from steps (1.),(2.a) of the Algorithm 4.1.

Let $O = (T \cup X)^A = {T_p}^A \cup {T_n}^A \cup {T_m}^A = R_p \cup R_n \cup R_m$,
where $R_p = {T_p}^A$, $R_n = {T_n}^A$, $R_m = {T_m}^A$. Let us analyze these reducts one by one.
Rules, generated by the step (2.b) do not contain negation as failure. Therefore $R_p = {T_p}^A = T_p$ and it consists of

$$\left.\begin{array}{l} 1.\ \textit{all the rules of the form} \\ \quad p \leftarrow \\ \ \ \textit{where } p \in atoms(X) \textit{ and} \\ 2.\ \textit{all the rules of the form} \\ \quad t \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_n \\ \ \ \textit{from program } T. \end{array}\right\} R_p$$

Part T_n has negation as failure only in the rules, generated by the step (3.). They have the form

$$\neg c \leftarrow not\ m_c$$

where $c \in \mathcal{V}$. As follows from the definition of A, $m_c \notin A$ iff $c \notin M$.
Therefore R_n consists of

$$\left.\begin{array}{l} 1.\ \textit{all the rules of the form} \\ \quad \neg c \leftarrow \\ \ \ \textit{where } c \in \mathcal{V}, c \notin M \textit{ and} \\ 2.\ \textit{all the rules of the form} \\ \quad \neg p \leftarrow \\ \ \ \textit{where } \neg p \in X. \end{array}\right\} R_n$$

We will now show that $\neg p \in X$ iff $p \in \mathcal{P}$ and $p \notin M$.
Let p be an atom from $\mathcal{P}$. Since $X \in Dom(\Pi)$ and program Π_X is coherent, by Lemma 6.1, we have that X is consistent and complete w.r.t. $\mathcal{P}$. Therefore $\neg p \in X$ iff $p \notin X$.
Since M is a stable model of Π_X and no rules with the heads from $\mathcal{P}$ belong to Π, $p \in X$ iff $p \in M$.
Therefore $\neg p \in X$ iff $p \notin M$.
Thus, we can conclude that $R_n = \{\neg c \leftarrow \ : c \notin M\}$.

Part T_m consists of

$$\left.\begin{array}{l} 1.\ \textit{all the rules of the form} \\ \quad m_p \leftarrow not\ \neg p \\ \ \ \textit{where } p \in \mathcal{P} \textit{ and} \\ 2.\ \textit{all the rules of the form} \\ \quad m_t \leftarrow m_{b_1}, \ldots, m_{b_n}, not\ c_1, \ldots, not\ c_k \\ \ \ \textit{from program } T. \end{array}\right\} T_m$$

For any atom $p \in \mathcal{P}$, $\neg p \notin A$ iff $p \in M$ iff $p \in atoms(X)$. Hence, R_m consists of

$$\left.\begin{array}{l} 1.\ \textit{all the rules of the form} \\ \quad m_p \leftarrow \\ \ \ \textit{where } p \in atoms(X) \textit{ and} \\ 2.\ \textit{all the rules of the form} \\ \quad m_t \leftarrow m_{b_1}, \ldots, m_{b_n} \\ \ \ \textit{such that there is a rule} \\ \quad m_t \leftarrow m_{b_1}, \ldots, m_{b_n}, not\ c_1, \ldots, not\ c_k \\ \ \ \textit{in } T_m \textit{ and } \forall j, 1 \leq j \leq k\ \ c_j \notin A \\ \ \ (\textit{which is equivalent to } c_j \notin M). \end{array}\right\} R_m$$

By definition, $O = (R_p \cup R_n) \cup R_m$. It is easy to see, that $language(R_p \cup R_n) \cap language(R_m) = \emptyset$ and, therefore the answer set of the program O equals to the union of the answer set of the program $R_p \cup R_n$ and the answer set of the program R_m.

Consider R_{p0}, obtained from R_p by

$$\left.\begin{array}{l} \text{1. } \textit{removing from } R_p \textit{ every rule of the form} \\ \quad t \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_k, \\ \quad \textit{which contains } \neg c_i \textit{ such that } c_i \in M \textit{ and} \\ \text{2. } \textit{removing all other occurences of} \\ \quad \textit{negative literals from the rules.} \end{array}\right\} R_{p0}$$

It is easy to see that S is the answer set of the program $R_p \cup R_n$ iff S is the answer set of $R_{p0} \cup \{\neg c \leftarrow \ : c \notin M\}$.

Now we will compute the answer sets of the programs $R_p \cup R_n$ and R_m.

Using the definition of R_{p0} it is easy to check that $R_{p0} = (\Pi_X)^M$, and, hence, the set $M \cup \widetilde{M}$ is the answer set of $R_p \cup R_n$.

Consider program R_m and let us rename every atom m_p by p. We will call the resulting program $ren(R_m)$.
It is easy to check that $ren(R_m) = (\Pi_X)^M$, and, hence, M is the answer set of the program $ren(R_m)$. Therefore the answer set of the program R_m is equal to the set $\{m_c : c \in M\}$.

Thus, we proved that the answer set of the program $(T \cup X)^A$ is equal to the set $M \cup \widetilde{M} \cup \{m_c : c \in M\} = A$. $\square$

Lemma 6.4 For any general logic program Π and any $X \in Dom(\Pi)$

$$\mathcal{A}(\Pi)(X) \subseteq \Pi(X)$$

$\square$

Proof: Let $T = \mathcal{A}(\Pi)$. By definition,

$$\Pi(X) = \bigcap_{M \in S(\Pi_X)} (M \cap \mathcal{V}) \cup \bigcap_{M \in S(\Pi_X)} (\widetilde{M} \cap \mathcal{V})$$

$$T(X) = \bigcap_{A \in a(T_X)} A \cap \mathcal{V}.$$

If $S(\Pi_X) = \emptyset$, then $T(X) \subseteq \Pi(X) = Lit(\mathcal{V})$ and the inclusion is true.

Consider now the case,when $S(\Pi_X) \neq \emptyset$.
(1) Suppose atom $p \in T(X)$, but $p \notin \Pi(X)$. This means that there is $M_0 \in S(\Pi_X)$ such that $p \notin M_0$. Then, using Lemma 6.3, we can build the answer set A_0 such that $p \notin A_0$, therefore $p \notin T(X)$. This contradiction proves that $p \in \Pi(X)$.

(2) Suppose literal $\neg p \in T(X)$, but $\neg p \notin \Pi(X)$. This means that there is $M_0 \in S(\Pi_X)$ such that $p \in M_0$. Then, using Lemma 6.3, we can built the answer set A_0 such that $p \in A_0$, therefore $\neg p \notin T(X)$. This contradiction proves that $\neg p \in \Pi(X)$. $\square$

Lemma 6.5 (Monotonicity) Let Π be a general logic program and $X_1 \subseteq X_2 \subseteq \mathcal{P}$. Then $\mathcal{A}(\Pi)(X_1) \subseteq \mathcal{A}(\Pi)(X_2)$. $\square$

Proof: Let us denote $\mathcal{A}(\Pi)$ by T. Consider $MB = \{m_p : p \in \mathcal{L}_\Pi\}$. It is easy to see, that this set is a signing for $T \cup X$ for every $X \subseteq \mathcal{P}$. Consider programs $P = T \cup X_1$ and $Q = T \cup X_2$. Set MB is their common signing and, as easily can be seen, the conditions of the Theorem 2.4 [Tur93] are satisfied.
Indeed, $literals(P) \cap \overline{MB} \subseteq literals(Q) \cap \overline{MB}$, $Q_{MB} = P_{MB}$ and $P_{\overline{MB}} \subseteq Q_{\overline{MB}}$.
Therefore we can conclude, that program Q entails every literal in $\overline{MB}$, that is entailed by P. In particular, $T(X_1) \subseteq T(X_2)$. $\square$

Lemma 6.6 Let Π be a general logic program. Then for any $X \subseteq \mathcal{P}$

$$\mathcal{A}(\Pi)(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X})$$

$\square$

Proof: Let $T = \mathcal{A}(\Pi)$. *(case 1)* $S(X) = \emptyset$

$$\bigcap_{\hat{X} \in S(X)} \Pi(\hat{X}) = Lit(\mathcal{P}),$$

hence the inclusion is true.
(case 2) $S(X) \neq \emptyset$
Let $s \in T(X)$. Since, for all $\hat{X} \in S(X)$, $X \subseteq \hat{X}$, by monotonicity we have $s \in T(\hat{X})$. Therefore from Lemma 6.4 we have $s \in \Pi(\hat{X})$. Hence,

$$\forall X \subseteq \mathcal{P}, \quad T(X) \subseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X}).$$

$\square$

Theorem 4.2 (Properties of $\mathcal{A}(\Pi)$ for an arbitrary general program Π) Let Π be a general logic program and a natural representation of a query Q, then $\mathcal{A}(\Pi)$ is a sound interpolation of Π. $\square$

Proof: Condition (1.) of Definition 3.5 is the statement of Lemma 6.5 and condition (3.) of Definition 3.5 is the statement of Lemma 6.6. $\square$

Lemma 6.7 Let Π be a stratified general logic program. Then

- for any $X \subseteq \mathcal{P}$, $(\mathcal{A}(\Pi)_X)^+$ is a stratified program;
- for any $X \in Dom(\Pi)$ program $\mathcal{A}(\Pi)_X$ has the unique answer set $A = M \cup \widetilde{M} \cup \{m_p : p \in M\}$, where M is a unique stable model of Π_X.

$\square$

Proof: Let $T = \mathcal{A}(\Pi)$.
Consider a stratification of the program Π_X :
$\pi^0 = \{q_1, \ldots, q_n\}$
$\pi^1 = \{p_{1,1}, \ldots, p_{1,k_1}\}$
$\pi^2 = \{p_{2,1}, \ldots, p_{2,k_2}\}$
...
$\pi^l = \{p_{l,1}, \ldots, p_{l,k_l}\}$
(where all the predicates from X belong to π^0).
Therefore general logic program $(T_X)^+$ is also stratified program with the stratification:
$\pi_T^0 = \{q_1, \ldots, q_n, q'_1, \ldots, q'_n\}$
$\pi_T^1 = \{m_{q_1}, \ldots, m_{q_n}\}$,
$\pi_T^2 = \{m_{p_{1,1}}, \ldots, m_{p_{1,k_1}}, p_{1,1}, \ldots, p_{1,k_1}\}$
$\pi_T^3 = \{p'_{1,1}, \ldots, p'_{1,k_1}\}$
...
$\pi_T^{2l-1} = \{p'_{l-1,1}, \ldots, p'_{l-1,k_{l-1}}\}$
$\pi_T^{2l} = \{m_{p_{l,1}}, \ldots, m_{p_{l,k_l}}, p_{l,1}, \ldots, p_{l,k_l}\}$
$\pi_T^{2l+1} = \{p'_{l,1}, \ldots, p'_{l,k_l}\}$

Hence, for any $X \subseteq \mathcal{P}$ ([GL88]) $(T_X)^+$ has a unique stable model. If $X \in Dom(\Pi)$, then, as follows from Lemma 6.3, A^+ is this unique stable model. □

Theorem 4.3 (Properties of $\mathcal{A}(\Pi)$ for a stratified general program Π) Let Π be a stratified general logic program and a natural representation of a query Q, then $\mathcal{A}(\Pi)$ is a weak interpolation of Π. □

Proof: Let $T = \mathcal{A}(\Pi)$.
Condition (1) and (3) of Definition 3.5 directly follows from Theorem 4.2.
Now we will prove Equivalence (Condition (2)) of Definition 3.5), that is $\forall X \in Dom(\Pi)$, $\Pi(X) = T(X)$. Indeed, since for any $X \subseteq \mathcal{P}$ program Π_X has a unique stable model M, $\Pi(X) = (M \cap \mathcal{V}) \cup (\widetilde{M} \cap \mathcal{V})$.
As was proved in Lemma 6.7 program T_X has the unique answer set $A = M \cup \widetilde{M} \cup \{m_p : p \in M\}$, therefore $T(X) = A \cap \mathcal{V}$.
If p is an atom, then $p \in \Pi(X)$ iff $p \in A \cap \mathcal{V} = T(X)$.
For literal $\neg p$, $\neg p \in \Pi(X)$ iff $\neg p \in \widetilde{M} \cap \mathcal{V}$ iff $p \notin M$ and $\neg p \in \mathcal{V}$
iff $\neg p \in T(X)$. □

Lemma 6.8 If Π is a signed program then
$Dom(\Pi) = \{X : X \subseteq \mathcal{P}, X \text{ is consistent and complete}\}$. □

Proof: It follows from Lemma 6.1 and the fact that for any $X \subseteq \mathcal{P}$ program Π_X is coherent [Dun92, Fag90]). □

Lemma 6.9 For any $X \subseteq \mathcal{P}$, the program $\mathcal{A}(\Pi) \cup X$ is coherent. □

Proof: Let $T = \mathcal{A}(\Pi)$. Consider the set $MB = \{m_p : p \in \mathcal{L}_\Pi\}$. It is easy to see, that MB is a signing for the general logic program $(T_X)^+$ for any $X \subseteq \mathcal{P}$. Therefore [Dun92, Fag90] program $(T_X)^+$ has a stable model and, hence, program T_X has an answer set (which, of course, may be inconsistent).

Lemma 6.10 Let $X \subseteq \mathcal{P}$. Program $\mathcal{A}(\Pi) \cup X$ is contradictory iff X is inconsistent. $\square$

Proof: Let $T = \mathcal{A}(\Pi)$. If X is inconsistent, then program $T \cup X$ is evidently contradictory.
Let us denote the language of the program T_X by $\mathcal{L}_T$ and consider $A = Lit(\mathcal{L}_T)$.
Following the arguments similar to the ones in the proof of Lemma 6.3, we can write, that

$$T_X = T_p \cup T_n \cup T_m$$

and

$$(T_X)^A = {T_p}^A \cup {T_n}^A \cup {T_m}^A = R_p \cup R_n \cup R_m.$$

Since $\forall p \quad p, \neg p \in A$,

1. $R_m = \emptyset$;
2. $R_n = \{\neg p : \neg p \in X\}$;
3. $R_p = T_p$.

Therefore $(T_X)^A = R_p \cup R_n$.
Consider the smallest subset S, closed under the rules of the program $R_p \cup R_n$. It is easy to see that $S \neq Lit(\mathcal{L}_T)$. Therefore if the answer set of $(T_X)^A$ equals to A then S must contain a pair of complementary literals [Gel91]. Since $\mathcal{P} \cap \mathcal{V} = \emptyset$, S contains a pair of complementary literals iff set X is inconsistent. $\square$

Let us introduce the following notations: for a given set $A \in Lit(\mathcal{L}_T)$, $A_p = atoms(A)$, $A_{mb} = \{m_p : m_p \in A\}$, $A_n = \{\neg p : \neg p \in A\}$, $A_u = \{p : p \notin A, \neg p \notin A\}$.
Obviously, $A = A_p \cup A_n \cup A_{mb}$.
Notice that for any atom p either $p \in A_p$ or $\neg p \in A_n$ or $p \in A_u$.

Lemma 6.11 Let Π be a general logic program with a signing R, $X \in Dom(\Pi)$, and A is a consistent answer set of $\mathcal{A}(\Pi)_X$. Then $M_0 = A_p \cup (A_u \cap R)$ is a stable model of Π_X. $\square$

Proof: Let $T = \mathcal{A}(\Pi)$.
First we will prove that M_0 is a *model* of the program $(\Pi_X)^{M_0}$ (we will denote $(\Pi_X)^{M_0}$ by S).
Consider a rule

$$t \leftarrow b_1, \ldots, b_n$$

with $n \geq 0$, from S. By definition of $(\Pi_X)^{M_0}$ there is a rule

$$t \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_k$$

with $k \geq 0$ in Π_X such that $\forall i, 1 \leq i \leq k,\ c_i \notin M_0$.
We now show that if $\forall i, 1 \leq i \leq k,\ c_i \notin M_0$ and $\forall j, 1 \leq j \leq n,\ b_j \in M_0$, then $t \in M_0$.
Consider 2 cases:

1. $\{t, b_1, \ldots, b_n\} \subset \bar{R}$ and $\{c_1, \ldots, c_k\} \subset R$;
2. $\{t, b_1, \ldots, b_n\} \subset R$ and $\{c_1, \ldots, c_k\} \subset \bar{R}$.

As follows from the definition of signing, for any rule from Π_X either 1. or 2. is true.

Consider case 1. Since $\{b_1, \ldots, b_n\} \subset M_0$ and $\{b_1, \ldots, b_n\} \subset \bar{R}$, from definition of M_0 we can conclude that $\{b_1, \ldots, b_n\} \subset A_p$.
Since $\{c_1, \ldots, c_k\} \subset R$ and $\forall i, 1 \leq i \leq k,\ c_i \notin M_0$, it is easy to see that $c_i \notin A_p$ and $c_i \notin A_u$, and, hence, $\neg c_i \in A$.
Therefore the corresponding rule

$$t \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_k$$

from T_X is such that its body is true in A and, hence, $t \in A_p \subset M_0$.

Consider case 2. Since $\{b_1, \ldots, b_n\} \subset M_0$ and $\{b_1, \ldots, b_n\} \subset R$ we can conclude that $\forall j, 1 \leq j \leq n$ either $b_j \in A_p \cap R$ or $b_j \in A_u \cap R$. In both cases, using Lemma 6.2 we can show that $m_{b_j} \in A_{mb} \subset A$.
Since $\forall i, 1 \leq i \leq k,\ c_i \notin M_0$ (and therefore $c_i \notin A_p$) we can conclude that $\forall i, 1 \leq i \leq k$ either $\neg\, c_i \in A_n$ (and, hence, $c_i \notin A$)
or $c_i \in A_u \cap \bar{R}$ (therefore also $c_i \notin A$).
Therefore the corresponding rule

$$m_t \leftarrow m_{b_1}, \ldots, m_{b_n}, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_k$$

from $T \cup X$ is such that its body is true in A and, hence, $m_t \in A_{mb}$.
If $t \in A_p \subset M_0$ then we proved our statement. If $t \notin A_p$ then since $m_t \in A$ from Lemma 6.2 follows that $\neg t \notin A_n$, hence $t \in A_u \cap R \subset M_0$.
Thus we proved our statement.

Now we will prove that M_0 is the minimal model of the program S.
Suppose it is not true and there exists $M' \subset M_0$ such that M' is a model of S.
Let $A' = M' \cup A_{mb} \cup A_n$. Using Lemma 6.2, we can show that set A' is consistent.
Since $atoms(X) \subset M'$ and $\{\neg p : \neg p \in X\} \subset A_n$, we conclude that $X \subset A'$.
We will now prove that A' is a model of $(T_X)^A$. Let us show that for any rule r from $(T_X)^A$ if the body of r is true in A' then its head belongs to A'. The program $(T_X)^A$ consists of the following types of rules:

1. $p_i \leftarrow$ or $\neg p_j \leftarrow$ such that $p_i, \neg p_j \in X$;
2. $m_p \leftarrow$ (for every $p \in \mathcal{P}$ such that $\neg p \notin A$);
3. $t \leftarrow b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_k$
(for every rule $t \leftarrow b_1, \ldots, b_n, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_k$ from the program Π_X);
4. $m_t \leftarrow m_{b_1}, \ldots, m_{b_n}$
(for every rule $t \leftarrow b_1, \ldots, b_n, \mathit{not}\ c_1, \ldots, \mathit{not}\ c_k$ from the program Π_X such that $\{c_1, \ldots, c_k\} \cap A_p = \emptyset$);
5. $\neg t \leftarrow$ (for every $t \in \mathcal{V}$ such that $m_t \notin A$).

We consider consequently every type of a rule.
Type 1.: since $X \subset A'$ and a head of this rule is always from X, our statement is true for any rule of this type;

Type 2.: since X is complete and consistent (as follows from Lemma 6.8) and $A \cap \mathcal{P} = X$, for $p \in \mathcal{P}$ $\neg p \notin A$ iff $p \in X$; hence, rule $m_p \leftarrow$ is in $(T \cup X)^A$ iff $p \in X$;
its body is true, and $m_p \in A_{mb} = A'_{mb} \subset A'$.

Type 3.: suppose $\{b_1, \ldots, b_n, \neg c_1, \ldots, \neg c_k\} \subset A'$, then $\{b_1, \ldots, b_n\} \subset M'$;
since $\{\neg c_1, \ldots, \neg c_k\} \subset A$ and A is consistent we can conclude that
$c_i \notin A_p$ and $c_i \notin A_u$ and therefore $c_i \notin M_0$ $\forall i, 1 \leq i \leq k$.
Hence the rule $t \leftarrow b_1, \ldots, b_n$ belong to the program S, its body is true in M' and (since M' is a model of S) $t \in M' \subset A'$.

Type 4.: suppose $\{m_{b_1}, \ldots, m_{b_n}\} \subset A'$, then since $A'_{mb} = A_{mb}$ we have that $\{m_{b_1}, \ldots, m_{b_n}\} \subset A$ and, therefore, $m_t \in A_{mb} = A'_{mb} \subset A'$.

Type 5.: since $m_t \notin A$, $\neg t \in A_n \subset A'$.

Thus we proved that A' is a model of $(T_X)^A$ and since $A' \subset A$, we obtained a contradiction with minimality of A. Therefore our initial assumption that there is a model $M' \subset M_0$ is not correct and M_0 is the minimal model of the program $(\Pi_X)^{M_0}$. So, we proved that M is a stable model of the program Π_X. □

Lemma 6.12 Let Π be a general signed logic program with signing R. Then for any $X \in Dom(\Pi)$ $\mathcal{A}(\Pi)(X) = \Pi(X)$. □

Proof: Let $T = \mathcal{A}(\Pi)$. First we will show that any answer set of T_X is consistent.
As follows from Lemma 6.9, $a(T_X) \neq \emptyset$. Consider an answer set $A \in a(T_X)$. If $A = Lit$, then, as was shown in [Gel91], A is the only answer set of T_X. Since R is the signing for Π_X, by [Dun92, Fag90] the program Π_X has a stable model M. As follows from Lemma 6.3, we can build a consistent answer set $A' = M \cup \widetilde{M} \cup \{m_c : c \in M\}$ of T_X, and, thus, we obtain a contradiction. Thus, we conclude that any answer set of T_X is consistent.

Due to Lemma 6.4 it is enough to prove, that $T(X) \supseteq \Pi(X)$.
Suppose atom $p \in \Pi(X)$, but $p \notin T(X)$. This means that there exists an answer set A_0 of $T(X)$ such that $p \notin A_0 \cap \mathcal{V}$. Since $R, \bar{R}$ are both signings for program Π_X, using Lemma 6.11, we can build the stable models $M_1 = A_{0p} \cup (A_{0u} \cap R)$ and $M_2 = A_{0p} \cup (A_{0u} \cap \bar{R})$ of program Π_X. Since $p \in \Pi(X) \subset M_1 \cap M_2 = atoms(A_0) \subset A_0$, we get a contradiction, which proves that $p \in T(X)$.
Suppose literal $\neg p \in \Pi(X)$, but $\neg p \notin T(X)$. This means that $\forall\, M \in S(\Pi_X)$ $p \notin M$ and there exists and answer set A_0 of $T(X)$ such that $\neg\, p \notin A_0$.
Since A_0 is consistent, from Lemma 6.2 it follows that $m_p \in A_0$. From the definitions of A_{0p}, and A_{0u} it follows that $m_p \in A_0$ iff $p \in A_{0p} \cup A_{0u}$. If $p \in R$, from Lemma 6.11 it follows that $p \in M_1$, where $M_1 = A_{0p} \cup (A_{0u} \cap R)$ is a stable model of Π_X. If $p \in \bar{R}$, from Lemma 6.11 it follows that $p \in M_2$, where $M_2 = A_{0p} \cup (A_{0u} \cap \bar{R})$ is a stable model of Π_X. In both cases we get a contradiction, which proves that $\neg\, p \in T(X)$. □

Lemma 6.13 Let Π be a general logic program with signing R. Then for any $X \subseteq \mathcal{P}$

$$\mathcal{A}(\Pi)(X) \supseteq \bigcap_{\hat{X} \in S(X)} \Pi(\hat{X}).$$

□

Proof: Let $T = \mathcal{A}(\Pi)$. Consider the following set
$S = \{p : p \in R\} \cup \{m_p : p \in R\} \cup \{p' : p \in \bar{R}\}$.
Since for any $X \subseteq \mathcal{P}$ sets R and $\bar{R}$ are both signings of program $T \cup X$, it is easy to check that S is a signing for the general logic program $T^+ \cup X^+$ for any $X \subseteq \mathcal{P}$.
The corresponding set $\bar{S}$ is also a signing for $T^+ \cup X^+$, where $\bar{S}$ equals to
$\{p : p \in \bar{R}\} \cup \{m_p : p \in \bar{R}\} \cup \{p' : p \in R\}$.
Let $\mathcal{P}_R = \{p : p \in \mathcal{P}, p \in R\}$ and $\mathcal{P}_{\bar{R}} = \{p : p \in \mathcal{P}, p \in \bar{R}\}$.
As follows from Lemma 6.12 $\forall X \in Dom(\Pi)$ $\Pi(X) = T(X)$, therefore it is enough to prove that for any $X \in \mathcal{P}$

$$T(X) \supseteq \bigcap_{\hat{X} \in S(X)} T(\hat{X}).$$

As follows from Lemma 6.9, program $T \cup X$ has an answer set.
If X is inconsistent then, as follows from Lemma 6.10, the answer set equals to Lit, therefore $T(X) = Lit(\mathcal{V})$ and the inclusion is true.
If X is consistent then, as follows from Lemma 6.10, $T \cup X$ is not contradictory and $T(X)$ is consistent.
We will now use Theorem 1 from [Tur93] for some special programs P and Q.
Consider

$$X_A = X \cup \{q : q \in \mathcal{P}_R \text{ } and \text{ } q, \neg q \notin X\} \cup \{\neg q : q \in \mathcal{P}_{\bar{R}} \text{ } and \text{ } q, \neg q \notin X\}.$$

Let $Q = T^+ \cup X^+$ and $P = T^+ \cup {X_A}^+$.
Let us check that conditions of Theorem 1 [Tur93] are satisfied.
Since all the difference between X and X_A is in S, $Q_S \subseteq P_S$, $Q_{\bar{S}} = P_{\bar{S}}$, and $literals(P) \cap \bar{S} = literals(Q) \cap \bar{S}$.
Therefore from Theorem 1 we can make the conclusion (which we will call Conclusion 1) that program Q entails every atom $f \in \bar{S}$ that is entailed by P or, in other words, for any atom $f \in \bar{S}$ if $T^+ \cup X_A \models f$ then $T^+ \cup X^+ \models f$.
Consider

$$X_B = X \cup \{q : q \in \mathcal{P}_{\bar{R}} \text{ } and \text{ } q, \neg q \notin X\} \cup \{\neg q : q \in \mathcal{P}_R \text{ } and \text{ } q, \neg q \notin X\}.$$

Let $Q = T^+ \cup X^+$, $P = T^+ \cup {X_B}^+$ and $S' = \bar{S}$.
Arguments similar to the ones in the first case show that $Q_{S'} \subseteq P_{S'}$, $Q_{\bar{S}'} = P_{\bar{S}'}$ and $literals(P) \cap S = literals(Q) \cap S$. Thus, the conditions of Theorem 1 are satisfied. Therefore we can make the conclusion (Conclusion 2), that program Q entails every atom $f \in S$ that is entailed by P or, in other words, for any atom $f \in S$
if $T^+ \cup {X_B}^+ \models f$ then $T^+ \cup X^+ \models f$.

Now we will prove that for any $X \in \mathcal{P}$

$$T(X) \supseteq \bigcap_{\hat{X} \in S(X)} T(\hat{X}).$$

Notice that, as follows from Lemma 6.8, $X_A, X_B \in S(X)$.
Let $f \in T(\hat{X})$ for any $\hat{X} \in S(X)$.
Consider the following cases:

- f is an atom.

1. $f \in \bar{R}$. Since $T \cup X_A \models f$ and $T \cup X_A$ is not contradictory, $T^+ \cup X^+{}_A \models f$, therefore from Conclusion 1 it follows that $T^+ \cup X^+ \models f$, and from the consistency of $T \cup X$ it follows that $f \in T(X)$.
2. $f \in R$. Since $T \cup X_B \models f$ and $T \cup X_B$ is not contradictory, $T^+ \cup X^+{}_B \models f$, therefore from Conclusion 2 it follows that $T^+ \cup X^+ \models f$, and from the consistency of $T \cup X$ it follows that $f \in T(X)$.

- $f = \neg p$, where p is an atom.

1. $p \in R$. Since $T \cup X_A \models f$ and $T \cup X_A$ is not contradictory, $T^+ \cup X^+{}_A \models f^+$, therefore from Conclusion 1 it follows that $T^+ \cup X^+ \models f^+$, and from the consistency of $T \cup X$ it follows that $f \in T(X)$.
2. $f \in \bar{R}$. Since $T \cup X_B \models f$ and $T \cup X_B$ is not contradictory, $T^+ \cup X^+{}_B \models f^+$, therefore from Conclusion 2 it follows that $T^+ \cup X^+ \models f^+$, and from the consistency of $T \cup X$ it follows that $f \in T(X)$.

Since in all cases we have $f \in T(X)$, we proved the inclusion. □

Theorem 4.4 (Properties of $\mathcal{A}(\Pi)$ for a signed program Π) For any signed general logic program Π that is a natural representation of a query Q, $\mathcal{A}(\Pi)$ is an interpolation of Π. □

Proof: We will use Proposition 3.1 to prove the Theorem.
Lemma 6.12 proves that Condition 1 (Equivalence) from Proposition 3.1 is satisfied.
Lemma 6.5 proves that Condition 2 (Monotonicity) from Proposition 3.1 is satisfied.
Lemma 6.13 proves that Condition 3 (Maximal Informativeness) from Proposition 3.1 is satisfied.
Therefore, as follows from Proposition 3.1, program T interpolates program Π. □

Theorem 4.5 (Categoricity) Let Π be a signed general logic program with signing R, a natural representation of a query Q, and $X \subseteq \mathcal{P}$. Program $\Pi \cup atoms(X)$ is categorical if and only if program $\mathcal{A}(\Pi) \cup X$ is categorical. □

Proof: Let us denote $\mathcal{A}(\Pi)$ by T and let $X \subseteq \mathcal{P}$.
"$\Longleftarrow$" Let us denote by A a unique answer set of T_X. Suppose there exist $M_1, M_2 \in S(\Pi_X)$ such that $M_1 \neq M_2$. Then, using Lemma 6.3, we can built answer sets $A_1 = M_1 \cup \widetilde{M_1} \cup \{m_c : c \in M_1\}$, $A_2 = M_2 \cup \widetilde{M_2} \cup \{m_c : c \in M_2\}$ of program T_X. Since $A_1 \neq A_2$, we get a contradiction, which proves that program Π_X is categorical.
"$\Longrightarrow$" Let M be the unique stable model of program Π_X.
As follows from Lemma 6.3, the set $A = M \cup \widetilde{M} \cup \{m_p : p \in M\}$ is a consistent answer set of program T_X. Therefore, as follows from [Gel91], T_X is not contradictory.
Suppose there is another answer set $A1$ of T_X. We will show that $A1 = A$.
Program T_X is not contradictory, therefore $A1$ is a consistent set. Since $R, \bar{R}$ are both signings of Π_X, using Lemma 6.11, we can built stable models $M1 = A1_p \cup (A1_u \cap R)$, $M2 = A1_p \cup (A1_u \cap \bar{R})$ of program Π_X.
From the uniqueness of M follows that $M1 = M2 = M = A1_p$ and $A1_u = \emptyset$. Hence $A = A1_p \cup \{\neg\, p : p \notin A1_p\} \cup \{m_p : p \in A1_p\} = A1$. Thus we proved that program $\mathcal{A}(\Pi)_X$ is categorical. □