On the semantics of logic program composition

Antonio Brogi

Department of Computer Science, University of Pisa, Italy

Abstract. This paper aims at offering an insightful synthesis of different compositional semantics for logic program composition which have been developed in the literature. In particular, we will analyse the notions of program equivalence, compositionality, and full abstraction for logic programs. We will show how the notion of *supported interpretation* provides a unifying compositional model-theoretic characterisation both of positive programs and of programs containing negation.

1 Introduction

Building complex software systems by combining existing components is a standard methodology of modern software development. The effectiveness of the program composition approach depends on the possibility of reasoning on the composition process itself. The availability of well-founded characterisations of programs and program compositions is needed to perform transformation, analysis and verification.

One of the most important relations between programs (in any programming language) is *program equivalence*. This relation is at the basis of most, if not all, programming methodologies. Each method of giving a semantics to programs induces an equivalence relation on programs. It is therefore essential to understand how these equivalences are related.

As pointed out in [30], different formulations that define identical equivalences offer different frameworks in which to reason about programs. Moreover, stronger equivalence relations may be used to reason about programs and ensure that the programs are equivalent in a weaker sense, which might not be as suitable for reasoning. Reasoning about programs concerns also the correctness of source-to-source transformations such as those occurring in program development [35].

The properties of *compositionality* and *full abstraction* play a crucial role in the study of the semantics of programming languages. Simply stated, a semantics is compositional (or homomorphic) if the meaning of a program can be obtained from the meaning of its components. If a semantics is compositional with respect to some composition operations then the induced equivalence relation is a congruence for those operations. This property establishes a firm foundation for reasoning about programs and program transformations. Suppose that a program P consists of two parts, Q and R say, suitably composed together. Suppose also that R' is a more efficient version of R, obtained for instance by applying some program transformation technique to R. If R' is equivalent to R

in the chosen semantics then the property of compositionality ensures that the substitution of R' for R will not affect the meaning of the whole program P.

Often the semantics $\mathcal{O}$ describing the observable behaviour of programs is not compositional. In these cases it is then necessary to consider a more distinguishing (or finer) semantics $\mathcal{S}$ which preserves $\mathcal{O}$ and which is a congruence for the set of compositions considered. The compositionality of $\mathcal{S}$ ensures that programs (or program parts) which are $\mathcal{S}$-equivalent can be replaced with one another without affecting the intended semantics $\mathcal{O}$ of the whole system. The property of full abstraction establishes that the equivalence relation induced by $\mathcal{S}$ is the largest equivalence relation that can be used to substitute programs (or program parts) without affecting the intended semantics of the whole system.

In this paper we analyse the properties of compositionality and full abstraction in the context of logic programming. Indeed, because of the declarative programming style it features, logic programming can be fruitfully employed as the specification language of software components. Logic programming supports a wealth of programming styles developed in algorithmic programming, database programming, and artificial intelligence programming via a small number of powerful features (unification, recursion, and nondeterminism). On the other hand, logic programming has firm foundations in mathematical logic. The availability of different equivalent characterisations of programs offers the ground to perform sound semantics-based transformation, analysis and verification.

In this paper, we focus on the most basic composition operation over logic programs, the *union* of programs, and we analyse and compare different semantics that have been proposed in the literature. In the perspective of providing an insightful synthesis of these semantics, we will show how the notion of *supported interpretation* provides a unifying characterisation of both positive programs and of programs containing negation. Notice that the aim of this paper is not to provide a comprehensive survey of the compositional semantics for logic programs which have been proposed in the last ten years. (The interested reader may refer to [10] for a survey which covers also different modular extensions of logic programming.)

The rest of the paper is organised as follows.

Section 2 introduces some background material, namely some logic programming terminology, the notions of compositionality and full abstraction, and a hierarchy of logic program equivalences.

Section 3 is devoted to analyse compositional semantics of definite programs. We first analyse three equivalence relations considered in [30]: Subsumption equivalence, weak subsumption equivalence and logical equivalence. We show that while they are all compositional, logical equivalence is *the* fully abstract relation. We then consider a different model-theoretic characterisation, based on the notion of *admissible model* presented in [10]. The relation between admissible models and the other semantics is illustrated, and a fully abstract refinement of admissible models is presented here for the first time. The section is concluded by introducing the notion of *supported interpretation* which provides an alternative characterisation of logical equivalence and which will be used also to charac-

terise normal programs. The results presented in this section are summarised in Figure 2 which contains a hierarchy of compositional semantics for definite programs.

Section 4 is devoted to analyse compositional semantics for normal programs, that is, programs containing negation. Two main problems arise here: (1) the existence of many "intended" semantics for normal programs, and (2) the orthogonality of non-monotonicity and compositionality. We will show that the notion of supported interpretation introduced in Section 3 provides a unifying characterisation of a number of "intended" semantics for normal programs. A general full abstraction result will be also presented here for the first time.

Finally, Section 5 briefly discusses other forms of program compositions, while Section 6 contains some concluding remarks.

To simplify the reading, all proofs are reported in the Appendix.

2 Preliminaries

2.1 Logic programming

We will use the standard definitions and terminology of logic programming, as reported for instance in [2, 29]. A *definite logic program* is a finite set of clauses of the form $A \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$, where A,B_1,...,B_n are atoms. A *normal logic program* is a finite set of clauses of the form $A \leftarrow B_1, \ldots, B_n$ $(n \geq 0)$, where A is an atom and where B_1,...,B_n are possibly negated atoms.

Clauses without premise part, i.e., of the form $A \leftarrow$, will be called *extensional* (or unit) clauses, and programs containing only extensional clauses will be called *extensional programs*. We will also denote by $Defs(P)$ the set of predicates defined in a program P.

We will use the standard notions of Herbrand interpretations and Herbrand models, and we will denote by $LHM(P)$ the least Herbrand model of a program P. We will also use the definition of the standard immediate consequence operator $T(P)$:

$$T(P)(I) = \{A \mid \exists \overline{B} : A \leftarrow \overline{B} \in ground(P) \wedge \overline{B} \subseteq I\}$$

where P is a definite program, $\overline{B}$ denotes a (possibly empty) conjunction of atoms, and $ground(P)$ denotes the set of ground instances of clauses of P.

2.2 Compositionality and full abstraction

A semantics for a programming language provides meanings for programs or, more generally, program parts. Moreover, each method of giving semantics to a programming language induces an equivalence relation on programs. Namely, two programs are equivalent if and only if they have the same meaning in the chosen semantics.

An equivalence relation $\equiv_1$ is finer than another equivalence relation $\equiv_2$ ($\equiv_1 \subseteq \equiv_2$) if and only if whenever $P \equiv_1 Q$ then $P \equiv_2 Q$. Furthermore $\equiv_1$ is

strictly finer than $\equiv_2$ ($\equiv_1 \subset \equiv_2$) if and only if $\equiv_1$ is finer than $\equiv_2$ and $\equiv_2$ is not finer than $\equiv_1$.

The properties of *compositionality* and *full abstraction* have been recognised as two fundamental concepts in the studies on the semantics of programming languages [32, 36]. Informally, a semantics is compositional if equivalent programs (or program parts) are indistinguishable, that is, if they exhibit equal observable behaviour in all possible context. On the other hand, a semantics is fully abstract if indistinguishable programs (or program parts) are equivalent.

The formal definition of these properties relies on the notion of observable behaviour of a program and on the notion of program composition. The former can be represented by a mapping $\mathcal{O}$ which associates with every program P an object $\mathcal{O}(P)$ denoting the observable behaviour of P. The latter can be represented by a set Com of (possibly partial) functions over programs.

A semantics is *compositional* if the induced equivalence is compositional for the pair $(\mathcal{O}, Com)$, that is, if it preserves the observables and is a congruence for the set of compositions. Formally, an equivalence relation $\equiv$ is *compositional* for $(\mathcal{O}, Com)$ if and only if:

1. $\equiv$ *preserves* $\mathcal{O}$, that is $\forall P, Q: \ P \equiv Q \implies \mathcal{O}(P) = \mathcal{O}(Q)$.
2. $\equiv$ *is a congruence* for Com, that is $\forall \mathcal{F} \in Com$, $\forall P_1, \ldots, P_n, Q_1, \ldots, Q_n$: $P_i \equiv Q_i \ (i = 1, \ldots, n) \implies \mathcal{F}(P_1, \ldots, P_n) \equiv \mathcal{F}(Q_1, \ldots, Q_n)$.

There is always a coarsest congruence for $(\mathcal{O}, Com)$, which is intuitively the "indistinguishability relation". A semantics is fully abstract if the induced equivalence includes this largest congruence. A semantics is both compositional and fully abstract if it coincides with it. Two programs P and Q are *distinguishable* under $(\mathcal{O}, Com)$ if there exists a context $\mathcal{C}[.]$ (defined via Com) in which the substitution of P with Q changes the external behaviour (defined via $\mathcal{O}$) of the context. Formally, P and Q are *distinguishable* iff $\exists \mathcal{C}[.]: \ \mathcal{O}(\mathcal{C}[P]) \neq \mathcal{O}(\mathcal{C}[Q])$. We put:

$$P \cong Q \iff P \text{ and } Q \text{ are not distinguishable under } (\mathcal{O}, Com).$$

Then an equivalence relation $\equiv$ is *fully abstract* for $(\mathcal{O}, Com)$ if and only if:

$$\forall P, Q: \ P \cong Q \implies P \equiv Q.$$

In this paper, we consider both definite and normal programs, and we assume that the language (or vocabulary) in which programs are written is fixed. Namely, the Herbrand base $\mathcal{B}$ we refer to is determined by a set of function and predicate symbols that include all function and predicate symbols used in the programs being considered. We will consider (set-theoretic) union of programs (denoted by $\cup$) as the only composition operation. The observable behaviour of a logic program may be defined in different ways, depending on which aspects of the computation one is interested in looking at. In the case of definite logic programs, a natural choice of the observables is the *success set* of a program [2, 29, 40]. The success set $SS(P)$ of a program P is the set of ground atoms A such that $P \cup \{\leftarrow A\}$ has a SLD-refutation. Therefore we put:

$$\mathcal{O}(P) = SS(P).$$

2.3 Equivalence of definite logic programs

A number of different equivalence relations for logic programs were studied and compared with one another in [30].

The equivalence relation induced by the immediate consequence operator $T(P)$ is one of these equivalences. Namely two programs are equivalent if and only if they have the same $T(P)$, that is, their immediate consequence operators coincide on every Herbrand interpretation. In [30] a syntactic notion of equivalence, *subsumption equivalence* was also introduced, and it was shown to coincide with the equality of $T(P)$ functions on programs. Let C_1 and C_2 be the definite clauses $A \leftarrow \bar{B}$ and $D \leftarrow \bar{E}$, respectively. C_1 is subsumed by C_2 if there is a substitution ϑ such that $A = D\vartheta$ and $\bar{E}\vartheta \subseteq \bar{B}$. Two logic programs P and Q are subsumption equivalent if every clause of P is subsumed by some clause of Q and vice-versa. Existing algorithms [25] can be therefore exploited to determine whether two programs are $T(P)$ equivalent.

Another equivalence relation considered in [30] is a refinement of subsumption equivalence, named *weak subsumption equivalence*. Namely two programs are weakly subsumption equivalent if and only if the two programs without tautologies are subsumption equivalent. As for the previous case, an equivalent formulation is given in terms of a refinement of the $T(P)$ semantics, defined by means of a $T(P) + Id$ function.

Furthermore, logical equivalence ($\models P \longleftrightarrow Q$) and the corresponding equivalence when only Herbrand models are considered ($M(P) = M(Q)$) are studied. It is also shown that these two equivalent relations can be equivalently formulated in terms of the functional semantics defined in [27].

Finally, the standard equivalence relation induced by the operational semantics of logic programs is considered, which identifies programs with same success set ($SS(P) = SS(Q)$), and the latter coincides with the least Herbrand model semantics [40].

Different formulations of equivalence are also compared in terms of their relative strength. In addition to the previously mentioned correspondences, it is shown that subsumption equivalence is strictly finer than weak subsumption equivalence, which is in turn strictly finer than logical equivalence, which is in turn strictly finer than operational equivalence.

Some of the results presented in [30] are summarised in Figure 1, where an arrow from $\equiv_1$ to $\equiv_2$ denotes that $\equiv_1$ is strictly finer than $\equiv_2$ (viz., $\equiv_1 \subset \equiv_2$).

3 Composition of definite programs

The union of programs is the most basic composition operation over logic programs. Actually, every logic program consists of the union of all its clauses. The starting point of our analysis is the observation that the standard (model-theoretic, fixpoint or operational) semantics of logic programs is not compositional w.r.t. the union of programs.

The least Herbrand model is usually taken as the the intended meaning of a definite logic program. Unfortunately, the least Herbrand model of the union of

$$
\begin{array}{rcl}
P \text{ s-e } Q & \longleftrightarrow & T(P) = T(Q) \\
 & & \downarrow \\
P \text{ w s-e } Q & \longleftrightarrow & T(P) + Id = T(Q) + Id \\
 & & \downarrow \\
\models P \leftrightarrow Q & \longleftrightarrow & M(P) = M(Q) \\
 & & \downarrow \\
SS(P) = SS(Q) & \longleftrightarrow & LHM(P) = LHM(Q)
\end{array}
$$

Fig. 1. Equivalence hierarchy for logic programs.

two programs cannot always be determined from the least Herbrand models of the separate programs.

Example 1. For instance the program:

$$fallible(x) \leftarrow human(x)$$

is equivalent to the empty program, as the empty set is the least model of both programs. On the other hand, if these programs are composed with the program:

$$human(socrates) \leftarrow$$

we obtain two programs which have different least models ($\{human(socrates), fallible(socrates)\}$ and $\{human(socrates)\}$, respectively). $\Diamond$

The above example shows that the least Herbrand model semantics is not compositional w.r.t. the union of programs. The same observation applies to the standard least fixpoint and to the standard operational semantics, as these three semantics are all equivalent for definite programs [40].

A number of different compositional denotational semantics for logic programs have been proposed. In the next sections we will present some of those semantics, and analyse the existing relations among them.

3.1 Subsumption equivalence

One of the first compositional semantics for logic programs was presented in [31]. Intuitively speaking, the idea of [31] is to adopt a higher-order semantics in order to achieve a compositional denotation of programs. Simply stated, a program P is denoted by its immediate consequence operator $T(P)$ rather than by the least fixpoint of $T(P)$, as done in the standard least fixpoint semantics of logic programs.

Indeed, the immediate consequences of the union of two programs can be determined by the immediate consequences of the two programs in the following way [31]:

$$T(P \cup Q)(I) = T(P)(I) \cup T(Q)(I)$$

(where abusing notation the $\cup$ on the left-hand side denotes program union while the $\cup$ on the right-hand side denotes set-theoretic union).

Let us denote by $\equiv_T$ the equivalence relation induced by the $T(P)$ semantics:

$$P \equiv_T Q \iff T(P) = T(Q).$$

Namely two programs are equivalent if and only if their immediate consequence operators coincide on every Herbrand interpretation. The equivalence relation $\equiv_T$ is a congruence for the union of programs (as well as for several other interesting composition operations, as shown for instance in [7]). Moreover, the equivalence relation $\equiv_T$ preserves the observables since:

$$\mathcal{O}(P) = SS(P) = LHM(P) = T^{\omega}(P)(\emptyset).$$

The equivalence relation $\equiv_T$ is hence a congruence for $(\mathcal{O}, \{\cup\})$. It is however easy to observe that $\equiv_T$ is not fully abstract for $(\mathcal{O}, \{\cup\})$. Indeed there are programs which are not subsumption equivalent, though they cannot be distinguished operationally.

Example 2. Consider for instance the programs:

P	Q
$a \leftarrow$	$a \leftarrow b$
$b \leftarrow$	$b \leftarrow$

We see that P and Q are indistinguishable under $(\mathcal{O}, \{\cup\})$ though they are not subsumption equivalent — since $T(P)(\emptyset) = \{a, b\}$ while $T(Q)(\emptyset) = \{b\}$. $\Diamond$

3.2 Weak subsumption equivalence

The weak subsumption equivalence relation was introduced in [30] as a refinement of subsumption equivalence. Two programs are weakly subsumption equivalent if and only if the two programs without tautologies are subsumption equivalent.

Weak subsumption equivalence can be characterised in terms of the function $T(P)$ by introducing a new operator $(T(P) + Id)$ defined as follows:

$$(T(P) + Id)(I) \;=\; I \cup T(P)(I)$$

and then by proving that [30]:

$$P \textit{ is weakly subsumption equivalent to } Q \iff (T(P) + Id = T(Q) + Id).$$

Let us denote by $\equiv_{T+Id}$ the equivalence relation induced by $T(P) + Id$, that is:

$$P \equiv_{T+Id} Q \iff (T(P) + Id = T(Q) + Id).$$

As shown in [12], $\equiv_{T+Id}$ is a congruence for the union of programs. Indeed, for any interpretation I:

$$(T(P_1 \cup P_2) + Id)(I) = I \cup T(P_1 \cup P_2)(I) = I \cup T(P_1)(I) \cup T(P_2)(I).$$

Therefore, if $P_1 \equiv_{T+Id} Q_1$ and $P_2 \equiv_{T+Id} Q_2$ then for all I:

$$(T(P_1 \cup P_2) + Id)(I) \;=\; (T(Q_1 \cup Q_2) + Id)(I).$$

Since the equivalence relation $\equiv_{T+Id}$ preserves $\mathcal{O}$ [30] it is hence compositional for $(\mathcal{O}, \{\cup\})$. Weak subsumption equivalence is coarser than subsumption equivalence, that is $\equiv_{T+Id}$ distinguishes less programs than $\equiv_T$.

Example 3. For instance $\equiv_T$ distinguishes the programs:

P	Q
$a \leftarrow b$	$a \leftarrow b$
	$b \leftarrow b$

(since $T(P)(\{b\}) \subset T(Q)(\{b\})$) while they are equivalent under $\equiv_{T+Id}$. Indeed programs P and Q are identical up to tautologies and for each I:

$$(T(P) + Id)(I) = (T(Q) + Id)(I) = \begin{cases} I & \text{if } b \notin I \\ I \cup \{a\} & \text{if } b \in I \end{cases}$$

$\Diamond$

As for the case of $\equiv_T$, we can however observe that $\equiv_{T+Id}$ is not fully abstract for $(\mathcal{O}, \{\cup\})$. For instance, programs P and Q of Example 2 are indistinguishable under $(\mathcal{O}, \{\cup\})$ though they are not weak subsumption equivalent since $(T(P) + Id)(\{\}) = \{a, b\}$ and $(T(Q) + Id)(\{\}) = \{b\}$.

3.3 Logical equivalence

While subsumption equivalence and weak subsumption equivalence are both compositional for $(\mathcal{O}, \{\cup\})$, they are not fully abstract for $(\mathcal{O}, \{\cup\})$ since they both distinguish programs that are instead operationally indistinguishable. If we look for a fully abstract denotation of programs, we must then consider some weaker equivalence relation over programs. The natural next candidate to examine, following the hierarchy of Figure 1, is logical equivalence.

In the case of logic programs, logical equivalence coincides with the equivalence induced by the set of (all) Herbrand models of a program. Two definite programs are logically equivalent if and only if they have the same Herbrand models. If we denote by $M(P)$ the set of Herbrand models of a program P:

$$M(P) \;=\; \{I \;\mid\; I \models P\}$$

then logical equivalence can be denoted as follows:

$$P \equiv_M Q \iff M(P) = M(Q).$$

The reason why the least Herbrand model semantics does not properly cope with program composition derives from the underlying Closed World Assumption (CWA) [39]. According to the CWA, and to the corresponding completion

semantics [15], a logic program is interpreted as a complete knowledge specification. Such an interpretation does not reflect the implicit assumption underlying program composition, that is, that a program is an incomplete chunk of knowledge to be possibly completed with other knowledge. As a consequence, each program cannot be simply denoted by its least Herbrand model, where only provable formulae are considered. Also non-minimal Herbrand models of a program must be considered, including formulae not provable in the program, but which can possibly become provable after some program composition.

In this perspective, a compositional semantics of logic programs was defined in [11] by denoting a program with the set of *all* its Herbrand models. Indeed the Herbrand models of the union of two programs can be determined by the Herbrand models of the separate programs, as shown by the following observation.

Observation 1 *Let P and Q be definite programs. Then:*

$$I \in M(P \cup Q) \iff I \in M(P) \ \wedge \ I \in M(Q).$$

Namely an interpretation I is a model of the union of two programs if and only if I is a model of both programs. Therefore the set of models of the union of two programs coincides with the intersection of the set of models of the two programs, that is:

$$M(P \cup Q) = M(P) \cap M(Q)$$

and the least Herbrand model of the union of two programs is hence the least Herbrand interpretation which is a model of both programs.

Logical equivalence preserves the least Herbrand models semantics $\equiv_{LHM}$, since $\mathcal{O}(P) = \bigcap \{I \mid I \in M(P)\}$, and hence logical equivalence is compositional for $(\mathcal{O}, \{\cup\})$. This means that if two programs are logically equivalent then they are also operationally indistinguishable, that is, they exhibit the same observable behaviour in all possible contexts.

Differently from subsumption equivalence and weak subsumption equivalence, logical equivalence is fully abstract for $(\mathcal{O}, \{\cup\})$. Indeed, as proved in [12], programs which are indistinguishable w.r.t. $(\mathcal{O}, \{\cup\})$ are also logically equivalent. It is perhaps worth recalling here a sketch of the proof of the full abstraction of logical equivalence reported in [12].

> *The proof shows that if two programs P and Q are not logically equivalent, then there exists a context in which they exhibit different observational behaviour. By definition of logical equivalence, if $P \not\equiv_M Q$ then there exists an interpretation I such that $I \in M(P)$ and $I \notin M(Q)$. By definition of Herbrand model [29], this means that $T(P)(I) \subseteq I$ and $T(Q)(I) \not\subseteq I$. This implies that there exists a finite subset F of I such that $A \in T(Q)(F)$ while $A \notin F$, for some atom A. The proof is finally concluded by considering the program $R = \{B \leftarrow \mid B \in F\}$ and by showing that $A \notin \mathcal{O}(P \cup R)$ while $A \in \mathcal{O}(Q \cup R)$.*

Example 4. Consider for instance the programs:

P	Q
$a(x) \leftarrow c(x)$	$a(x) \leftarrow b(x)$
$b(x) \leftarrow c(x)$	$b(x) \leftarrow c(x)$

Any interpretation of the form:

$$I = \{a(t) \mid t \in T\} \cup \{b(u) \mid u \in U\}$$

(where T and U are —possibly infinite— sets of ground terms such that $T \subset U$) is a model for P and not for Q. Following the above proof sketch, we observe that there exists a finite subset F of I (for instance $F = \{b(k)\}$ for any $k \in U - T$) such that $a(k) \in T(Q)(F)$ and $a(k) \notin F$. If we then consider the program:

R
$b(k) \leftarrow$

we see that $a(k) \notin \mathcal{O}(P \cup R)$ while $a(k) \in \mathcal{O}(Q \cup R)$. $\Diamond$

The results proved in [12] establish that logical equivalence is *the* fully abstract compositional equivalence relation for $(\mathcal{O}, \{\cup\})$. This means that Herbrand models induce the coarsest equivalence relation on programs w.r.t. $(\mathcal{O}, \{\cup\})$, in that any other denotation of programs one may choose must induce the same equivalence relation to be compositional and fully abstract.

3.4 Admissible models

Before the full abstraction of logical equivalence was established, a different compositional model-theoretic semantics for definite programs was presented in [10]. The idea of [10] was to model the composition of definite programs by denoting each program with a subset of its Herbrand models, called the *admissible* Herbrand models. Intuitively speaking, a model is considered to be admissible if it is "supported" by a set of hypotheses which all occur in the bodies of the program clauses. The intuition behind the admissible model semantics is to consider only those Herbrand models which somehow denote the effects of the possible compositions of a program with other programs.

It is worth observing that each Herbrand model is "supported" by the assumption of a set of hypotheses.

Lemma 1. *Let P be a program and let $I \subseteq \mathcal{B}$. Then:*

$$I \in M(P) \iff \exists H \subseteq \mathcal{B} : I = LHM(P \cup H).$$

Following [10], a set of admissible hypotheses is formally defined as follows:

> *A set $H \subseteq \mathcal{B}$ is an* admissible set of hypotheses *for a program P if and only if for all $h \in H$ there exists a ground instance $A \leftarrow \overline{B}$ of a clause in P such that $h \in \overline{B}$.*

The set possible admissible hypotheses for a program P is hence defined as follows:

$$AH(P) = \{h \mid h \in \mathcal{B} \wedge \exists A, \overline{B} : (A \leftarrow \overline{B} \in ground(P) \wedge h \in \overline{B})\}.$$

The notion of admissible model is then defined as follows:

> *Let P be a program, let $I \subseteq \mathcal{B}$, and let $H \subseteq AH(P)$. Then I is an* admissible model *for P under the hypotheses H if and only if $I = LHM(P \cup H)$.*

A model I that is admissible under the set of hypotheses H is denoted by $I(H)$ to explicitly record the set of hypotheses supporting it. The set of admissible models $AM(P)$ for a program P is hence defined as follows:

$$AM(P) = \{I(H) \mid I \subseteq \mathcal{B} \ \wedge \ H \subseteq AH(P) \ \wedge \ I = LHM(P \cup H)\}.$$

It is easy to observe that the least Herbrand model is always an admissible model (under the empty set of hypotheses).

Example 5. Consider for instance the program P:

$$\begin{array}{l} P \\ a \leftarrow b \\ c \leftarrow \end{array}$$

which has three Herbrand models: $\{c\}$, $\{a, c\}$, and $\{a, b, c\}$. Since b is the only admissible hypothesis for P, there are only two admissible models for P: $\{c\}$ — admissible under the empty set of interpretations — and $\{a, b, c\}$ — admissible under the set of interpretations $\{b\}$. The model $\{a, c\}$ is instead considered not admissible since there is no admissible set of hypotheses supporting it. $\Diamond$

As shown in [10], admissible models define a compositional semantics for logic programs. Indeed the admissible models of the union of two programs can be determined by composing the admissible models of the separate programs. Such composition is defined in [10] by means of a $\mathcal{T}(S)$ operator which, given a set S of admissible models, maps Herbrand interpretations into Herbrand interpretations. Intuitively, the definition of $\mathcal{T}(S)$ lifts the definition of $T(P)$ from program clauses to program (admissible) models. Namely, $T(P)(I)$ yields the union of all the atoms A such that P contains an implication "A if B" whose premise B is true in the interpretation I. Similarly, $\mathcal{T}(S)(I)$ yields the union of all the conclusions J such that $J(H)$ is an admissible model whose premise H is true in I.

The equivalence induced by the admissible models semantics is defined as follows:

$$P \equiv_{AM} Q \iff AM(P) = AM(Q).$$

Namely $P \equiv_{AM} Q$ if and only if the set of pairs $\langle I, H \rangle$ such that $I(H)$ is an admissible model is the same for both programs. It is easy to observe that the equivalence relation $\equiv_{AM}$ preserves the observables $\mathcal{O}$ since

$$LHM(P) = \bigcap \{I \mid I(H) \in AM(P)\}$$

and hence $\equiv_{AM}$ is compositional for $(\mathcal{O}, \{\cup\})$.

We can however observe that $\equiv_{AM}$ is not fully abstract for $(\mathcal{O}, \{\cup\})$. Indeed there are programs which do not have the same set of admissible models, although they cannot be distinguished operationally.

Example 6. For instance the programs:

P	Q
$a \leftarrow b$	$a \leftarrow$
$b \leftarrow$	$b \leftarrow a$

are indistinguishable under $(\mathcal{O}, \{\cup\})$ though they do not have the same set of admissible models. Indeed P has two admissible models, $\{a, b\}(\emptyset)$ and $\{a, b\}(\{b\})$, while Q has the admissible models $\{a, b\}(\emptyset)$ and $\{a, b\}(\{a\})$. $\Diamond$

The above example also highlights that programs having the same Herbrand models may have different sets of admissible models. Indeed this *per se* shows the non-fully abstractness of $\equiv_{AM}$ once the full abstraction of logical equivalence has been established.

3.5 Minimal admissible models

In Section 3.3 we have shown that the set of Herbrand models induces a fully abstract compositional equivalence relation (viz., logical equivalence $\equiv_M$). In the previous section we have shown that the idea of considering only *admissible* Herbrand models yields a compositional equivalence relation (viz., $\equiv_{AM}$) which is however not fully abstract.

An intriguing question is whether it is possible to refine the notion of admissible Herbrand model so as to restrict the set of admissible models of a program and to obtain a fully abstract denotation of programs.

Example 6 reported at the end of the previous section shows that the set of admissible models of a program includes models that are somehow "redundant" in view of possible program compositions.

Example 7. Consider again program P of Example 6:

P
$a \leftarrow b$
$b \leftarrow$

We observe that the inclusion of the admissible model $\{a, b\}(\{b\})$ does not really add much information to the program denotation, given the presence of the model $\{a, b\}(\emptyset)$. Intuitively speaking, the possible effects of the hypothesis b becoming true (because of some program composition) are already denoted by the admissible model $\{a, b\}(\emptyset)$. $\Diamond$

Following the above observation, the definition of admissible model may hence be refined so as to exclude the somehow "redundant" models. Intuitively speaking, we might consider $I(H)$ to be a "minimal" admissible model only if H is the

minimal set of hypotheses needed to derive the set of conclusions I, that is, only if:

$$\forall K \subset H : LHM(P \cup K) \subset LHM(P \cup H)$$

This constraint would eliminate some redundant admissible models —e.g., the indistinguishable programs P and Q of Example 6 would now have $\{a, b\}(\emptyset)$ as the only admissible model.

A stronger constraint is however needed to avoid all redundant models.

Example 8. For instance the program:

P
$a \leftarrow a$

would still have two admissible models ($\emptyset(\emptyset)$ and $\{a\}(\{a\})$) while being operationally indistinguishable from the empty program. Intuitively speaking, the information contained in the model $\{a\}(\{a\})$ for P is somehow already contained in the model $\emptyset(\emptyset)$, since the addition of the new hypothesis a does not add any other conclusion besides itself. $\Diamond$

We therefore say that an admissible model $I(K \cup \Delta)$ is not redundant w.r.t. another admissible model $J(K)$ only if the extra hypotheses Δ add some other conclusions besides themselves, that is, only if $(I - J) \supset \Delta$. Formally, we define the set of minimal admissible models for a program P as follows:

$$\mu AM(P) = \{I(H) \mid I \subseteq \mathcal{B} \ \wedge \ H \subseteq AH(P) \ \wedge \ I = LHM(P \cup H) \\ \wedge \forall K \subset H : LHM(P \cup H) - LHM(P \cup K) \supset H - K\}.$$

Let us consider a simple example in order to better illustrate the way in which minimal admissible models restrict admissible models.

Example 9. Consider the program:

P
$a \leftarrow b, c$
$b \leftarrow$
$c \leftarrow d, c$

which has eight admissible models:

$$\begin{array}{rr} \{b\}(\emptyset) & \{a, b, c\}(\{c\}) \\ \{b\}(\{b\}) & \{a, b, c\}(\{b, c\}) \\ \{b, d\}(\{d\}) & \{a, b, c, d\}(\{c, d\}) \\ \{b, d\}(\{b, d\}) & \{a, b, c, d\}(\{b, c, d\}) \end{array}$$

Remarkably only two of such models are minimal admissible models, that is:

$$\mu AM(P) \ = \ \{\{b\}(\emptyset), \{a, b, c\}(\{c\})\}.$$

Indeed the model $\{b\}(\{b\})$ is redundant w.r.t. $\{b\}(\emptyset)$ since $LHM(P \cup \{b\}) = LHM(P)$. In other words the addition of the hypothesis b does not add any

new conclusion. The models $\{b,d\}(\{d\})$ and $\{b,d\}(\{b,d\})$ are redundant w.r.t. $\{b\}(\emptyset)$ too, since in both cases $LHM(P \cup H) - LHM(P) = \{d\} \not\supset \{d\}$. Again, the addition of the set of hypotheses $\{d\}$ or $\{b,d\}$ does not add any other conclusion besides the hypotheses themselves. Similar considerations apply to the other non-minimal admissible models $\{a,b,c\}(\{b,c\})$, $\{a,b,c,d\}(\{c,d\})$, and $\{a,b,c,d\}(\{b,c,d\})$ which are all redundant w.r.t. $\{a,b,c\}(\{c\})$. $\Diamond$

The following proposition shows that for each admissible model $I(H)$ there exists a minimal admissible model $J(K)$ supported by a smaller set of hypotheses and such that $LHM(P \cup H) = LHM(P \cup K) \cup (H - K)$.

Proposition 1. *Let P be a program, let $I \subseteq \mathcal{B}$ and let $H \subseteq \mathcal{B}$.*

$$I(H) \in AM(P) \implies \exists K, J : (K \subseteq H \wedge J(K) \in \mu AM(P) \wedge I = J \cup (H - K)).$$

We finally prove that the equivalence relation $\equiv_{\mu AM}$, induced by the set of minimal admissible models of a program, does coincide with the fully abstract equivalence relation $\equiv_M$. To simplify the equivalence proof, we first provide the following alternative characterization of logical equivalence.

Lemma 2. *Let P and Q be two programs. Then:*

$$P \equiv_M Q \iff \forall H \subseteq \mathcal{B} : LHM(P \cup H) = LHM(Q \cup H).$$

We are now ready to establish that the equivalence relations $\equiv_{\mu AM}$ and $\equiv_M$ coincide.

Proposition 2. $\equiv_M = \equiv_{\mu AM}$.

To conclude our analysis of admissible models, let us reconsider Example 9 to illustrate the relation between Herbrand and (minimal) admissible models.

Example 10. Consider again the program:

P
$a \leftarrow b, c$
$b \leftarrow$
$c \leftarrow d, c$

If $\mathcal{B} = \{a,b,c,d\}$ then the set of Herbrand, admissible and minimal admissible models of program P are, respectively:

$HM(P)$	$AM(P)$	$\mu AM(P)$
$\{b\}$	$\{b\}(\emptyset)$	$\{b\}(\emptyset)$
	$\{b\}(\{b\})$	
$\{a,b\}$		
$\{b,d\}$	$\{b,d\}(\{d\})$	
	$\{b,d\}(\{b,d\})$	
$\{a,b,c\}$	$\{a,b,c\}(\{c\})$	$\{a,b,c\}(\{c\})$
	$\{a,b,c\}(\{b,c\})$	
$\{a,b,d\}$		
$\{a,b,c,d\}$	$\{a,b,c,d\}(\{c,d\})$	
	$\{a,b,c,d\}(\{b,c,d\})$	

It is worth noting that since $\mu AM(P) = \{\{b\}(\emptyset), \{a,b,c\}(\{c\})\}$ then the minimal admissible model semantics (correctly) identifies the above program P for instance with the program:

Q
$a \leftarrow c$
$b \leftarrow$

whose Herbrand, admissible, and minimal admissible models are, respectively:

$HM(Q)$	$AM(Q)$	$\mu AM(Q)$
$\{b\}$	$\{b\}(\emptyset)$	$\{b\}(\emptyset)$
$\{a,b\}$		
$\{b,d\}$		
$\{a,b,c\}$	$\{a,b,c\}(\{c\})$	$\{a,b,c\}(\{c\})$
$\{a,b,d\}$		
$\{a,b,c,d\}$		

We see that while the admissible model semantics distinguishes P and Q, the two programs are identified both by the minimal admissible model semantics ($\mu AM(P) = \mu AM(Q)$) and by logical equivalence ($HM(P) = HM(Q)$). $\Diamond$

3.6 Supported interpretations

We finally introduce an alternative characterization of logical equivalence, which is defined by means of the notion of *supported interpretation* originally introduced in [9].

Besides providing another equivalent formulation of logical equivalence, the notion of supported interpretation will be exploited in the following sections to define a compositional semantics for extended logic programs, such as programs containing negation.

Intuitively speaking, a (Herbrand) interpretation I for a program P is supported by a set of hypotheses H if I is the least Herbrand model of the program P extended with H. More precisely, I is an interpretation for P supported by H if and only if I is the least Herbrand model of the program $P \cup H$, where $P \cup H$ stands for $P \cup \{h \leftarrow \mid h \in H\}$.

If the sets of hypotheses to be considered are arbitrary subsets of the Herbrand base $\mathcal{B}$, we obtain the following definition of supported interpretation:

> *Let P be a program, and let $I \subseteq \mathcal{B}$, $H \subseteq \mathcal{B}$. Then I is an interpretation for P* supported *by H if and only if $I = LHM(P \cup H)$.*

An interpretation I supported by a set of hypotheses H is denoted by $I(H)$ to explicitly record the set of hypotheses supporting it.

The set of supported interpretations for a program P is then defined as follows:

$$SI(P, \mathcal{B}) = \{I(H) \mid H \subseteq \mathcal{B} \wedge I = LHM(P \cup H)\}.$$

As pointed out in [9], the above notion of supportedness properly extends the notion of admissibility introduced in Section 3.4. Namely the admissible models of a program are in general a subset of its supported interpretations, that is, $AM(P) \subseteq SI(P, \mathcal{B})$.

Lemma 1 shows that there is a direct correspondence between the supported interpretations and the Herbrand models of a program. Namely for each program P:

$$I \in M(P) \iff \exists H : I(H) \in SI(P, \mathcal{B}).$$

Let us denote by $\equiv_{SI(\mathcal{B})}$ the equivalence relation induced by the set of supported interpretations of a program, that is:

$$P \equiv_{SI(\mathcal{B})} Q \iff SI(P, \mathcal{B}) = SI(Q, \mathcal{B}).$$

We now establish that the equivalence relation $\equiv_{SI(\mathcal{B})}$ coincides with logical equivalence $\equiv_M$.

Proposition 3. $\equiv_M = \equiv_{SI(\mathcal{B})}$.

In the following sections, we will use a more general definition of supported interpretation by considering a set $\mathcal{H}$ of *assumable hypotheses*, where $\mathcal{H}$ is some pre-defined subset of the Herbrand base $\mathcal{B}$ (viz., $\mathcal{H} \subseteq \mathcal{B}$). Namely the set of supported interpretations of a program P w.r.t. a set of assumable hypotheses $\mathcal{H}$ is defined as follows:

$$SI(P, \mathcal{H}) = \{I(H) \mid H \subseteq \mathcal{H} \wedge I = LHM(P \cup H)\}.$$

Notice that the choice of the set of assumable hypotheses affects the induced equivalence relations as pointed out by the following proposition.

Proposition 4. *Let $\mathcal{H} \subseteq \mathcal{B}$, $\mathcal{K} \subseteq \mathcal{B}$ be two sets of assumable hypotheses. Then:*

(1) $\mathcal{H} \subseteq \mathcal{K} \implies \equiv_{SI(\mathcal{K})} \subseteq \equiv_{SI(\mathcal{H})}$
(2) $\mathcal{H} \subset \mathcal{K} \implies \equiv_{SI(\mathcal{K})} \subset \equiv_{SI(\mathcal{H})}$

3.7 Summary

In the previous sections, we analysed different compositional semantics for definite logic programs that have been proposed in the literature. The relations between the semantics considered are summarized in Figure 2.

We have first considered three equivalence relations analyzed in [30]: subsumption equivalence ($\equiv_T$), weak subsumption equivalence ($\equiv_{T+Id}$), and logical equivalence ($\equiv_M$). We have shown that they are all compositional w.r.t. $(\mathcal{O}, \{\cup\})$, and while the first two semantics are not fully abstract w.r.t. $(\mathcal{O}, \{\cup\})$, logical equivalence is the fully abstract compositional equivalence w.r.t. $(\mathcal{O}, \{\cup\})$.

We have then considered a different equivalence relation ($\equiv_{AM}$) defined in terms of the admissible Herbrand models of a program. We have shown that while admissible models induce a compositional denotation of programs, the corresponding equivalence relation is not fully abstract w.r.t. $(\mathcal{O}, \{\cup\})$. We have

then shown how the notion of admissible model can be suitably constrained so as to obtain a fully abstract denotation of programs. The new equivalence relation $\equiv_{\mu AM}$, induced by the so-called minimal admissible models of a program, coincides with the fully abstract equivalence relation $\equiv_M$.

Finally, we have introduced the notion of supported interpretation that will be used in the following sections. We have shown that supported interpretations provide an alternative definition of logical equivalence, that is, the induced equivalence relation $\equiv_{SI(\mathcal{B})}$ coincides with the fully abstract compositional equivalence relation $\equiv_M$.

The relations between the semantics considered are summarized in Figure 2, where a double arrow from $\equiv_A$ to $\equiv_B$ denotes that $\equiv_A$ coincides with $\equiv_B$, while a single arrow from $\equiv_A$ to $\equiv_B$ denotes that $\equiv_A$ is strictly finer than $\equiv_B$, that is, $\equiv_A \subset \equiv_B$.

$$
\begin{array}{ccccc}
 & & T(P) = T(Q) & & \\
 & & \downarrow & & \\
 & & T(P) + Id = T(Q) + Id) & & AM(P) = AM(Q) \\
 & & \downarrow & & \downarrow \\
SI(P,\mathcal{B}) = SI(Q,\mathcal{B}) & \longleftrightarrow & M(P) = M(Q) & \longleftrightarrow & \mu AM(P) = \mu AM(Q) \\
 & & \downarrow & & \\
 & & LHM(P) = LHM(Q) & &
\end{array}
$$

Fig. 2. The new equivalence hierarchy.

The chain of inclusions $\equiv_T \subset \equiv_{T+Id} \subset \equiv_M \subset \equiv_{LHM}$ was established in [30].

The strict inclusion $\equiv_{AM} \subset \equiv \mu AM$ has been established in Section 3.5. Indeed the minimal admissible models $\mu AM(P)$ of a program P are obtained from the admissible models $AM(P)$ of P. It is hence easy to show that $\equiv_{AM} \subseteq \equiv_{\mu AM}$, that is, if $P \equiv_{AM} Q$ then $P \equiv_{\mu AM} Q$. Moreover, as shown in Section 3.5, $\equiv_{\mu AM} \not\subseteq \equiv_{AM}$, that is, there exist programs that have the same set of minimal admissible models, while they have different sets of admissible models. For instance the empty program and the program P of Example 8 are distinguished by the admissible models semantics, since $AM(P) = \{\emptyset(\emptyset), \{a\}(\{a\})\}$, while $\emptyset(\emptyset)$ is the only admissible model for the empty program. On the other hand the two programs are identified by the minimal admissible model semantics in that $\emptyset(\emptyset)$ is the only minimal admissible model for both programs.

It is worth noting that while $\equiv_{AM}$ is strictly finer than $\equiv_{\mu AM}$ (and hence than $\equiv_M$ and $\equiv_{SI(\mathcal{B})}$), $\equiv_{AM}$ is not comparable with either $\equiv_T$ or $\equiv_{T+Id}$. Indeed there are programs which are (weak) subsumption equivalent and not admissible interpretations equivalent, and vice-versa.

Example 11. For instance consider the programs:

P	Q
$a \leftarrow$	$a \leftarrow$
	$a \leftarrow a$

We observe that $P \equiv_T Q$ (since $\forall I$: $T(P)(I) = \{a\} = T(Q)(I)$) and $P \equiv_{T+Id} Q$ (since $\forall I$: $(T + Id)(P)(I) = \{a\} \cup I = (T + Id)(Q)(I)$), while $P \not\equiv_{AM} Q$ since $\{a\}(\{a\}) \in AM(Q) - AM(P)$. On the other hand, if we consider the programs:

P	Q
$a \leftarrow$	$a \leftarrow b$
$b \leftarrow$	$b \leftarrow$
$c \leftarrow b$	$c \leftarrow b$

we see that $P \equiv_{AM} Q$ since $AM(P) = AM(Q) = \{\{a, b, c\}(\emptyset), \{a, b, c\}(\{b\})\}$, while $P \not\equiv_T Q$ and $P \not\equiv_{T+Id} Q$ (since $T(P)(\emptyset) = \{a, b\}$ and $T(Q)(\emptyset) = \{b\}$). $\Diamond$

In this section, we have examined a number of compositional semantics for definite programs. However, as already anticipated in the Introduction, our analysis is not intended to be exhaustive. Other semantics have been proposed in the literature, such as [27, 33] which were then extended by [31]. A survey describing these and other modular extensions of logic programming can be found in [14].

Moreover, our analysis focusses on standard Herbrand interpretations [2, 29]. Many efforts have been devoted to define compositional semantics by using extended interpretations containing possibly non-ground atoms (e.g., see [6, 22]). The relation between these semantics and the semantics based on standard Herbrand interpretations is summarised in [1].

4 Composition of normal programs

The need of extending definite programs to deal with forms of non-monotonic reasoning was recognized since the early years of logic programming. Negation as failure was introduced in [15] to express negative information, and it has been shown to support various forms of non-monotonic reasoning. A number of other extensions have been then proposed to further enrich the expressive power of logic programming as a general formalism for knowledge representation (see [4] for a survey).

The formalization of these extensions has called for new semantics capable to capture their "intended" meaning. Even for the case of negation as failure, many different characterizations have been defined from different perspectives, most of them inspired by an interpretation of negation as failure as a more general notion of negation by default (e.g., [19]). A survey of the semantics of logic programs with negation (by default) is reported in [3].

Something similar happened for other extensions such as abductive logic programming (e.g., [17, 19, 26]) and logic programming with a second form of negation in addition to negation by default (e.g., [24, 34, 38]).

For each extension there is no general agreement on what its semantics should be. Formal comparisons among different semantics are hard to be drawn, mainly

because they are often based on different grounds. Furthermore, many proposals are based on a proof-theoretic approach rather than on a model-theoretic approach, and this constitutes a further obstacle for the study of formal properties, and hence formal comparisons, of different proposals.

On these premises, analysing compositionality issues in extended logic programs seems to be a quite difficult enterprise since:

- Many semantics have been proposed for different extensions of logic programming, and there is no general agreement on the intended meaning of each extension.
- Non-monotonic reasoning and compositionality are intuitively orthogonal issues that do not seem easy to be reconciled. Indeed the semantics for extended logic programs are typically non-compositional w.r.t. program union.

Consider for instance the case of normal logic programs, that is, logic programs with negation as default. It is easy to show, for instance, that the stable model semantics [23] is not compositional with respect to the union of programs.

Example 12. Consider for instance the programs:

P	Q
$a \leftarrow$	$a \leftarrow not\ b$

which have the same (unique) stable model $\{a\}$. If these programs are extended with the program:

R
$b \leftarrow$

we obtain two programs which have different stable models ($\{a, b\}$ and $\{b\}$, respectively). Therefore it is not possible, in general, to determine the stable models of a program from the stable models of its clauses. $\Diamond$

A unifying view of different extensions of logic programming was presented in [9], where it is shown how the meaning of various extensions of logic programming can be expressed by means of the supported interpretations of a program. Many extensions are considered in [9], including negation-by-default, other forms of negation and abduction.

The approach can be summarised as follows:

1. Given an extended logic program, construct its *"positive" version.*
2. Consider the set of supported interpretations of (the positive version of) the program.
3. Select among the supported interpretations the *complete* models which characterise the intended meaning of a program.

The interest of complete models derives from their correspondence relation with other models proposed in the literature. Such a correspondence makes the supported interpretation approach a general framework for characterising and comparing different semantics of various extensions of logic programming.

In this paper we will focus only on one of these extensions, negation-by-default, and on the corresponding class of normal programs.

4.1 Positive version of a normal program

A *normal program* P is a set of clauses of the form

$$A \leftarrow L_1, \ldots, L_n \quad (n \geq 0)$$

where A is an atom and $L_1, \ldots, L_n$ are literals. Negated literals in clause bodies have the form *not* B, where B is an atom. In the following, without loss of generality, we will consider only (possibly infinite) propositional programs. A non-propositional program is then understood as a shorthand for the (possibly infinite) set of ground clauses obtained by instantiating the original rules in all possible ways over the Herbrand universe.

Following [18, 38], the *positive version* P^+ of a normal program P is the definite program obtained by replacing in P each negated atom *not* A by a new positive atom not_A. The Herbrand base $\mathcal{B}^+$ associated with P^+ is then the set obtained by extending the Herbrand base $\mathcal{B}$ of P with the new set of atoms:

$$not\_\mathcal{B} = \{not\_A \mid A \in \mathcal{B}\}$$

that is:

$$\mathcal{B}^+ = (\mathcal{B} \cup not\_\mathcal{B}).$$

The intended meaning of P will be defined by suitably restricting the Herbrand models of P^+, which are subsets of the extended Herbrand base.

From now onward, we will not distinguish any further between a normal program P and its positive version P^+, that is we will denote P directly by its positive version.

4.2 Supported interpretations for normal programs

Let us take the negative part $not\_\mathcal{B}$ of the extended Herbrand base $\mathcal{B}^+$ as the set of assumable hypotheses. We get the following definition of supported interpretations of a program P:

$$SI(P, not\_\mathcal{B}) = \{I(H) \mid H \subseteq not\_\mathcal{B} \wedge I = LHM(P \cup H)\}.$$

As shown in [9], the set of supported interpretations can be suitably restricted in a step-wise way so as to identify the set of complete models of a program. Such a step-wise process can be summarised as follows:

- A supported interpretation $I(H)$ of a program P is a supported *model* of P if I is consistent (i.e., $\nexists A : A \in I \wedge not\_A \in I$).
- A supported interpretation $J(K)$ is a *conservative extension* of a supported model $I(H)$ if and only if $K \supseteq H$ and $(\nexists A : not\_A \in K \wedge A \in I)$.
- A supported model $I(H)$ is a *complete model* of P if and only if $\forall a \in \mathcal{B}$:

$$not\_a \in H \iff (a \notin J \text{ for each conservative extension } J(K) \text{ of } I(H)).$$

- Each normal programs has at least one complete model.

Example 13. Consider for instance the program:

$$P$$
$$a \leftarrow not\ b$$

and suppose for the sake of simplicity that $\mathcal{B} = \{a, b\}$. Then the supported interpretations $SI(P, not\_\mathcal{B})$ are:

$$\emptyset\ (\emptyset)$$
$$\{not\_a\}\ (\{not\_a\})$$
$$\{a, not\_b\}\ (\{not\_b\})$$
$$\{a, not\_a, not\_b\}\ (\{not\_a, not\_b\})$$

It is easy to observe that each supported interpretation of P is a conservative extension of the model supported by the empty set of hypotheses. On the other hand, there is no conservative extension of the model $\{a, not\_b\}(\{not\_b\})$ since its only extension $\{a, not\_a, not\_b\}(\{not\_a, not\_b\})$ is not conservative with it (viz., not_a is an hypothesis of the latter while a belongs to the model of the former). We can therefore observe that $\{a, not\_b\}(\{not\_b\})$ is the only complete model for P. $\Diamond$

Notably these complete models have a tight relation with other models proposed in the literature. In the case of normal programs, complete models have been shown in [9] to correspond to:

- stable models semantics [23],
- well-founded semantics [41],
- stationary semantics [38], and
- preferential semantics [17].

For instance, in [9], it is shown that:

> *Let P be a normal program and let $M \subseteq \mathcal{B}$. Then M is a stable model of P if and only if $M \cup \{not\_A \mid A \notin M\}$ is a total complete model for P. (An interpretation M is total iff for each $A \in \mathcal{B}$: $A \in M$ or $not\_A \in M$.)*

Similar correspondences are established with well-founded models [41], with stationary expansions [38], and with complete scenaria [17].

These correspondences can be equivalently described in the following way. For each semantics $\mathcal{S}$ considered, there exists a suitable *projection* function $\psi_{\mathcal{S}}$ which, given the set of supported interpretations $SI(P^+, not\_\mathcal{B})$ of (the positive version of) a program P, yields the models $\mathcal{S}(P)$ of the corresponding semantics $\mathcal{S}$

$$\psi_{\mathcal{S}} : SI(P^+, not\_\mathcal{B}) \rightarrow \mathcal{S}(P).$$

In terms of program equivalence, this means that the supported interpretations semantics preserves all the semantics which have been considered in [9]. Indeed for any such semantics $\mathcal{S}$:

$$P^+ \equiv_{SI(not\_\mathcal{B})} Q^+ \implies P \equiv_{\mathcal{S}} Q.$$

4.3 Compositionality

As the supported interpretations semantics preserves many different meanings of programs, the compositionality of the induced equivalence relation $\equiv_{SI(not\_\mathcal{B})}$ would be of major importance. Indeed it would allow the substitution of parts of normal programs without affecting the meaning of the whole program, for *any* meaning considered.

Unfortunately, the equivalence relation $\equiv_{SI(not\_\mathcal{B})}$ is not a congruence w.r.t. the union of programs.

Example 14. Consider for instance the program

P
$a \leftarrow b$

and let Q be the empty program. While $P \equiv_{SI(not\_\mathcal{B})} Q$ there exists a program R such that $P \cup R \not\equiv_{SI(not\_\mathcal{B})} Q \cup R$. For instance, consider the program

R
$b \leftarrow not\ c$

It is easy to see that $SI(P \cup R, not\_\mathcal{B}) \neq SI(Q \cup R, not\_\mathcal{B})$ since the supported interpretations $\{a, b, not\_c\}(\{not\_c\})$ belongs to the former and not the latter. $\Diamond$

By exploiting Proposition 4 it is however possible to establish a compositionality result for normal programs. Indeed since $not\_\mathcal{B} \subset \mathcal{B}^+$ we have that:

$$\equiv_{SI(\mathcal{B}^+)} \subset \equiv_{SI(not\_\mathcal{B})}$$

and by the compositionality of logical equivalence we obtain the following result.

Proposition 5. $\equiv_{SI(\mathcal{B}^+)}$ *is compositional for* $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$.

Since the supported interpretations equivalence relation $\equiv_{SI(not\_\mathcal{B})}$ preserves all the semantics for normal programs considered in [9], we have that if (the positive versions of) two programs are logically equivalent then they have the same meaning $\mathcal{S}$ for each $\mathcal{S}$ considered in [9], that is they have the same complete scenaria [17], the same stationary expansions [38], the same stable models [23], and the same well-founded model [41].

Corollary 1. $\equiv_{SI(\mathcal{B}^+)}$ *is compositional for* $(\equiv_{\mathcal{S}}, \{\cup\})$*, for all* $\mathcal{S}$ *considered in [9].*

4.4 Full abstraction

The compositionality of logical equivalence for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$ states that *if* (the positive version of) two programs R and R' are logically equivalent then they can be exchanged with one another without affecting the meaning $\mathcal{S}$ of the context in which they occur.

An intriguing question is whether or not this is the largest class of programs which can be substituted one another without affecting the intended meaning $\mathcal{S}$ of the context in which they occur.

As the equivalence relation $\equiv_{SI(not\_\mathcal{B})}$ preserves several semantics for normal programs, it is interesting to determine whether logical equivalence is fully abstract for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$. This is exactly what we establish here for the first time with the following proposition.

Proposition 6. *$\equiv_{SI(\mathcal{B}^+)}$ is fully abstract for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$.*

It is worth noting that the above result states that logical equivalence defines the largest class of programs which can be substituted one another without affecting the set of negatively supported interpretations of the context in which they occur. The full abstraction result is hence relative to the equivalence relation $\equiv_{SI(not\_\mathcal{B})}$, and its importance is due to the fact that $\equiv_{SI(not\_\mathcal{B})}$ preserves a number of different intended semantics for normal programs. On the other hand, Proposition 6 does not imply that logical equivalence is fully abstract for every intended meaning of normal programs. For instance, an equivalence relation coarser than logical equivalence (of the positive versions) may be fully abstract for the stable model semantics of normal programs.

4.5 Summary

In the previous sections, we have discussed the issue of compositionality for the case of normal logic programs. As we observed at the beginning of Section 4, the two main problems of designing of a tour for analyzing compositionality issues in normal logic programs were:

(1) the existence of many different semantics for normal programs (no universal agreement on the intended meaning of a normal program, as it happens instead for the case for definite programs), and
(2) the orthogonality of negation and compositionality (existing semantics for normal programs are typically not compositional).

Following the steps of [9], we have observed that supported interpretations provide a unifying model-theoretic characterizations of a number of semantics for normal programs. The idea is to consider the positive version of programs and to take $not\_\mathcal{B}$ as the universe of assumable hypotheses. The induced equivalence relation $\equiv_{SI(not\_\mathcal{B})}$ then preserves the stable models semantics [23], the well-founded semantics [41], the preferential semantics [17], and the stationary semantics [38]. While $\equiv_{SI(not\_\mathcal{B})}$ is not a congruence for the union of programs, the equivalence relation $\equiv_{SI(\mathcal{B}^+)}$ is compositional for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$, and hence $\equiv_{SI(\mathcal{B}^+)}$ is compositional for $(\equiv_{\mathcal{S}_i}, \{\cup\})$, for each semantics $\mathcal{S}_i$ preserved by $\equiv_{SI(not\_\mathcal{B})}$. Finally, as a new result, we have shown that $\equiv_{SI(\mathcal{B}^+)}$ is the fully abstract equivalence relation for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$. The relations between the semantics considered in this section are summarised in Figure 3.

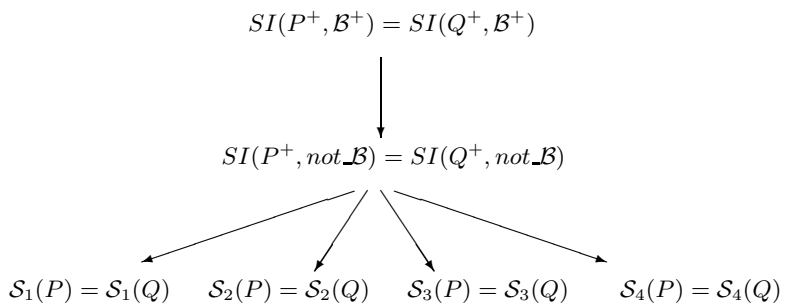

Fig. 3. The equivalence hierarchy for normal programs, where $\mathcal{S}_1$—$\mathcal{S}_4$ are the semantics for normal programs considered in [9].

It is worth observing that the notion of supported interpretation can be exploited to provide a compositional characterization of other extensions of logic programming, besides negation-by-default. Two main classes of extensions of logic programming are considered in [9]:

- *Other forms of negation* — such as explicit negation [34], answer set semantics [24], and 3-valued stable semantics [37];
- *Abduction* — as modeled for instance in [16] and in [26].

As shown in [9], all these extensions can be provided with a uniform characterization by means of supported interpretations by considering different universes of assumable hypotheses. Lack of space forces us to invite the reader to refer to [9] for more details.

Several other efforts have been devoted to investigate the composition of normal logic programs.

The *splitting* of a logic program into parts was investigated in [28] in the context of the answer set semantics [24]. The basic idea is that, in many cases, a program can be divided into a "bottom" part and a "top" part, such that the former does not refer to predicates defined in the latter. In [28] it is shown that computing the answer sets for a program can be simplified when the program is split into parts. It is also shown that the idea of splitting can be applied for proving properties of simple program compositions, like a conservative extension property for a program P extended by rules whose heads do not occur in P.

The problem of defining compositional semantics for normal logic programs was studied in [20] and in [42]. [20] defines a compositional semantics for normal programs by means of a first-order unfolding operator. Such a semantics is

applied for composing "open" normal programs by considering the union of programs as the only composition operation. [42] presents several results on the compositionality of normal logic programs by generalising the well-founded semantics of logic programming. More precisely, they identify some conditions under which the class of extended well-founded models of the union of two normal programs coincides with the intersection of the classes of extended well-founded models of the two programs. Another difference between the approaches presented in [20, 42] and ours is the restrictive naming policy imposed on the predicate names of the programs to be composed. Namely, in contrast with our naming policy, predicate definitions cannot be spread over different programs.

An even more restrictive naming policy is considered in [13] for defining a compositional model-theory for definite and disjunctive programs. The author investigates the usability of minimal logic for modeling the meaning of extended logic programs by allowing "local" inconsistencies. The compositionality results are however restricted to pairs of programs which define disjoint sets of predicates and which do not interact with each other.

A compositional semantics for normal programs was also defined in [8], where a family of program composition operations is considered. The semantics of programs and program compositions is defined in terms of three-valued logic by extending the three-valued immediate consequence operator for logic programs proposed in [21].

Finally, a related work is [5], where a set of operations for composing logic programs are studied in the context of *intensional negation*. Negation is handled in a constructive way by transforming normal programs into pairs of definite programs and by defining the composition operations on such pairs.

5 Other forms of program compositions

5.1 Program extension

The operation of union between programs can be employed to combine programs that fully cooperate in the deduction process. Namely one program may exploit partial conclusions of the other, and vice-versa.

Such a symmetric composition-by-union is not however the only form of program composition employed in program development. For instance, programs often import the functionalities of some existing module or library, where the latter does not in turn rely on the former for its computation [28].

A typical example comes from deductive databases, where a set of (recursive) rules R is composed with an extensional database D defining the values of a set of relations and typically consisting of extensional clauses. An intriguing question is when R can be substituted with a different, possibly more efficient, set of rules R' without affecting the meaning of the whole system.

Let us then consider here this form of program composition, that we call *program extension* to distinguish it from the composition-by-union considered in the previous sections. Formally, rather than considering arbitrary expressions of

the form:

$$Exp \ :== \ P \ \mid \ Exp \cup Exp$$

where P is a definite program, we now consider restricted compositions of the form:

$$Exp \ :== \ P \ \mid \ Exp \cup_\pi D$$

where D is an *extension*, that is, an extensional program consisting of unit clauses only. We may also assume that the predicates defined by extensions are all members of a pre-fixed set of predicates π. The symbol $\cup_\pi$ is hence both to record the set π and to distinguish program extension from the general composition-by-union considered in the previous sections. According to the above syntax, we therefore consider expressions of the form:

$$P, \quad P \cup_\pi D_1, \quad (P \cup_\pi D_1) \cup_\pi D_2, \quad ((P \cup_\pi D_1) \cup_\pi D_2) \cup_\pi D_3, \quad \ldots$$

and so on and so forth. Notice that instead of introducing the new symbol $\cup_\pi$ we may simply consider the union operation $\cup$ as a partial composition function which is defined only when its second argument is an extension.

We now show that supported interpretations can be naturally employed to obtain a compositional denotation of programs also for this new form of composition-by-extension. Consider as the set of assumable hypotheses the following set:

$$\mathcal{B}_\pi \ = \ \{A \mid A \in \mathcal{B} \wedge pred(A) \in \pi\}.$$

where $pred(A)$ denotes the predicate symbol of the atom A. Namely $\mathcal{B}_\pi$ is the set of atoms in the Herbrand base with predicates in the set π. The corresponding set of supported interpretations therefore is:

$$SI(P, \mathcal{B}_\pi) \ = \ \{I(H) \mid H \subseteq \mathcal{B}_\pi \wedge I = LHM(P \cup H)\}$$

and the induced equivalence relation is:

$$P \equiv_{SI(\mathcal{B}_\pi)} Q \iff SI(P, \mathcal{B}_\pi) = SI(Q, \mathcal{B}_\pi).$$

We observe that the equivalence relation $\equiv_{SI(\mathcal{B}_\pi)}$ is strictly coarser than $\equiv_{SI(\mathcal{B})}$ and strictly finer than $\equiv_{LHM}$.

Proposition 7. $\equiv_{SI(\mathcal{B})} \subset \equiv_{SI(\mathcal{B}_\pi)} \subset \equiv_{LHM}$.

Moreover, the equivalence relation $\equiv_{SI(\mathcal{B}_\pi)}$ is compositional for $(\mathcal{O}, \{\cup_\pi\})$, as stated by the following proposition.

Proposition 8. $\equiv_{SI(\mathcal{B}_\pi)}$ *is compositional w.r.t* $(\mathcal{O}, \{\cup_\pi\})$.

It is perhaps even more interesting to observe that supported interpretations can be employed to define a compositional characterisation of program extension also in the case of *normal* programs.

Indeed, consider expressions of the form:

$$Exp \ :== \ P \ \mid \ Exp \cup_\pi D$$

where P is a normal program and D is an extension, that is, an extensional program consisting of unit clauses only.

Then we simply consider as the set of assumable hypotheses the set:

$$\mathcal{B}_\pi \ \cup \ not\_\mathcal{B}$$

The corresponding set of supported interpretations therefore is:

$$SI(P, \mathcal{B}_\pi \cup not\_\mathcal{B}) \ = \ \{I(H) \mid H \subseteq (\mathcal{B}_\pi \cup not\_\mathcal{B}) \wedge I = LHM(P \cup H)\}$$

and the induced equivalence relation is:

$$P \equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})} Q \iff SI(P, \mathcal{B}_\pi \cup not\_\mathcal{B}) = SI(Q, \mathcal{B}_\pi \cup not\_\mathcal{B}).$$

It is easy to observe that $\equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})}$ is strictly coarser than $\equiv_{SI(\mathcal{B}^+)}$ and strictly finer than $\equiv_{SI(not\_\mathcal{B})}$.

Proposition 9. $\equiv_{SI(\mathcal{B}^+)} \subset \equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})} \subset \equiv_{SI(not\_\mathcal{B})}$.

Moreover the equivalence relation $\equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})}$ is compositional for $(\equiv_{SI(not\_\mathcal{B})}, \{\cup_\pi\})$, as proved by the following proposition.

Proposition 10. $\equiv_{SI(\mathcal{B}_\pi \cup not\_HB)}$ *is compositional for* $(\equiv_{SI(not\_\mathcal{B})}, \{\cup_\pi\})$.

5.2 A family of program composition operations

A family of program composition operations has been studied in [7]. Besides the union operation, other three main composition operations have been considered: *intersection* ($\cap$), *encapsulation* ($*$), and *import* ($\lhd$). These operations are defined in a semantics-driven style, following the observation that if the meaning of a program P is denoted by the corresponding immediate consequence operator $T(P)$, then such a meaning is a homomorphism for several interesting operations on programs. The formal semantics of the operations is defined as follows:

$$\begin{aligned} T(P \cap Q)(I) &= T(P)(I) \ \cap \ T(Q)(I) \\ T(P^*)(I) &= T^\omega(P)(\emptyset) \\ T(P \lhd Q)(I) &= T(P) \ (I \cup T^\omega(Q)(\emptyset)) \end{aligned}$$

Such a set of basic composition operations forms an algebra of logic programs with interesting properties for reasoning about programs and program compositions. From a programming perspective, the operations enhance the expressive power of logic programming by supporting a wealth of programming techniques, ranging from software engineering to artificial intelligence applications.

A thorough discussion of the family of composition operations is outside the scope of the present paper. It is however worth mentioning here that [12] showed that the chain of equivalence relations $\equiv_T \subset \equiv_{T+Id} \subset \equiv_M \subset \equiv_{LHM}$ (see Figure 2) coincides with the chain of fully abstract compositional equivalence relations for subsets of a family of program composition operations, as summarized in Figure 4.

	$\{*, \cup, \cap, \lhd\}$	$\{*, \cup, \cap\}$	$\{*, \cup\}$	$\{*\}$
$\equiv_T$	FAC	C	C	C
$\equiv_{T+Id}$	-	FAC	C	C
$\equiv_M$	-	-	FAC	C
$\equiv_{LHM}$	-	-	-	FAC

Fig. 4. The chain of fully abstract compositional equivalence relations. ("C" stands for compositional, "FAC" for fully abstract and compositional, while "-" stands for non-compositional.)

6 Concluding remarks

As we already mentioned in the Introduction, our analysis of compositional semantics for logic programs was not intended to be exhaustive, in the sense of analysing all the (many) different proposals that have been developed over the last ten years.

We have rather tried to guide the reader across different compositional semantics for logic programs by highlighting the existing relations among them, and by establishing new relations and results on our way.

During our tour, the notion of supported interpretation has been shown to provide a general and unifying mechanism for obtaining compositional denotations of both definite and normal programs, also in the case of asymmetric compositions.

Acknowledgements

I would like to thank here all the persons with whom I have had the pleasure of sharing many stimulating discussions on these topics. In particular, I would like to thank Simone Contiero, Evelina Lamma, Paolo Mancarella, Paola Mello, Dino Pedreschi, and Franco Turini, with whom I have had the pleasure of working together for many years.

References

1. K. Apt, M. Gabbrielli, and D. Pedreschi. A closer look to declarative interpretations. *Journal of Logic Programming*, 28(2):147–180, 1996.
2. K. R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 493–574. Elsevier, 1990. Vol. B.
3. K. R. Apt and R. Bol. Logic Programming and Negation: A Survey. *Journal of Logic Programming*, 19-20:9–71, 1994.
4. C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19-20:73–148, 1994.
5. R. Barbuti, P. Mancarella, D. Pedreschi, and F. Turini. A Transformational Approach to Negation in Logic Programming. *Journal of Logic Programming*, 8:201–228, 1990.

6. A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19-20:149–197, 1994.
7. A. Brogi. *Program Construction in Computational Logic*. PhD thesis, Department of Computer Science, University of Pisa, 1993.
8. A. Brogi, S. Contiero, and F. Turini. Programming by composing general logic programs. *Journal of Logic and Computation*, 9(1):7–24, 1999.
9. A. Brogi, E. Lamma, P. Mancarella, and P. Mello. A unifying view for logic programming with non-monotonic reasoning. *Theoretical Computer Science*, 184(1):1–59, 1997.
10. A. Brogi, E. Lamma, and P. Mello. Compositional Model-theoretic Semantics for Logic Programs. *New Generation Computing*, 11(1):1–21, 1992.
11. A. Brogi, P. Mancarella, D. Pedreschi, and F. Turini. Composition Operators for Logic Theories. In J.W. Lloyd, editor, *Computational Logic, Symposium Proceedings*, pages 117–134. Springer-Verlag, 1990.
12. A. Brogi and F. Turini. Fully abstract compositional semantics for an algebra of logic programs. *Theoretical Computer Science*, 149(2):201–229, 1995.
13. F. Bry. A Compositional Semantics for Logic Programs and Deductive Databases. In M. Maher, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming*, pages 453–467, 1996.
14. M. Bugliesi, E. Lamma, and P. Mello. Modularity in logic programming. *Journal of Logic Programming*, 19,20:443–502, 1994.
15. K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum, 1978.
16. C.V. Damasio and L.M. Pereira. Abduction over 3-valued extended logic programs. In *Proc. 3rd Int. Workshop on Logic Programming and Non Monotonic Reasoning*, pages 29–42. Springer-Verlag, 1995.
17. P.M. Dung. Negation as hypothesis: An abductive foundation for logic programming. In K. Furukawa, editor, *Proc. 8th Int. Conf. on Logic Programming ICLP91*, pages 3–17. The MIT Press, 1991.
18. P.M. Dung and P. Ruamviboonsuk. Well-founded reasoning with classical negation. In *Proc. 1st Int. Workshop on Logic Programming and Non-Monotonic Reasoning*, pages 120–132. The MIT Press, 1991.
19. K. Eshghi and R.A. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *Proc. 6th Int. Conf. on Logic Programming ICLP89*, pages 234–254. The MIT Press, 1989.
20. S. Etalle and F. Teusink. A Compositional Semantics for Normal Open Programs. In M. Maher, editor, *JICSLP 96 Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 468–482, 1996.
21. M. Fitting. A Kriple-Kleene semantics for general logic programs. *Journal of Logic Programming*, 4:295–312, 1985.
22. M. Gabbrielli, G. Levi, and M. C. Meo. Observable behaviors and equivalences of logic programs. *Information and Computation*, pages 1–29, 1995.
23. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings of the Fifth International Conference and Symposium on Logic Programming, Seattle, 1988*, pages 1070–1080. The MIT Press, 1988.
24. M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D.H.D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference and Symposium on Logic Programming*, pages 579–597. The MIT Press, 1990.
25. G. Gottlob and A. Leitsch. On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2):280–295, 1985.

26. A.C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proceedings of 9th European Conference on Artificial Intelligence ECAI90*, pages 385–391. Pitman Publishing, 1990.
27. J.L. Lassez and M.J. Maher. Closures and fairness in the semantics of logic programming. *Theoretical Computer Science*, 29:167–184, 1984.
28. V. Lifschitz and H. Turner. Splitting a logic program. In P. Van Entenryck, editor, *Proc. 11th International Conference on Logic Programming*, pages 23–37. The MIT Press, 1994.
29. J.W. Lloyd. *Foundations of logic programming.* Springer-Verlag, second edition, 1987.
30. M.J. Maher. Equivalences of logic programs. In J. Minker, editor, *Deductive databases and logic programming*, pages 627–658. Morgan Kaufmann, 1988.
31. P. Mancarella and D. Pedreschi. An algebra of logic programs. In R. A. Kowalski and K. A. Bowen, editors, *Proceedings Fifth International Conference on Logic Programming*, pages 1006–1023. The MIT Press, 1988.
32. R. Milner. Fully abstract models for typed λ-calculi. *Theoretical Computer Science*, 4:1–23, 1977.
33. R. O'Keefe. Towards an algebra for constructing logic programs. In J. Cohen and J. Conery, editors, *Proceedings of IEEE Symposium on Logic Programming*, pages 152–160. IEEE Computer Society Press, 1985.
34. L.M. Pereira and J.J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proc. ECAI92*, pages 102–106. John Wiley and Sons, 1992.
35. A. Pettorossi and M. Proietti. Transformations of logic programs: Foundations and techniques. *Journal of Logic Programming*, 19 & 20:261–320, 1994.
36. G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–256, 1977.
37. T. Przymusinski. Extended stable semantics for normal and disjunctive programs. In D.H.D. Warren and P. Szeredi, editors, *Proceedings of the Seventh International Conference and Symposium on Logic Programming*, pages 459–477. The MIT Press, 1990.
38. T. Przymusinski. Stationary semantics for normal and disjunctive logic programs. In C. Delobel, M. Kifer, and Y. Masunagar, editors, *Proceedings of DOOD'91.* Springer-Verlag, 1991.
39. R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 153–173. Plenum, 1978.
40. M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.
41. A. van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
42. S. Verbaeten, M. Denecker, and D. De Schreye. Compositionality of normal open logic programs. *Journal of Logic Programming*, 41(3):151–183, 2000.

Appendix

This Appendix contains the proofs of the propositions and lemmas stated in the paper. To simplify the proofs, we will sometimes abuse notation and denote with the same symbol, I say, both an interpretation I and the program $\{h \leftarrow | \ h \in I\}$.

Lemma 1
Let P be a program and let $I \subseteq \mathcal{B}$. Then:

$$I \in M(P) \iff \exists H \subseteq \mathcal{B} : I = LHM(P \cup H).$$

Proof. (*if part*) If $I = LHM(P \cup H)$ then, by definition of least Herbrand model, $I \in M(P \cup H)$ and, by Observation 1, $I \in M(P)$.
(*only-if part*) We observe that if $I \in M(P)$ then $I = LHM(P \cup I)$. By definition of least Herbrand model and by Observation 1, we have that:
$LHM(P \cup I) = min\ \{J \mid J \in M(P) \wedge J \in M(I)\}$.
Then since $I \in M(P)$ and since $I = LHM(I)$, we have that: $LHM(P \cup I) = I$.

Lemma 2
Let P and Q be two programs. Then:

$$P \equiv_M Q \iff \forall H \subseteq \mathcal{B} : LHM(P \cup H) = LHM(Q \cup H)$$

Proof. (*if part*) If $I \in M(P)$ then by Lemma 1 there exists $H \subseteq \mathcal{B} : I = LHM(P \cup H)$. By hypothesis this implies that $I = LHM(Q \cup H)$ and hence, by Lemma 1, that $I \in M(Q)$.
(*only-if part*) By definition of least Herbrand model and by Observation 1:
$LHM(P \cup H) = min\{I \mid I \in M(P) \wedge I \in M(H)\}$.
Hence, since $M(P) = M(Q)$ by hypothesis, we have that $LHM(P \cup H) = LHM(Q \cup H)$ for any $H \subseteq \mathcal{B}$.

We now introduce the following two lemmas, which will be used in the proofs of Propositions 1 and 2, respectively.

Lemma 3. *Let P be a program, let $K \subset H \subseteq \mathcal{B}$. Then:*

$$LHM(P \cup H) - LHM(P \cup K) \supset H - K \qquad \iff \qquad H \cap (LHM(P \cup K) - K) = \emptyset.$$

Proof. (*if* part) Suppose that $LHM(P \cup H) - LHM(P \cup K) \not\supset H - K$, that is, suppose that there exists x such that: $x \in (H - K) \wedge x \notin (LHM(P \cup H) - LHM(P \cup K))$. Since $H \subseteq LHM(P \cup H)$ this means that: $x \in (H - K) \wedge x \in LHM(P \cup K)$. Hence we have that there exists x: $x \in (H \cap (LHM(P \cup K) - K))$. Contradiction.
(*only-if* part) Suppose that there exists x such that: $x \in H \wedge x \in LHM(P \cup K) \wedge x \notin K$. Then: $x \in (H - K) \wedge x \notin (LHM(P \cup H) - LHM(P \cup K))$. Contradiction.

Lemma 4. *Let P and Q be definite programs. Then:*

$$LHM(P \cup Q) \supseteq LHM(P) \cup LHM(Q).$$

Proof. This corollary descends immediately from Observation 1. For instance: $LHM(P \cup Q) = min\{I \mid I \in M(P \cup Q)\} = min\{I \mid I \in M(P) \wedge I \in M(Q)\} \supseteq min\{I \mid I \in M(P)\} = LHM(P)$.

Proposition 1
Let P be a program, let $I \subseteq \mathcal{B}$ and let $H \subseteq \mathcal{B}$.

$$I(H) \in AM(P) \implies \exists K, J : (K \subseteq H \wedge J(K) \in \mu AM(P) \wedge I = J \cup (H - K)).$$

Proof. (case a) If $I(H) \in \mu AM(P)$ then the assertion trivially holds (just take $K = H$).
(case b) If $I(H) \notin \mu AM(P)$ then, by definition of minimal admissible model:
$\exists K \subset H : LHM(P \cup H) - LHM(P \cup K) \not\supset H - K$.
By Lemma 3, this means that:
$\exists K \subset H : H \cap (LHM(P \cup K) - K) \neq \emptyset$
Let $x \in H \cap (LHM(P \cup K) - K)$. We now show that:
$LHM(P \cup H) = LHM(P \cup H - \{x\}) \cup \{x\}$.
Since $x \in LHM(P \cup K)$ then by Observation 1 we have that $x \in LHM(P \cup K \cup \Delta)$ for any $\Delta \subseteq \mathcal{B}$. Therefore we have that $x \in LHM(P \cup K \cup (H - \{x\}))$, that is, since $x \notin K$, $x \in LHM(P \cup (H - \{x\}))$. Now if $x \in LHM(P \cup (H - \{x\}))$ then $LHM(P \cup H) = LHM(P \cup H - \{x\})$ and hence $LHM(P \cup H) = LHM(P \cup H - \{x\}) \cup \{x\}$.

Let now $J = LHM(P \cup H - \{x\})$. If $J(H - \{x\}) \in \mu AM(P)$ then the statement is proved. Otherwise we repeatedly apply the reasoning of (case b) to obtain a decreasing chain of sets of hypotheses $K_0 \supset K_1 \supset K_2 \supset \ldots$ where $K_0 = H$ and where for each K_i:
- $J_i(K_i) \in AM(P)$, where $J_i = LHM(P \cup K_i)$, and
- $J_i = J_{i+1} \cup (K_i - K_{i+1})$.

Such a chain has a greatest lower bound $\overline{K} = \bigcap_i K_i$ such that $\overline{J}(\overline{K}) \in \mu AM(P)$, in the worst case $\overline{K}$ being the empty set. We therefore have that: $LHM(P \cup H) = LHM(P \cup \overline{K}) \cup (H - \overline{K})$.

Proposition 2
$\equiv_M \; = \equiv_{\mu AM}$.

Proof. ($\subseteq$) We first show that:
$P \equiv_M Q \Longrightarrow P \equiv_{\mu AM} Q$.
Suppose that:
$\exists P, Q : P \equiv_M Q \wedge P \not\equiv_{\mu AM} Q$,
namely
$\exists I, H : I(H) \in \mu AM(P) \wedge I(H) \notin \mu AM(Q)$.
(The other case is analogous.)

(i) Suppose that $H \subseteq AH(Q)$. Since $I(H) \in \mu AM(P)$ then:
$\forall K \subset H : \ LHM(P \cup H) - LHM(P \cup K) \supset H - K.$
Since $P \equiv_M Q$ then, by Lemma 2:
$\forall K \subset H : \ LHM(Q \cup H) - LHM(Q \cup K) \supset H - K.$
Therefore we have that $I(H) \in \mu AM(Q)$. Contradiction.

(ii) Suppose that $H \not\subseteq AH(Q)$.
Then $\exists n : \ n \in H \wedge n \notin AH(Q)$. Let $K = H - \{n\}$.

* If $n \in LHM(Q \cup K)$ then
$LHM(Q \cup K \cup \{n\}) = LHM(Q \cup K)$,
that is,
$LHM(Q \cup H) = LHM(Q \cup K)$.
Therefore, since $P \equiv_M Q$, by Lemma 2:
$\exists K \subset H : LHM(P \cup H) - LHM(P \cup K) = \emptyset$.
Therefore $I(H) \notin \mu AM(P)$. Contradiction.
* Suppose instead that $n \notin LHM(Q \cup K)$. Since $n \notin AH(Q)$:
$LHM(Q \cup H) = LHM(Q \cup K) \cup \{n\}$.
Then, since $n \notin LHM(Q \cup K)$:
$LHM(Q \cup H) - LHM(Q \cup K) = \{n\}$.
Therefore since $P \equiv_M Q$ and by Lemma 2:
$\exists K \subset H : LHM(P \cup H) - LHM(P \cup K) \not\supset H - K$.
Hence $I(H) \notin \mu AM(P)$. Contradiction.

($\supseteq$) We now show that:
$P \equiv_{\mu AM} Q \Longrightarrow P \equiv_M Q$.
Suppose that:
$\exists P, Q : \ P \equiv_{\mu AM} Q \ \wedge \ P \not\equiv_M Q$
that is:
$\exists I : \ I \in M(P) \ \wedge \ I \notin M(Q)$.
(The other case is analogous.)
Observe that
$LHM(P \cup I) = I$
since, by Observation 1, $LHM(P \cup I) = min\{J \mid J \in M(P) \wedge J \in M(I)\}$ and since $I = LHM(I)$ and $I \in M(P)$. Moreover:
$LHM(Q \cup I) \supset I$
since, by Observation 1, if $J \in M(Q \cup I)$ then $J \supseteq I$ and since $I \notin M(Q)$.
Let now $I_{AQ} = I \cap AH(Q)$ and let $I_{NQ} = I - I_{AQ}$. Then:
$LHM(Q \cup I) = LHM(Q \cup I_{AQ}) \cup I_{NQ}$.
Since, by Lemma 4:
$I = LHM(P \cup I) \supseteq LHM(P \cup I_{AQ}) \cup LHM(I_{NQ}) = LHM(P \cup I_{AQ}) \cup I_{NQ}$
we observe that:
$LHM(P \cup I_{AQ}) \subseteq I$ and $I_{NQ} \subseteq I$.
Therefore, since $LHM(Q \cup I) \supset I$ and $LHM(Q \cup I) = LHM(Q \cup I_{AQ}) \cup I_{NQ}$ and $I_{NQ} \subseteq I$, we observe that:
$LHM(Q \cup I_{AQ}) \not\subseteq I$.
Consider now $N = LHM(Q \cup I_{AQ})$. Since $N(I_{AQ}) \in AM(Q)$ then, by Proposition 1, $\exists K, J$:

$K \subseteq I_{AQ} \ \wedge \ J(K) \in \mu AM(Q) \ \wedge \ LHM(Q \cup I_{AQ}) = LHM(Q \cup K) \cup (I_{AQ} - K)$.
Since $LHM(Q \cup I_{AQ}) \not\subseteq I$ and since $I_{AQ} \subseteq I$, this implies that:
$LHM(Q \cup K) \not\subseteq I$.
Since $J(K) \in \mu AM(Q)$ and since $P \equiv_{\mu AM} Q$ then $J(K) \in \mu AM(P)$ and hence:
$LHM(P \cup K) \not\subseteq I$.
On the other hand, since $K \subseteq I_{AQ}$:
$I \supseteq LHM(P \cup I_{AQ}) \cup I_{NQ} \supseteq LHM(P \cup I_{AQ}) \supseteq LHM(P \cup K)$.
Contradiction.

Proposition 3
$\equiv_M \ = \equiv_{SI(\mathcal{B})}$.

Proof. ($\subseteq$) If $P \equiv_M Q$ then, by Lemma 2:
$\forall H \subseteq \mathcal{B} : LHM(P \cup H) = LHM(Q \cup H)$.
Hence:
$\{I(H) \mid H \subseteq \mathcal{B} \wedge I = LHM(P \cup H)\} \ = \ \{I(H) \mid H \subseteq \mathcal{B} \wedge I = LHM(Q \cup H)\}$
that is:
$P \equiv_{SI(\mathcal{B})} Q$.
($\supseteq$) By Lemma 1, if $I \in M(P)$ then $\exists H : I(H) \in SI(P)$. Since $SI(P) = SI(Q)$ by hypothesis, then $I(H) \in SI(Q)$ and hence $I \in M(Q)$ by Lemma 1.

Proposition 4
Let $\mathcal{H} \subseteq \mathcal{B}$, $\mathcal{K} \subseteq \mathcal{B}$ be two sets of assumable hypotheses. Then:

(1) $\mathcal{H} \subseteq \mathcal{K} \implies \equiv_{SI(\mathcal{K})} \subseteq \equiv_{SI(\mathcal{H})}$
(2) $\mathcal{H} \subset \mathcal{K} \implies \equiv_{SI(\mathcal{K})} \subset \equiv_{SI(\mathcal{H})}$

Proof. (1) We show that if $\mathcal{H} \subseteq \mathcal{K}$ then $(P \equiv_{SI(\mathcal{K})} Q \implies P \equiv_{SI(\mathcal{H})} Q)$. Indeed if $I(H)$ belongs to the set $SI(P, \mathcal{H})$ then it also belongs to $SI(P, \mathcal{K})$ since $\mathcal{H} \subseteq \mathcal{K}$. Since $P \equiv_{SI(\mathcal{K})} Q$ by hypothesis, $I(H)$ also belongs to $SI(Q, \mathcal{K})$, as well as to $SI(Q, \mathcal{H})$ since $\mathcal{H} \subseteq \mathcal{K}$.
(2) We now show that if $\mathcal{H} \subset \mathcal{K}$ then $\equiv_{SI(\mathcal{K})} \subset \equiv_{SI(\mathcal{H})}$. Since we know from (1) that $\equiv_{SI(\mathcal{K})} \subseteq \equiv_{SI(\mathcal{H})}$, we just have to show that there exist two programs P and Q such that $P \not\equiv_{SI(\mathcal{K})} Q$ and $P \equiv_{SI(\mathcal{H})} Q$.
Let $b \in \mathcal{K}$ and $b \notin \mathcal{H}$. Consider the program

P
$a \leftarrow b$

and let Q be the empty program. It is easy to see that $P \equiv_{SI(\mathcal{H})} Q$ since:
$SI(P, \mathcal{H}) = SI(Q, \mathcal{H}) = \{I(I) \mid I \subseteq \mathcal{H}\}$.
On the other hand $P \not\equiv_{SI(\mathcal{K})} Q$ since $\{a, b\}(\{b\}) \in SI(P, \mathcal{K})$
while $\{a, b\}(\{b\}) \notin SI(Q, \mathcal{K})$.

Proposition 5
$\equiv_{SI(\mathcal{B}^+)}$ *is compositional for* $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$.

Proof. Since $not\_\mathcal{B} \subset \mathcal{B}^+$ we have, by Proposition 4, that $\equiv_{SI(\mathcal{B}^+)} \subset \equiv_{SI(not\_\mathcal{B})}$, that is $\equiv_{SI(\mathcal{B}^+)}$ preserves $\equiv_{SI(not\_\mathcal{B})}$.

Moreover $\equiv_{SI(\mathcal{B}^+)}$ is a congruence for $\{\cup\}$ since $\equiv_M$ is a congruence for $\{\cup\}$ and since $\equiv_M = \equiv_{SI(\mathcal{B}^+)}$ by Proposition 3.

Proposition 6
$\equiv_{SI(\mathcal{B}^+)}$ *is fully abstract for* $(\equiv_{SI(not\_\mathcal{B})}, \{\cup\})$.

Proof. By counter-positive, we show that if $P \not\equiv_{SI(\mathcal{B}^+)} Q$ then there is a context in which the substitution of P with Q does modify the set of negatively supported interpretations. Let P and Q be two programs such that $P \not\equiv_{SI(\mathcal{B}^+)} Q$.

(1) We first show that there exists a finite program R such that:
$LHM(P \cup R) \neq LHM(Q \cup R)$.
Since we are considering positive versions of programs, P and Q are definite programs and hence, by Proposition 3:
$P \not\equiv_{SI(\mathcal{B}^+)} Q \Longrightarrow P \not\equiv_M Q$
This means that:
$\exists I \subseteq \mathcal{B}^+ :\ I \in M(P) \wedge I \notin M(Q)$.
(The other case is analogous.) That is, by definition of Herbrand model:
$\exists I \subseteq \mathcal{B}^+ :\ T(P)(I) \subseteq I \wedge T(Q)(I) \not\subseteq I$.
Therefore, by definition of $T(Q)$:
$\exists F, A :\ F \subseteq I \ \wedge\ F$ finite $\ \wedge A \in T(Q)(F) \ \wedge\ A \notin F$.
Consider now the program:
$R = \{B \leftarrow | \ B \in F\}$.

- We now show that $A \notin LHM(P \cup R)$, that is, $A \notin T^\omega(P \cup R)(\emptyset)$.
We first show, by induction on n, that:
$\forall n :\ T^n(P \cup R)(\emptyset) \subseteq I$.
Indeed $T^0(P \cup R)(\emptyset) = \emptyset$. Assume now that $T^n(P \cup R)(\emptyset) \subseteq I$. By definition of powers of T:
$T^{n+1}(P \cup R)(\emptyset) = T(P \cup R)(T^n(P \cup R)(\emptyset))$.
By inductive hypothesis and by the monotonicity of T:
$T^{n+1}(P \cup R)(\emptyset) \subseteq T(P \cup R)(I)$.
Then by definition of $\cup$:
$T^{n+1}(P \cup R)(\emptyset) \subseteq T(P)(I) \cup T(R)(I)$.
Since $T(P)(I) \subseteq I$ and since $F \subseteq I$, we then have that:
$T^{n+1}(P \cup R)(\emptyset) \subseteq I$.
Therefore, by the continuity of T:
$T^\omega(P \cup R)(\emptyset) \subseteq I$
and hence, since $A \notin I$, we conclude that:
$A \notin LHM(P \cup R)$.

- We now show that: $A \in LHM(Q \cup R)$, that is, $A \in T^{\omega}(Q \cup R)(\emptyset)$. Indeed:
  $A \in T(Q)(F) \subseteq T(Q)(T(R)(\emptyset))$
  By definition of $\cup$ and by monotonicity of T:
  $T(Q)(T(R)(\emptyset)) \subseteq T(Q)(T(Q \cup R)(\emptyset)) \subseteq T(Q \cup R)(T(Q \cup R)(\emptyset))$
  hence
  $A \in T^2(Q \cup R)(\emptyset) \subseteq T^{\omega}(Q \cup R)(\emptyset)$
  by continuity of T.

(2) We have therefore shown that if $P \not\equiv_{SI(\mathcal{B}^+)} Q$ then
$\exists R: \ LHM(P \cup R) \neq LHM(Q \cup R)$.
Notice that, since we are considering positive versions of programs, R can be a finite program of the form:
$R = \{B \leftarrow | \ B \in F\}$
where $F \subseteq (\mathcal{B} \cup not\_\mathcal{B})$. Namely R may be a finite program of the form:

R
$C_1 \leftarrow$
$\ldots$
$C_m \leftarrow$
$not\_D_1 \leftarrow$
$\ldots$
$not\_D_n \leftarrow$

Let now $R^+ = \{C \leftarrow | \ C \in F \cap \mathcal{B}\}$ and $R^- = \{not\_D \leftarrow | \ not\_D \in F \cap not\_\mathcal{B}\}$.

- We now observe that if $LHM(P \cup R^+) \neq LHM(Q \cup R^+)$ then:
  $\exists I, J: \ I(\emptyset) \in SI(P \cup R^+, not\_\mathcal{B}) \wedge J(\emptyset) \in SI(Q \cup R^+, not\_\mathcal{B}) \wedge I \neq J$.
  This means that there exists a normal program R^+ such that:
  $P \cup R^+ \not\equiv_{SI(not\_\mathcal{B})} Q \cup R^+$
  and hence concludes the proof.
- If instead $LHM(P \cup R^+) = LHM(Q \cup R^+)$ then, since $LHM(P \cup R) \neq LHM(Q \cup R)$, we observe that:
  $LHM(P \cup R^+ \cup R^-) \neq LHM(Q \cup R^+ \cup R^-)$.
  Let $F^- = \{not\_D \mid not\_D \leftarrow \in R^-\}$. Then:
  $\exists I, J: \ I(F^-) \in SI(P \cup R^+, not\_\mathcal{B}) \ \wedge \ J(F^-) \in SI(Q \cup R^+, not\_\mathcal{B})$
  $\wedge \ I \neq J$.
  That is, there exists a normal program R^+ such that:
  $P \cup R^+ \not\equiv_{SI(not\_\mathcal{B})} Q \cup R^+$.

Proposition 7
$\equiv_{SI(\mathcal{B})} \subset \equiv_{SI(\mathcal{B}_\pi)} \subset \equiv_{LHM}$.

Proof. By Proposition 4 we know that if $\mathcal{B}_\pi \subset \mathcal{B}$ then $\equiv_{SI(\mathcal{B})} \subset \equiv_{SI(\mathcal{B}_\pi)}$. Moreover if $P \equiv_{SI(\mathcal{B}_\pi)} Q$ then $LHM(P) = LHM(Q)$, that is, $\equiv_{SI(\mathcal{B}_\pi)} \subseteq \equiv_{LHM}$. To show that $\equiv_{SI(\mathcal{B}_\pi)} \neq \equiv_{LHM}$, consider the programs

P	Q
$a \leftarrow b$	$a \leftarrow c$

where $\{b, c\} \subseteq \mathcal{B}_\pi$. It is easy to observe that $P \equiv_{LHM} Q$ while $P \not\equiv_{SI(\mathcal{B}_\pi)} Q$ since for instance $\{a, b\}(\{b\}) \in SI(P, \mathcal{B}_\pi) - SI(Q, \mathcal{B}_\pi)$.

Proposition 8
$\equiv_{SI(\mathcal{B}_\pi)}$ *is compositional w.r.t* $(\mathcal{O}, \{\cup_\pi\})$.

Proof. Indeed $\equiv_{SI(\mathcal{B}_\pi)}$ preserves $\mathcal{O}$ by Prop. 7.

Moreover $\equiv_{SI(\mathcal{B}_\pi)}$ is a congruence for $\cup_\pi$, namely $\forall P, Q$:

$$P \equiv_{SI(\mathcal{B}_\pi)} Q \implies P \cup_\pi D \equiv_{SI(\mathcal{B}_\pi)} Q \cup_\pi D$$

for any extension D.

Let $H \subseteq \mathcal{B}_\pi$. We observe that, since both D and H are extensional programs:

$$LHM((P \cup_\pi D) \cup H) = LHM(P \cup_\pi LHM(D \cup H)).$$

Therefore, since $def s(D \cup H) \subseteq \pi$ and since $P \equiv_{SI(\mathcal{B}_\pi)} Q$:

$$LHM(P \cup_\pi LHM(D \cup H)) = LHM(Q \cup_\pi LHM(D \cup H))$$

and hence

$$LHM((P \cup_\pi D) \cup H) = LHM((Q \cup_\pi D) \cup H).$$

Proposition 9
$\equiv_{SI(\mathcal{B}^+)} \subset \equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})} \subset \equiv_{SI(not\_\mathcal{B})}$.

Proof. Immediate by Proposition 4, whenever $\pi \neq \emptyset$ (and $\mathcal{B}_\pi \subset \mathcal{B}$).

Proposition 10
$\equiv_{SI(\mathcal{B}_\pi \cup not\_\mathcal{B})}$ *is compositional for* $(\equiv_{SI(not\_\mathcal{B})}, \{\cup_\pi\})$.

Proof. The proof is analogous to the proof of Proposition 8.