ELSEVIER

Theoretical Computer Science 184 (1997) 1–59

Theoretical Computer Science

Fundamental Study

A unifying view for logic programming with non-monotonic reasoning

Antonio Brogi [a], Evelina Lamma [b,*], Paolo Mancarella [a], Paola Mello [c]

[a] *Dipartimento di Informatica, Università di Pisa, Corso Italia 40, Pisa, Italy*
[b] *DEIS, Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*
[c] *Dipartimento di Ingegneria, Università di Ferrara, Via Saragat, 41100 Ferrara, Italy*

Received February 1995; revised March 1996
Communicated by G. Levi

Abstract

We provide a simple formulation of a framework where some extensions of logic programming with non-monotonic reasoning are treated uniformly, namely, two kinds of negation and abduction. The resulting semantics is purely model-theoretic, and gives meaning to any non-contradictory abductive logic program. Moreover, it embeds and generalizes some existing semantics which deal with negation and abduction. The framework is equipped with a correct top-down proof procedure.

Contents

* Corresponding author. E-mail: elamma@deis.unibo.it

PII S0304-3975(96)00222-8

1. Introduction and motivations

Logic programming (LP) has been widely recognized as a powerful formalism for knowledge representation in various artificial intelligence (AI) domains. Both LP and AI have greatly benefited from such an interaction. In particular, using logic programming in typical AI domains has called for various extensions to the basic LP framework and for a better understanding of existing extensions, such as negation or abduction. Actually, the need of extending pure LP (Horn clauses) to deal with forms of non-monotonic reasoning was recognized since the early years of LP. Negation as failure was first introduced in [26] to express negative information, and it has been later recognized as a suitable form of non-monotonic reasoning. For instance, negation as failure provides a simple and elegant solution to the well-known *frame problem* in AI, as exemplified by the so-called "Yale Shooting Problem" [13, 31, 43]. More recently, other extensions have been proposed to further enrich the expressive power of LP in order to turn it into a general formalism for knowledge representation (for a survey, see [15]). Among others, we mention *abductive* LP [42, 35, 48] and LP with different forms of negation (e.g., [45, 56, 46, 52, 60, 10]). In particular, in [60] it is shown how logic programs with two kinds of negation (i.e., default and explicit negation) is applied to diverse domains of knowledge representation such as hierarchies and reasoning about actions.

The formalization of these extensions has called for new semantics capable of capturing their "intended" meaning. Even for the case of negation as failure, a number of different characterizations have been defined from different perspectives, most of which have been inspired from an interpretation of negation as failure as a more general notion of negation by default (e.g., [42]). A survey of the semantics of logic programs with negation (by default) is reported in [14]. The survey focuses on the proof-theoretic and model-theoretic issue, and on their relations.

Something similar is happening for other extensions such as abductive LP [47] and LP with a second form of negation in addition to negation by default (see, for instance, [45, 65, 56, 60]. 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. Another crucial issue is to understand the relations among different extensions. For instance, the rela-

tion between negation by default and abduction has been thoroughly studied starting from [42].

The ongoing proliferation of alternative proposals may ultimately create a sort of "labyrinth of LP and NMR" (similar to the labyrinth of LP and Concurrency) with the potential risk of getting too far away from LP. For this reason, trying to give a uniform view of extended LP which encompasses various extensions proposed so far is an laudable effort. Recently, some work has been done in this direction [6, 38, 16, 33, 65, 36]. For instance, Dung [36] studies the relation between stable and well-founded semantics for normal logic programs, and shows that stable semantics can be considered as a "two-valued well-founded semantics". Other authors [16, 47, 40] have pointed out that the framework of [35] is in fact an argumentation approach to logic programming and non-monotonic reasoning. In Dung's later work [38, 39], the intuitive notions of arguments and attack between arguments has been adopted both for reconstructing a number of NMR formalisms and for understanding different semantics of extended LP in a quite simple way. Inspired by the work of Dung, in [16] a general assumption-based framework for non monotonic reasoning has been developed, within which, again, many semantics for extended logic programming can be easily reconstructed. Still the choice of employing an argumentation-based framework, though simple and powerful, may be seen as a departure from the standard formalisms which are widely employed within the LP community. Moreover, at the time being, the absence of a model-theory for Dung's argumentation framework constitutes an obstacle to the study of formal properties which are typically based on model-theoretic grounds, such as compositionality.

A further valuable work, but not based on argumentation, is that by Alferes and Pereira [6], where the authors define a parameterizable schema to encompass and characterize a diversity of semantics involving two kinds of negation (namely default negation and classical, strong, weak, pseudo or explicit negation).

The goal of this paper is not to propose yet another semantics for LP and its extensions. Instead, our aim is to provide a simple formulation of a unified *LP-based* semantic framework within which three main extensions of LP for NMR are treated uniformly, namely two forms of negation and abduction. Our semantics is based on the well known notion of Herbrand models for definite Horn programs, since (extended) logic programs are simply viewed as definite (Horn) programs, through the notion of *positive version* of a program. Intuitively the idea is to restrict, in a step-wise fashion, the set of Herbrand models of the positive version of a program in order to identify the intended models of the original program.

This paper is an extended version of a previous paper [20], where we have applied these ideas to the case of normal logic programming, that is logic programming with negation by default. In [20], we provided an alternative characterization of the semantics of normal logic programs, based on purely model-theoretic arguments, which allowed us to reconstruct and compare existing, and seemingly different, proposals, such as Przymusinski's stationary semantics [65] and Dung's preferential semantics [35]. In this

paper, we first present the results of [20][1] in a slightly different way, partly inspired on the work of [38, 16]. This allows us next to naturally generalize the approach to other extensions of logic programming, namely those including a second form negation and abduction with integrity constraints. The final outcome is a semantics framework, based on model-theoretic grounds, within which these extensions can be accommodated in an easy and uniform way.

An interesting issue related to a semantic characterization is the study of the equivalence notion it induces with particular reference to compositionality and fully abstractness properties [54]. In the paper, we will discuss this topic, and, in particular, we will show how the compositionality property can be easily accommodated in our semantic framework, thanks to its model-theoretic grounds.

We also develop an abstract specification of a proof-procedure for the overall framework, which is a simple extension of the proof-procedure for abductive logic programming, first presented in [42] and further developed in [35, 48, 47]. The proof-procedure is shown to be sound and weakly complete with respect to the model-theory.

2. Preliminaries

Let us first set up some basic notations and terminology. We shall use the basic concepts of logic programming (e.g., as in [12]). Lowercase letters (possibly subscripted) denote *atoms* (e.g., $a, b, c, \ldots$) and *literals* (e.g., $l, l_0, \ldots$). The standard terminology must be extended to deal with the extensions of logic programming we are dealing with. In particular, we will need to distinguish between negation by default, denoted by *not*, and a second form of negation, denoted by $\neg$.

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

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

where $m, n \geqslant 0$ and each l_i $(i = 1, \ldots, m+n)$ is a literal of the form a or $\neg a$, where a is an atom.

In the following, since we will always refer to Herbrand interpretations and models, 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 rules obtained by instantiating the original rules in all possible ways over the Herbrand Universe of the program.

2.1. Positive version of a program

By the *positive version* of an extended program P we mean the definite program obtained by replacing negated literals of the form *not* a, $\neg a$ and *not* $\neg a$ by new positive atoms (following, e.g., [41, 65, 20]). Given an extended program P, let $\mathscr{L}_P^+$ denote the set of all ground atoms a such that either a, *not* a, $\neg a$, or *not* $\neg a$ occurs in P. For

[1] We also include proofs for the results simply stated in [20].

each $a \in \mathscr{L}_P^+$ three new propositional symbols are introduced to represent the default negation of a (not_a), the negation $\neg a$ of a ($a^\neg$), and the default negation of $\neg a$ ($not\_a^\neg$). We will use the following notations for the sets of symbols occurring in the positive version of a program:

$$\mathscr{L}_P^\neg = \{a^\neg \mid a \in \mathscr{L}_P^+\} \qquad \mathscr{L}_P^{\mathscr{C}} = \mathscr{L}_P^+ \cup \mathscr{L}_P^\neg \qquad \mathscr{L}_P^{\mathscr{D}} = \{not\_l \mid l \in \mathscr{L}_P^{\mathscr{C}}\}.$$

To point out their intuitive meaning, atoms in $\mathscr{L}_P^+$, $\mathscr{L}_P^\neg$, $\mathscr{L}_P^{\mathscr{C}}$ and $\mathscr{L}_P^{\mathscr{D}}$ will be referred to as *positive*, *negated*, *standard*, and *default* atoms, respectively.

The *positive version* of an extended program P is then obtained by replacing in P literals of the form $\neg a$ and $not\ l$ with the corresponding standard and default atoms in $\mathscr{L}_P^\neg$ and $\mathscr{L}_P^{\mathscr{D}}$, respectively.

Example 2.1. The positive version of the extended program

$a \leftarrow b, not\ \neg c$
$a \leftarrow not\ d$
$\neg c \leftarrow$

is the definite program

$a \leftarrow b, not\_c^\neg$
$a \leftarrow not\_d$
$c^\neg \leftarrow$

From now onwards, we will not distinguish any further between an extended program P and its positive version. Thus, we will denote directly by P the positive version of an extended program. The Herbrand Base of an extended program P is then

$$\mathscr{B}_P = \mathscr{L}_P^{\mathscr{C}} \cup \mathscr{L}_P^{\mathscr{D}}.$$

Since an extended program is always viewed as a definite program (viz., its positive version), we can talk about a Herbrand interpretation I of P, as usual, as a subset of $\mathscr{B}_P$. Given an interpretation I, $I^{\mathscr{D}}$ (resp., $I^{\mathscr{C}}$) will stand for the set of default (resp., standard) atoms in I, that is $I^{\mathscr{D}} = I \cap \mathscr{L}_P^{\mathscr{D}}$ (resp., $I^{\mathscr{C}} = I \cap \mathscr{L}_P^{\mathscr{C}}$).

The natural syntactic correspondence between a standard atom and its negation by default is given by the following notion of complement with respect to negation by default:

$$\overline{l} = \begin{cases} \alpha & \text{if } l = not\_\alpha \\ not\_\alpha & \text{otherwise} \end{cases}$$

where $\alpha \in \mathscr{L}_P^{\mathscr{C}}$. The above notion can be lifted to sets of atoms (interpretations) in the obvious way. Given an interpretation $X \subseteq \mathscr{B}_P$, its complement $\overline{X}$ with respect to negation by default is

$$\overline{X} = \{\overline{l} \mid l \in X\}.$$

An interpretation I is *total* if and only if $\forall L \in (\mathscr{L}^+ \cup \mathscr{L}^\neg)$ either $L \in I$ or $\overline{L} \in I$.

Semantically, the role of the two forms of negation is characterized by two basic properties of Herbrand interpretations, that we call *consistency* and *non-contradictoriness*. Both these properties can be defined via a notion of satisfaction of integrity constraints. Given a program P, by an *integrity constraint* we mean a denial of the form $\leftarrow l_1,\ldots,l_n$, with $\{l_1,\ldots,l_n\} \subseteq \mathscr{B}_P$.

Definition 2.1. Let P be an extended logic program and let Φ be a set of denials of the form $\leftarrow l_1,\ldots,l_n$, with $\{l_1,\ldots,l_n\} \subseteq \mathscr{B}_P$. Let also $I \subseteq \mathscr{B}_P$ be an interpretation. Given $\phi \in \Phi$, we say that I *satisfies* ϕ (denoted by $I \models \phi$) if and only if $\{l_1,\ldots,l_n\} \not\subseteq I$. Moreover, we say that I *satisfies* Φ (denoted by $I \models \Phi$) if and only if $I \models \phi$, for each $\phi \in \Phi$.

Two sets of integrity constraints, $IC^{\neg}$ and IC^{not}, are associated with (the positive version of) an extended program P. (In the sequel, we write simply $\mathscr{B}$, $\mathscr{L}^+$,... instead of $\mathscr{B}_P$, $\mathscr{L}_P^+$,...when this does not lead to ambiguity.)

Definition 2.2. Let P be an extended program. The integrity constraints associated with P are the the following two (possibly infinite) sets of denials:

$$IC^{\neg} = \{\leftarrow a, a^{\neg} \mid a \in \mathscr{L}^+\}, \qquad IC^{not} = \{\leftarrow l, not\_l \mid l \in \mathscr{L}^{\mathscr{C}}\}$$

The notions of consistency and non-contradictoriness of an interpretation can now be given as follows.

Definition 2.3. Let P be an extended program and $I \subseteq \mathscr{B}$ be a Herbrand interpretation. Then

(i) I is *consistent* if and only if $I \models IC^{not}$. I is *inconsistent* otherwise.
(ii) I is *non-contradictory* if and only if $I \models IC^{\neg}$. I is *contradictory* otherwise.

Namely, an interpretation I is consistent if it does not make both l and not_l true, for any standard atom l, that is, if and only if $I \cap \bar{I} = \emptyset$. On the other hand, I is non-contradictory if it does not make both a and $a^{\neg}$ true, for any standard atom a.

2.2. Supported interpretations for definite programs

Our semantics for extended logic programs, considering both negation and abduction with integrity constraints, is built upon the semantics characterization of *normal* logic programs, that is, logic programs with negation by default presented in [20].

The starting point is the characterization of the *supported interpretations* for definite, ground programs. The concept of *supported interpretations* adopted here is inherited from a similar concept first introduced in [22] and then exploited in [20]. In [22] supported interpretations are used to provide an alternative characterization of the standard Herbrand interpretations of a definite logic program, which provides a compositional model-theoretic semantics of logic programs (see also Section 6).

The notion of supported interpretation that we use (Definition 2.4 below) is a generalization of the one presented in [22]. In this section, a logic program P is a propositional definite program and $\mathscr{B}$ denotes the underlying Herbrand Base.

Definition 2.4. Let P be a logic program and let $\mathscr{H}$ be a predefined subset of $\mathscr{B}$, called the set of *assumables*. Given a Herbrand interpretation $I \subseteq \mathscr{B}$ and a set of assumables $H \subseteq \mathscr{H}$, we say that I is supported by H (denoted by $I(H)$) if I is the least Herbrand model of the program $P \cup H$.

It is worth noting that the notion of supported interpretation of [22] coincides with the one given in the previous definition when the set of assumables is the whole Herbrand Base $\mathscr{B}$. Indeed, the Herbrand models of a definite logic program P are just supported interpretations with respect to the set of assumables $\mathscr{H} = \mathscr{B}$, as shown in [22]. Moreover, for any predefined set of assumables $\mathscr{H}$, the standard least Herbrand model of a program P is obviously the interpretation supported by the empty set of assumables. It is also worth noting that the same interpretation I can be supported by different sets of assumables, as the following example shows.

Example 2.2. Let P be the program

P:

$p \leftarrow q$

$q \leftarrow$

and let the set of assumables be $\mathscr{H} = \{p\}$. Clearly the least Herbrand model of P, namely $\{p,q\}$ is supported both by $\emptyset$ and by $\{p\}$.

The semantics of the various extensions of logic programming we are going to deal with will be given by defining in each case which subset of $\mathscr{B}$ is taken as the predefined set of assumables, and by identifying a subset of the supported interpretations as the intended models of the extended program.

As a first example, if the initial program P is a definite logic program, its standard meaning (i.e. its least Herbrand model) can be simply reconstructed by taking $\mathscr{H} = \emptyset$ and then by considering the unique supported interpretation $M(\emptyset)$.

3. Negation by default

3.1. Model theory

Let us first consider the case of logic programs with negation by default, usually referred to as *normal logic programs*. (Notice that the definitions and results stated in this section have been already presented in [20], where no proof was included.)

(The positive version of) A normal logic program is a set of rules of the form

$$a \leftarrow l_1, \ldots, l_n$$

where a is a positive atom and each l_i is either a positive atom or a negation by default atom not_b, b being a positive atom. In other words, normal logic programs are such that $\mathscr{L}^{\neg} = \emptyset$. Hence, $\mathscr{L}^{\mathscr{C}} = \mathscr{L}^{+}$ and $\mathscr{L}^{\mathscr{D}}$ reduces to the set $\{not\_a \mid a \in \mathscr{L}^{+}\}$.

The set of assumables in this case is the set of default atoms in the Herbrand Base, that is

$$\mathscr{H} = \mathscr{L}^{\mathscr{D}}.$$

The definition of supported interpretation (Definition 2.4) can be suitably instantiated according to the above definition of the set of assumables $\mathscr{H}$ for normal programs.

Definition 3.1. Let P be a normal logic program, and let $H \subseteq \mathscr{L}^{\mathscr{D}}$ be a set of assumables. A Herbrand interpretation $I \subseteq \mathscr{B}$ is *supported* by H if I is the least Herbrand model of $P \cup H$.

Intuitively speaking, a supported interpretation $I(H)$ corresponds to assuming the truth of a set of default hypotheses, that is, a set of default atoms of the form not_a. This view basically corresponds to the *abductive* interpretation of negation by default, first introduced in [42] and further refined in [35].

Example 3.1. Consider the program P:

$$a \leftarrow not\_b$$

The Herbrand Base associated with P is $\mathscr{B} = \{a, b, not\_a, not\_b\}$ and the set of assumables is $\mathscr{L}^{\mathscr{D}} = \{not\_a, not\_b\}$. The supported interpretations for P are then

$\{\}$	supported by	$\{\}$
$\{not\_a\}$	supported by	$\{not\_a\}$
$\{a, not\_b\}$	supported by	$\{not\_b\}$
$\{a, not\_a, not\_b\}$	supported by	$\{not\_a, not\_b\}$

Intuitively speaking, a supported interpretation represents a possible world where some default atoms are assumed to hold. Whether or not a supported interpretation can be taken as an intended model of the original normal program depends on various issues. First of all, a supported interpretation has to be *consistent*, in the sense of Definition 2.3 (that is, it should not make both a and not_a true for any given atom a).

Definition 3.2. Let P be a normal program. A supported interpretation $I(H)$ of P is a supported model of P if it is consistent.

Secondly, any supported model must satisfy an *admissibility* condition which we are going to discuss next and which is the basis of the proposed semantics for negation by default. It is worth noting that such a condition is a reconstruction, in the framework of supported interpretations, of the admissibility condition introduced by Dung [35] in his definition of the *preferential semantics* for normal logic programs. As shown in

[20], our formulation of admissibility allows us to easily reconstruct also the *stationary semantics* for normal logic programming introduced by [65]. This allows us to make a formal comparison of the two approaches within the unifying framework of supported interpretations.

The notion of admissibility relies on a notion of *conservative extension* of a supported model. A supported model may be extended by enlarging the set of assumables (i.e. default atoms). Namely, the extensions of a supported model $I(H)$ are all the supported interpretations which assume more default hypotheses than H. If we look at supported models as representing consistent possible worlds, the extension of a model represents a richer world where more defaults are assumed to hold. Clearly, when enriching the set of defaults of a possible world, we are interested only in assuming new defaults not_a such that a is not true in the world we start with, so as not to move into an inconsistent world. The extensions of a supported model I which satisfy this property are the *conservative extensions* of I. An interpretation I is then admissible if all its conservative extensions do not contradict the default hypotheses of I. In other words, a supported model is admissible if there is no way to conservatively extend it and to defeat some of its default assumptions. These notions are formalized in the next two definitions. In Section 3.2.1, we will also present an argumentation based interpretation of these notions, along the lines of the work of [38].

Definition 3.3. Let P be a normal program and let $I(H)$ and $J(K)$ be two supported interpretations. Let also $I(H)$ be consistent. Then, $J(K)$ is a *conservative extension* of $I(H)$ if and only if $K \supseteq H$ and $\overline{K} \cap I = \emptyset$. Given a supported interpretation I, $CE(I)$ denotes the set of the conservative extensions of I.[2]

Notice that J can extend I by enlarging the set of default atoms (assumables) which are not inconsistent with I. In fact, the condition $(\overline{K} \cap I) = \emptyset$ in the previous definition states that there is no atom a in I such that not_a is assumed in J. Furthermore, J may be possibly inconsistent even if I is consistent.

Example 3.2. Consider again program P

$$a \leftarrow not\_b$$

It is easy to see that each supported interpretation of P conservatively extends the model supported by the empty set. On the other hand, there is no conservative extension of the supported model $\{a, not\_b\}$ since its only extension $\{a, not\_a, not\_b\}$ is not consistent with it (viz. $\overline{\{not\_a, not\_b\}} \cap \{a, not\_b\} = \{a\} (\neq \emptyset)$).

The admissibility of a supported model depends on its conservative extensions. A supported model I is admissible if none of its conservative extensions defeats the default assumptions in I.

[2] With abuse of notation, we write $CE(I)$ instead of $CE(I(H))$.

Definition 3.4. Let P be a normal program. A supported model $I(H)$ is an *admissible model* if and only if for any conservative extension $J(K)$ of I, $\overline{H} \cap J = \emptyset$.

Remark 3.1. Notice that, for normal programs, a supported model $I(H)$ is admissible if and only if the following property holds:

$$(\forall a \in \mathscr{L}^{\mathscr{C}}.\ not\_a \in H \Rightarrow (\forall J \in CE(I).\ a \notin J)).$$

As mentioned before, a supported interpretation must be both consistent and admissible in order to be taken as a candidate intended model of a program.

Once we have determined the admissible models of a normal program, we wish to identify those which, intuitively, assume as many default hypotheses as possible. A *complete* model is then a model which assumes all (and only) the defaults *not_a* such that a is not true in all its conservative extensions.

Definition 3.5. Let P be a normal program. An admissible model $I(H) \subseteq \mathscr{B}$ is a *complete model* of P if and only if the following property holds:

$$(\forall a \in \mathscr{L}^{\mathscr{C}}.(not\_a \in H \ \Leftrightarrow\ (\forall J \in CE(I) : a \notin J))).$$

In the next section we will show that complete models coincide with Dung's complete scenarios [35, 40]. The following proposition is then a direct consequence of the results of [35, 40].

Proposition 3.1. *Let P be a normal program. Then:*

(i) *P has at least one complete model;*

(ii) *Any total supported model of P is a complete model of P.*

Moreover, we will show that our framework allows one to reconstruct Przymusinski's stationary semantics [65]. As a byproduct, we obtain a formal result which states that Dung's and Przymusinski's semantics coincide.

3.2. Related work

In this section, we discuss the relations between the semantics for negation by default presented in the previous section and other approaches to the semantics of normal logic programming. We first present the relations with Dung's preferential semantics and Przymusinski's stationary semantics. (These results were first stated in [20].) As already mentioned, a unified view of these semantics provides a formal result of equivalence between them. This equivalence can be further extended to Saccà and Zaniolo's partial stable models semantics [69] by exploiting the results of [50, 73], thus establishing the equivalence of the three main approaches to the semantics of normal logic programming presented so far. Finally, we mention the relations with other semantics of normal logic programming, such as the stable model semantics of [44] and the well-founded semantics of [72].

3.2.1. Preferential semantics

In [35], Dung exploited the treatment of negation as hypothesis in logic programming first introduced in [42], in order to set up a simple semantics for logic programs with negation by default. The main idea is to look at negative literals as possible abductive hypotheses, and this has highly inspired subsequent proposals, including ours. In [35] and [40] the semantics of a program P is given through the notion of *admissible* extensions of the theory represented by P. An extension (called *scenario*) is obtained by augmenting P with a set of abductive hypotheses H. Stated otherwise a scenario is a theory $P \cup H$, with $H \subseteq \mathscr{L}^{\mathscr{Q}}$.

Let us recall the basic notions and results of [35]. To do this, we do not follow the original presentation of [35] but rather a different, though equivalent, argumentation-theoretic formulation first presented in [47] and further refined in [16, 38, 40]. The reason is that the argumentation-theoretic view of these concepts is quite simple and intuitive. Moreover, to better understand the relations with our framework, we restate these notions by adopting as much as possible the terminology and notations set up in Section 2.

As mentioned above, the semantics of a normal program P is given in terms of *admissible* scenaria of the form $P \cup H$, where $H \subseteq \mathscr{L}^{\mathscr{Q}}$ is a set of default assumptions. The admissibility condition of a scenario is based on an argumentation-theoretic criterion [38]. Roughly speaking, an admissible set of assumptions must be able to successfully "counter-attack" any other set of assumptions which "attacks" it. Intuitively speaking, a set of assumptions corresponds to a set of arguments which are able to defeat any counter-argument against them. The key notion is then the notion of attack between sets of arguments.

Definition 3.6. Let P be a normal program and let $H, K \subseteq \mathscr{L}^{\mathscr{Q}}$ be two sets of assumables. Then H *attacks* K if and only if

$$\exists not\_a \in K.\ P \cup H \vdash a$$

Remark 3.2. In the above definition, since P is viewed as a positive program, $\vdash$ stands for the standard provability relation in Horn clause logic. Hence,

$$\forall a \in \mathscr{L}^{+}.\ P \cup H \vdash a \Leftrightarrow a \in T_{P \cup H} \uparrow \omega$$

A scenario is a theory $P \cup H$ such that H does not attack itself.

Definition 3.7. Let P be a normal program. A *scenario* is a theory $P \cup H$, where $H \subseteq \mathscr{L}^{\mathscr{Q}}$ is such that H does not attack H.

Among scenaria, Dung identifies as admissible those which are able to (counter-) attack any attack against them. We call them *D-admissible* scenaria in order to avoid confusion with our notion of admissibility.

Definition 3.8. Let P be a normal program. A scenario $P \cup H$ is *D-admissible* if and only if

$$\forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } H \Rightarrow H \text{ attacks } E$$

Notice that "E attacks H" means that $P \cup E \vdash a$ for some $not\_a \in H$, that is E is "evidence" for the contrary a of some hypothesis $not\_a \in H$. The D-admissibility of a scenario $P \cup H$ can then be informally explained as the ability of H to defeat any evidence for the contrary of any hypothesis in H.

The *preferential semantics* of a program P is given by the set of all its maximal D-admissible scenaria, called *preferred extensions* in [35].

Definition 3.9. Let P be a normal program. A *preferred extension* of P is a maximal (with respect to set inclusion) D-admissible scenario.

As shown in [35], preferred extensions can also be viewed as the maximal elements in the space of *complete scenaria*. Again, we refer to them as *D-complete* scenaria in order to avoid confusion with our notion of completeness. A scenario $P \cup H$ is D-complete if it contains all the hypotheses *acceptable* with respect to it, that is it contains any hypothesis not_a such that H (counter-)attacks any attack to $\{not\_a\}$.

Definition 3.10. Let P be a normal program. A D-admissible scenario $P \cup H$ is *D-complete* if and only if

$$H = \{not\_a \in \mathscr{L}^{\mathscr{Q}} \mid \forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } \{not\_a\} \Rightarrow H \text{ attacks } E\}$$

The following proposition summarizes the main results of [35].

Proposition 3.2. *Let P be a normal program. Then:*

(i) *P admits at least one preferred extension;*
(ii) *P admits at least one D-complete scenario;*
(iii) *The well-founded model of P corresponds to the* least (*w.r.t set inclusion*) *D-complete scenario.*

We are now in the position of stating the direct correspondence between D-complete/D-admissible scenaria and complete/admissible models. First of all, notice that there is an obvious direct correspondence between supported models and scenaria.

Proposition 3.3. *Let P be a normal program, $H \subseteq \mathscr{L}^{\mathscr{Q}}$ and $M(H)$ be the interpretation supported by H. Then, $P \cup H$ is a scenario $\Leftrightarrow$ $M(H)$ is consistent. Moreover, for each $a \in \mathscr{L}^{+}$, $a \in M^{\mathscr{C}} \Leftrightarrow P \cup H \vdash a$.*

Proof. Trivial. □

Moreover we can easily relate the definitions of attack, and then D-admissibility, in terms of properties of supported interpretations as follows.

Proposition 3.4. *Let P be a normal program, $H,K \subseteq \mathscr{L}^{\mathscr{Q}}$ and $I(H),J(K)$ the interpretations supported by H,K respectively. Then H attacks $K \Leftrightarrow \overline{K} \cap I \neq \emptyset$.*

Proof.

H attacks K

$\Leftrightarrow$ {Definition 3.6}

$\exists not\_a \in K.\ P \cup H \vdash a$

$\Leftrightarrow$ {remark 3.2}

$\exists not\_a \in K.\ a \in T_{P \cup H} \uparrow \omega$

$\Leftrightarrow$ {definition 3.1}

$\exists not\_a \in K.\ a \in I$

$\Leftrightarrow$ $\{\overline{not\_a} = a\}$

$\overline{K} \cap I \neq \emptyset$ □

The next theorem relates Dung's admissible scenaria and admissible models, by showing that they actually coincide for each normal program. Notice that, in Dung's definition (Definition 3.8) the admissibility of a scenario $P \cup H$ requires that H is able to counter-attack any attack E against it. In our view (Definition 3.4) the admissibility of a supported model $I(H)$ requires instead that for any K such that $J(K)$ is a conservative extension of $I(H)$, K does not attack H.[3] These apparently different notions are reconciled by the next theorem.

Theorem 3.5. *Let P be a normal program and $H \subseteq \mathscr{L}^{\mathscr{Q}}$. Then, $M(H)$ is an admissible model of $P \Leftrightarrow P \cup H$ is an admissible scenario of P.*

Proof. $\Rightarrow$: Let $M(H)$ be an admissible model of P. We need to show that $P \cup H$ is an admissible scenario. First of all, $P \cup H$ is a scenario, since $M(H)$ is consistent (proposition 3.3). We need to show that $P \cup H$ is admissible, i.e.

$$\forall E \subseteq \mathscr{L}^{\mathscr{Q}}.E \quad \text{attacks} \quad H \Rightarrow H \quad \text{attacks} \quad E.$$

Assume $P \cup H$ is not admissible, that is, by Definition 3.8,

$$\exists E \subseteq \mathscr{L}^{\mathscr{Q}}.E \quad \text{attacks} \quad H \wedge H \text{ does not attack } E.$$

Then, let $I(E)$ be the interpretation supported by E. We have

$$\exists E \subseteq \mathscr{L}^{\mathscr{Q}}.E \quad \text{attacks} \quad H \wedge H \quad \text{does not attack} \quad E$$

$\Leftrightarrow$ {Proposition 3.4}

$$\exists E \subseteq \mathscr{L}^{\mathscr{Q}}.\overline{H} \cap I \neq \emptyset \ \wedge\ \overline{E} \cap M = \emptyset \tag{1}$$

[3] Notice that in Dung's semantics evidence to the contrary may be inconsistent. As well, in our approach conservative extensions may be possibly inconsistent.

Let now $K = H \cup E$ and consider the interpretation $J(K)$. Then $J \in CE(M)$, since

(i) $K \supseteq H$ by construction;

(ii) $\overline{K} \cap M = \emptyset$, since $M \cap \overline{H} = \emptyset$ (by the consistency of M) and $M \cap \overline{E} = \emptyset$ (by (1) above).

Since $I \subseteq J$, by (1) above we have $\overline{H} \cap J \neq \emptyset$, thus contradicting the fact that $M(H)$ is an admissible model.

$\Leftarrow$: Assume $P \cup H$ is a D-admissible scenario. Then $M(H)$ is consistent by Proposition 3.3. Assume $M(H)$ is not an admissible interpretation, that is

$$\exists J(K) \in CE(M).\ J \cap \overline{H} \neq \emptyset.$$

We have

$\exists J(K) \in CE(M).\ J(K) \cap \overline{H} \neq \emptyset$

$\Leftrightarrow$ {Proposition 3.4}

$\exists J(K) \in CE(M).\ K$ attacks H

$\Rightarrow$ {Definitions 3.4 and 3.3}

$\exists K \subseteq \mathscr{L}^{\mathscr{Q}}.\overline{K} \cap M = \emptyset \ \wedge\ K$ attacks H

$\Leftrightarrow$ {Proposition 3.4}

$\exists K \subseteq \mathscr{L}^{\mathscr{Q}}.\ H$ does not attack $K \ \wedge\ K$ attacks H.

Thus we have a contradiction with the D-admissibility of $P \cup H$. □

Finally, we establish a one-to-one correspondence between complete models and D-complete scenarios.

Theorem 3.6. *Let P be a normal program, $H \subseteq \mathscr{L}^{\mathscr{Q}}$ and $M(H)$ the interpretation supported by H. Then, $M(H)$ is a complete model of $P \Leftrightarrow P \cup H$ is a D-complete scenario.*

Proof. $\Rightarrow$: Assume $M(H)$ is a complete model. Then, by Theorem 3.5, $P \cup H$ is D-admissible. Assume it is not D-complete. Then, by Definition 3.10,

$$\exists not\_a \in \mathscr{L}^{\mathscr{Q}}.\ not\_a \notin H \ \wedge\ (\forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } \{not\_a\} \Rightarrow H \text{ attacks } E),$$

or equivalently

$$\exists not\_a \notin H.\ (\forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ H \text{ does not attack } E \Rightarrow E \text{ does not attack } \{not\_a\}). \quad (1)$$

Let $J(K)$ be any conservative extension of $M(H)$. Then, by Definition 3.3 and by Proposition 3.4, we have that H does not attack K. Then, by (1) above, we have

$$\exists not\_a \notin H.\ (\forall J \in CE(M).\ a \notin J),$$

thus contradicting the fact that $M(H)$ is complete.

$\Leftarrow$: Assume that $P \cup H$ is a D-complete scenario. Then $M(H)$ is an admissible model by theorem 3.5. Assume that $M(H)$ is not complete, that is, by Definition 3.5,

$$\exists not\_a \notin H.\ (\forall J(K) \in CE(M).\ a \notin J). \quad (2)$$

By (2) and the D-completeness of $P \cup H$, we have

$$\exists E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } \{not\_a\} \ \wedge\ H \text{ does not attack } E. \quad (3)$$

Let now $K = H \cup E$, and $J(K)$ be the interpretation supported by K. Then, $J \in CE(M)$, since

(i) $\overline{H} \cap M = \emptyset$, by the consistency of M, and

(ii) $\overline{E} \cap M = \emptyset$, by (3) above, since H does not attack E.

Moreover, since by (3) E attacks $\{not\_a\}$ and by construction $E \subseteq K$, we also have that K attacks $\{not\_a\}$, or equivalently $a \in J(K)$. Thus we have that $J(K) \in CE(M)$ and $a \in J(K)$, thus contradicting (2). □

3.2.2. Stationary semantics

In [65], Przymusinski interprets negation in (disjunctive) normal programs as negation by default. The negation by default *not* ϕ of a propositional formula ϕ is treated as a new propositional symbol. The introduction of default propositions not_ϕ allows one to deal with both classical negation $\neg\phi$ and negation by default *not* ϕ. In Przymusinski's framework, a formula ϕ of the form a or $\neg a$ is called an *objective proposition*, while not_ϕ is called a *default proposition*. Since not_a is considered as a new predicate symbol, and classical negation is also taken into account, we talk about consistency and inconsistency in the classical sense, as opposed to our notions of consistency and inconsistency (see Definition 2.3).

In the sequel we recall the basic definitions and results of [65]. The basic idea is to define the semantics of a program P in terms of its *stationary* expansions, namely theories obtained by augmenting P with some default propositions of the form not_a or $\neg not\_a$, and by imposing a stationarity condition on them. Let us go into the details of Przymusinski's definitions. Let then P be a normal program and T an expansion of P, that is a *theory* obtained by augmenting P with some default propositions of the form not_a or $\neg not\_a$. In the sequel, we will use the classical notions of Herbrand interpretations and models of a theory T. Recall that, given a theory T, a Herbrand interpretation I of T can be identified with a subset of the Herbrand base $\mathscr{L}^+ \cup \mathscr{L}^{\mathscr{D}}$. Given an interpretation I, a default or objective proposition ϕ of the kind a or not_a is true in I if $\phi \in I$, and it is false in I otherwise. Moreover, I is a model of T if it satisfies all the statements in T.

Definition 3.11. Let P be a normal program and T an expansion of P. A model M of T is *minimal* if and only if there is no model N of T such that $N \subset M$ and N coincides with M on default propositions. We denote by $MIN(T)$ the class of all minimal models of a theory T.

According to [65], if a formula ϕ is true in all minimal models of T, we write $T \models_{min} \phi$.[4] Among the expansions of a program P, Przymusinski identifies as *stationary* those expansions T which are obtained by adding to the original program P some default

[4] As pointed out in [65], this notion of minimal model corresponds to considering predicate circumscription $CIRC(T; O; D)$ of the theory T, in which objective propositions O are minimized and default propositions D are fixed.

propositions of the form not_a (resp. $\neg not\_a$), in such a way that the corresponding objective propositions a are false (resp. true) in all minimal models of T.

Definition 3.12. Let P be a normal program, $H \subseteq \mathscr{L}^{\mathscr{D}}$ and $K \subseteq \{\neg not\_a \mid not\_a \in \mathscr{L}^{\mathscr{D}}\}$. The theory $T = P \cup H \cup K$ is a *stationary expansion* of P if and only if it is non-contradictory and satisfies the conditions:

(S1) $T \models not\_a \Leftrightarrow T \models_{min} \neg a$

(S2) $T \models \neg not\_a \Leftrightarrow T \models a$

for any objective proposition a (i.e. $a \in \mathscr{L}^{+}$).

Let us illustrate the above definition through a simple example.

Example 3.3. Consider the following program P, taken from [65]:

$a \leftarrow not\_b$

$b \leftarrow not\_a$

$c \leftarrow a, b$

This program has three stationary expansions, namely:

$$T_0 = P$$

$$T_1 = P \cup \{not\_a, \neg not\_b,\ not\_c\}$$

$$T_2 = P \cup \{not\_b,\ \neg not\_a, not\_c\}$$

Notice that T_1 and T_2 have only one minimal model, namely:

$$M_1 = \{b,\ not\_a,\ \neg not\_b,\ not\_c\}$$

$$M_2 = \{a,\ not\_b,\ \neg not\_a,\ not\_c\}$$

respectively.

In order to state the relation between stationary expansions and complete models of a program P, we first need to relate the framework of supported interpretations to the one of theories and expansions of theories. This is achieved by the following definition.

Definition 3.13. Let P be a normal program, $I(H)$ be a supported interpretation of P and T be an expansion of P. Then

(i) $\mathscr{T}(I)$, the *theory* associated with I, is defined as follows:

$$\mathscr{T}(I) = P \cup H \cup \{\neg not\_a \mid a \in I^{\mathscr{C}}\}$$

(ii) $\mathscr{S}(T)$, the *supported interpretation* associated with T, is the supported interpretation $J(K)$ such that $K = T \cap \mathscr{L}^{\mathscr{D}}$.

The next two propositions clarify the meaning of the above mapping, by showing the correspondence between minimal models of theories and conservative extensions of supported interpretations. More precisely, the first proposition states that for any supported model $I(H)$, the set of conservative extensions of I coincides with the set of minimal models of $\mathscr{T}(I)$. On the other hand, the second proposition states that, for any theory T satisfying condition $(S2)$ of definition 3.12, the set of minimal models of T coincides with the set of conservative extensions of $\mathscr{S}(T)$.

Proposition 3.7. *Let P be a normal program and $I(H)$ be a supported model. Then $J(K)$ is a conservative extension of $I \Leftrightarrow J$ is a minimal model of $\mathscr{T}(I)$.*

Proof. $\Rightarrow$: $I(H)$ is consistent. Let $J(K)$ be a conservative extension of I. We first show that J is a model of $\mathscr{T}(I)$, that is:

(1) $not\_a \in \mathscr{T}(I) \Rightarrow not\_a \in J$

(2) $\neg not\_a \in \mathscr{T}(I) \Rightarrow not\_a \notin J$

(3) for each rule $a \leftarrow b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m$ in P we have:

$$\{b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m\} \subseteq J \;\Rightarrow\; a \in J$$

Proof of (1):

$not\_a \in \mathscr{T}(I)$

$\Rightarrow$ {by construction of $\mathscr{T}(I)$}

$not\_a \in H$

$\Rightarrow$ {$H \subseteq K$, since $J(K)$ is a conservative extension of $I(H)$}

$not\_a \in K$

$\Leftrightarrow$ {$K = J^{\mathscr{D}}$}

$not\_a \in J$

Proof of (2):

$\neg not\_a \in \mathscr{T}(I)$

$\Rightarrow$ {by construction of $\mathscr{T}(I)$}

$a \in I$

$\Rightarrow$ {$\overline{K} \cap I = \emptyset$, since $J(K)$ is a conservative extension of $I(H)$}

$not\_a \notin K$

$\Leftrightarrow$ {$K = J^{\mathscr{D}}$}

$not\_a \notin J$

Proof of (3): Let $a \leftarrow b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m$ be a rule in P such that:

$$\{b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m\} \subseteq J.$$

Then $a \in J$ since J is a supported interpretation.

Next, we need to show that $J(K)$ is minimal. This is again a consequence of the supportedness of J. In fact, assume that J is not minimal, i.e. there exists a model M

of $\mathscr{T}(I)$ such that:

(1) M agrees with J on default propositions

(2) for some $a \in \mathscr{B}$: $a \in J \ \wedge \ a \notin M$.

$a \in J$ means $a \in T_{P\cup K} \uparrow \omega$, i.e. $\exists n.\ a \in T_{P\cup K} \uparrow n$. We show, by induction on n, that

$$a \in T_{P\cup K} \uparrow n \ \Rightarrow \ a \in M. \qquad (*)$$

The base case is obvious. Assume $(*)$ holds for each $h < n$, and let a be such that $a \in T_{P\cup K} \uparrow n \ \wedge \ a \notin T_{P\cup K} \uparrow (n-1)$. We have:

$a \in T_{P\cup K} \uparrow n$

$\Rightarrow$ {definition of $T_{P\cup K} \uparrow n$}

$\exists a \leftarrow b_1,\ldots,b_n,not\_c_1,\ldots,not\_c_m$ in P.

$\{b_1,\ldots,b_n,not\_c_1,\ldots,not\_c_m\} \subseteq T_{P\cup K} \uparrow (n-1)$

$\Rightarrow$ {inductive hypothesis}

$\exists a \leftarrow b_1,\ldots,b_n,not\_c_1,\ldots,not\_c_m$ in P.

$\{b_1,\ldots,b_n\} \subseteq M \ \wedge \ \{not\_c_1,\ldots,not\_c_m\} \subseteq K$

$\Rightarrow$ {property (1) of M}

$\exists a \leftarrow b_1,\ldots,b_n,not\_c_1,\ldots,not\_c_m$ in P:

$\{b_1,\ldots,b_n\} \subseteq M \ \wedge \ \{not\_c_1,\ldots,not\_c_m\} \subseteq M$

$\Rightarrow$ {M is a model of $\mathscr{T}(I)$}

$a \in M$

hence we get a contradiction with (2).

$\Leftarrow$: Let $J \in MIN(\mathscr{T}(I))$, and let $K = J \cap \mathscr{L}^{\mathscr{Q}}$. By Definition 3.3, we have to show:

(0) J is supported by K;

(1) $K \supseteq H$;

(2) $\overline{K} \cap I = \emptyset$.

Proof of (0): Let M be the interpretation supported by K, that is:

- $M^{\mathscr{Q}} = K$
- $M^{\mathscr{C}} = (T_{P\cup K} \uparrow \omega)^{\mathscr{C}}$.

M is a model of $\mathscr{T}(I)$ by construction. Since M agrees with J on default propositions, $M = J$ by the minimality of J, hence J is supported.

Proof of (1):

$not\_a \in H$

$\Rightarrow$ {construction of $\mathscr{T}(I)$}

$not\_a \in \mathscr{T}(I)$

$\Rightarrow$ {J is a model of $\mathscr{T}(I)$}

$not\_a \in J$

$\Leftrightarrow$ {$J^{\mathscr{Q}} = K$}

$not\_a \in K$

Proof of (2):

$a \in I$

$\Rightarrow$ {construction of $\mathscr{T}(I)$}

$\neg not\_a \in \mathscr{T}(I)$

$\Rightarrow$ $\{J$ is a model of $\mathscr{T}(I)\}$
$not\_a \notin J$
$\Leftrightarrow$ $\{J^{\mathscr{D}} = K\}$
$not\_a \notin K$ □

Proposition 3.8. *Let P be a normal program and let T be a theory satisfying condition (S2) of Definition 3.12. Then, J is a minimal model of $T \Leftrightarrow J$ is a conservative extension of $\mathscr{S}(T)$.*

Proof. Let

$$H = \mathscr{S}(T)^{\mathscr{D}} \quad \text{and} \quad K = J^{\mathscr{D}}.$$

$\Rightarrow$: Assume J is a minimal model of T. By Definition 3.3, we need to show:
(1) $H \subset K$
(2) $\overline{K} \cap \mathscr{S}(T) = \emptyset$.
Proof of (1):
$not\_a \in H$
$\Rightarrow$ $\{$construction of $\mathscr{S}(T)\}$
$not\_a \in T$
$\Rightarrow$ $\{J$ model of $T\}$
$not\_a \in J$
$\Leftrightarrow$ $\{J^{\mathscr{D}} = K\}$
$not\_a \in K$
Proof of (2):
$a \in \mathscr{S}(T)$
$\Rightarrow$ $\{$construction of $\mathscr{S}(T)$ and J minimal model of $T\}$
$a \in J$
$\Rightarrow$ $\{$property $(S2)$ of $T\}$
$not\_a \notin J$
$\Leftrightarrow$ $\{J^{\mathscr{D}} = K\}$
$not\_a \notin K$

$\Leftarrow$: Assume J is a conservative extension of $\mathscr{S}(T)$. We need to show:
(i) $not\_a \in T \Rightarrow not\_a \in J$
(ii) $\neg not\_a \in T \Rightarrow not\_a \notin J$
(iii) for each rule $a \leftarrow b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m$ in P
$\{b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m\} \subseteq J \Rightarrow a \in J$
Proof of (i):
$not\_a \in T$
$\Rightarrow$ $\{$by construction of $\mathscr{S}(T)\}$
$not\_a \in H$
$\Rightarrow$ $\{H \subseteq K$, since $J(K) \in CE(\mathscr{S}(T))\}$
$not\_a \in K$

$\Leftrightarrow$ $\{J^{\mathscr{D}} = K\}$
$not\_a \in J$
Proof of (ii):
$\neg not\_a \in T$
$\Rightarrow$ {property ($S2$) of T and construction of $\mathscr{S}(T)$}
$a \in \mathscr{S}(T)$
$\Rightarrow$ $\{\overline{K} \cap \mathscr{S}(T) = \emptyset$, since $J \in CE(\mathscr{S}(T))\}$
$not\_a \notin K$
$\Leftrightarrow$ $\{J^{\mathscr{D}} = K\}$
$not\_a \notin J$
Proof of (iii): Let $a \leftarrow b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m$ be a rule in P such that $\{b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m\} \subseteq J$
Hence $a \in J$ by supportedness of J. The proof of minimality of J is analogous to the proof in Proposition 3.7. □

We can finally prove the main result which states that there is a one-to-one correspondence between complete models and stationary expansions.

Theorem 3.9. *Let P be a normal program. Then*:
(i) *if M is a complete model of P, then $\mathscr{T}(M)$ is a stationary expansion of P*
(ii) *if T is a stationary expansion of P, then $\mathscr{S}(T)$ is a complete model of P.*

Proof. Proof of (i): We have to show that $\mathscr{T}(M)$ satisfies both conditions ($S1$) and ($S2$) of Definition 3.12.

Proof of ($S2$): $\mathscr{T}(M) \models \neg not\_a \Leftrightarrow \mathscr{T}(M) \models a$
$\Leftarrow$:
$\mathscr{T}(M) \models a$
$\Rightarrow$ {definition of $\models$ }
$\forall M \in MIN(\mathscr{T}(M)) : a \in M$
$\Rightarrow$ {Proposition 3.7, M is consistent}
$\forall N \in CE(M) : a \in N^{\mathscr{C}}$
$\Rightarrow$ $\{M \in CE(M)\}$
$a \in M^{\mathscr{C}}$
$\Rightarrow$ {Construction of $\mathscr{T}(M)$}
$\neg not\_a \in \mathscr{T}(M)$
$\Rightarrow$
$\mathscr{T}(M) \models \neg not\_a$
$\Rightarrow$:
$\mathscr{T}(M) \models \neg not\_a$
$\Rightarrow$ {definition of $\models$ }
$\forall M \in MIN(\mathscr{T}(M)) : not\_a \notin M$
$\Rightarrow$ {proposition 3.7, M is consistent}

$\forall N \in CE(M) : not\_a \notin N^{\mathscr{D}}$
$\Rightarrow$ {definition of $CE(M)$}
$a \in M^{\mathscr{C}}$
$\Rightarrow$ {definition of $CE(M)$}
$\forall N \in CE(M) : a \in N$
$\Rightarrow$ {Proposition 3.7}
$\forall M \in MIN(\mathscr{T}(M)) : a \in M$
$\Rightarrow$
$\mathscr{T}(M) \models a$

Proof of (S1): $\mathscr{T}(M) \models not\_a \Leftrightarrow \mathscr{T}(M) \models_{min} \neg a$
$\mathscr{T}(M) \models_{min} \neg a$
$\Leftrightarrow$ {definition of $\models_{min}$}
$\forall M \in MIN(\mathscr{T}(M)) : a \notin M$
$\Leftrightarrow$ {proposition 3.8}
$\forall N \in CE(M) : a \notin N$
$\Leftrightarrow$ {M is a complete model of P }
$not\_a \in M$
$\Leftrightarrow$ {definition of $\mathscr{T}$}
$not\_a \in \mathscr{T}(M)$
$\Leftrightarrow$
$\mathscr{T}(M) \models not\_a$

Proof of (ii): We need to show that $\mathscr{S}(T)^{\mathscr{D}} = \{not\_a | \forall N \in CE(\mathscr{S}(T)) : a \notin N\}$
$not\_a \in \mathscr{S}(T)$
$\Leftrightarrow$ {definition 3.13 of $\mathscr{S}$}
$not\_a \in T$
$\Leftrightarrow$ {T is a stationary expansion}
$\mathscr{T}(M) \models_{min} \neg a$
$\Leftrightarrow$ {definition of $\models_{min}$ }
$\forall M \in MIN(\mathscr{T}(M)) : a \notin M$
$\Leftrightarrow$ {Proposition 3.8 since T satisfies (S2)}
$\forall N \in CE(\mathscr{S}(T)) : a \notin N$ □

As a corollary of Theorem 3.6 and theorem 3.9, we have one important result which states the equivalence between D-complete scenaria and stationary expansions of a normal program P.

Corollary 3.10. *Let P be a normal program. Then*:

(i) *if $P \cup H$ is a D-complete scenario of P then $P \cup H \cup \{\neg not\_a | a \in T_{P \cup H} \uparrow \omega\}$ is a stationary expansion of P.*

(ii) *if T is a stationary expansion of P then $P \cup (T \cap \mathscr{L}^{\mathscr{D}})$ is a D-complete scenario of P.*

As it will be discussed in Section 4, Theorem 3.9 holds for normal logic programs only and does not apply to the case of logic programs with two kinds of negations.

3.2.3. Stable and well-founded semantics

It is also worth observing that both stationary semantics and preferential semantics extend Gelfond and Lifschitz's stable model semantics [44] for normal programs. In fact, the stable models of a program P (if any) coincide with *total* stationary expansions and complete scenaria. Notably, the same result can be restated in our framework as follows.

Proposition 3.11. *Let P be a program and $M \subseteq \mathscr{B}$. Then M is a stable model of P $\Leftrightarrow$ $M \cup \{not\_a | a \notin M\}$ is a total model of P.*

Proof. Straightforward, since from Proposition 3.1(ii) we have that total models (and therefore stable models) are indeed complete, and complete models are equivalent to 3-valued stable models. □

Finally, both Dung and Przymusinski have related their semantics to the well-founded semantics defined in [72]. Indeed, given a program P, both the least complete scenario and the least stationary expansion correspond to the well-founded model of P. Thus, as a consequence of the results of previous sections, the least complete model corresponds to the well-founded model.

Proposition 3.12. *Let P be a program and $W = \bigcap\{M \mid M$ is a complete model of $P\}$. Then W is still a complete model, and it corresponds to the well-founded model of P, that is:*

$$M_W = W^{\mathscr{C}} \cup \{\neg a | not\_a \in W^{\mathscr{D}}\}$$

is the well-founded model of P.

Proof. Straightforward, from Theorem 3.6 and results in [35]. □

4. Other forms of negation

A large body of research (e.g., see [38, 45, 65, 56]) has been recently oriented to increase the expressive power of logic programming by introducing other forms of negation beyond negation by default. Different kinds of negation have been studied in the literature – such as *pseudo*, *classical*, *strong*, *weak* and *explicit* negation – and their usage for knowledge representation purposes has been investigated (e.g., see [10, 46, 52, 60]). A general framework dealing with many of these forms of negation is described in [6].

In this section, we first show how one of these forms of negation, called *pseudo negation*, can be modeled in the framework introduced in the previous section for

normal logic programs. We then discuss how some other forms of negation can be accommodated in our framework.

4.1. Model theory

Let us first extend the language introduced in Section 3, and consider the case of logic programs with two kinds of negation, often referred to as *extended logic programs*. (The positive version of) An extended logic program is a set of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\_l_{m+1}, \ldots, not\_l_{m+n}$$

where each l_i is a standard atom, i.e., either a positive atom a or the syntactic negation of an atom $a^{\neg}$. In this case the set of assumables is taken as the set of default atoms in the underlying Herbrand Base, that is

$$\mathscr{H} = \mathscr{L}^{\mathscr{D}}.$$

Notice, however, that

$$\mathscr{L}^{\mathscr{D}} = \{not\_l \mid l \in \mathscr{L}^{\mathscr{C}}\}.$$

that is, default atoms can be of the form not_a or $not\_a^{\neg}$, a being an atom.

We now define the semantics of negation for programs in this class. If negated atoms of the form $a^{\neg}$ are viewed as new positive atoms, their negation by default $not\_a^{\neg}$ can be treated in the same way as the negation by default not_a of ordinary positive atoms a was treated in the previous section. Under this view, we could think of employing the definitions of supported interpretation, conservative extension, admissible and complete model given in the previous section, and take the complete models as the intended models of an extended program P. Such a view is however too naive, since it does not take into account the non-contradictoriness (in the sense of Definition 2.3(ii)) of complete models. The following example illustrates the point.

Example 4.1. Let P be the program:

$a \leftarrow not\_b$

$a^{\neg} \leftarrow not\_b$

$b \leftarrow not\_a$

The Herbrand Base associated with P is $\mathscr{B} = \{a, a^{\neg}, b, b^{\neg}, not\_a, not\_b, not\_a^{\neg}, not\_b^{\neg}\}$ and the set of assumables is $\mathscr{H} = \{not\_a, not\_a^{\neg}, not\_b, not\_b^{\neg}\}$. Both $\{a, a^{\neg}, not\_b, not\_b^{\neg}\}$ and $\{b, not\_a, not\_b^{\neg}, not\_a^{\neg}\}$ are complete models (in the sense of definition 3.5), but the former is contradictory.

In order to take contradictions into account, we refine the definition of admissible model (Definition 3.4) by replacing it with the following definition of *admissible pre-model*. The definitions of supported interpretation and conservative extension are instead the same as in the previous section (Definitions 2.4 and 3.3).

Definition 4.1. Let P be an extended program. A consistent supported interpretation $I(H)$ is an *admissible pre-model* of P if and only if for any conservative extension $J(K)$ of $I(H)$: $\overline{H} \cap J = \emptyset$.

Notice that the above definition differs from Definition 3.4 for normal programs since in the former admissible consistent supported interpretations are called pre-models rather than models. Such a distinction is introduced since admissible consistent supported interpretations of extended programs may be contradictory.

Similarly, the analogous of complete models for normal programs are the complete pre-models of the next definition.

Definition 4.2. Let P an extended program. An admissible pre-model $I(H)$ is a *complete pre-model* of P if and only if the following property holds:

$$(\forall l \in \mathscr{L}^{\mathscr{C}}.(not\_l \in H \Leftrightarrow (\forall J \in CE(I(H)) : l \notin J))).$$

Definition 4.3. Let P be an extended program. An admissible (viz., complete) pre-model $I(H)$ is an *admissible* (viz., *complete*) *model* of P if and only if it is non-contradictory.

The intended meaning of an extended program P is therefore defined by the set of its complete models according to Definition 4.3.

There are however programs which are "inherently" contradictory, as the following example shows.

Example 4.2. Let P be the program:

$a \leftarrow$

$a^{\neg} \leftarrow$

The Herbrand Base associated with P is $\mathscr{B} = \{a, a^{\neg}, not\_a, not\_a^{\neg}\}$ and the set of assumables is $\mathscr{H} = \{not\_a, not\_a^{\neg}\}$. The interpretation $\{a, a^{\neg}\}$ is the only complete pre-model of P, but it is not a complete model, since it is contradictory.

One way to deal with contradictions is to restrict the class of extended programs for which the semantics is defined. We therefore introduce a notion of inherently *non-contradictory program* so as to rule out programs such as the program of Example 4.2. A natural condition is to require that the interpretation supported by the empty set of assumables is non-contradictory. For instance, in example 4.2, the interpretation supported by the empty set of assumables is contradictory, and hence any supported interpretation is contradictory.

Definition 4.4. An extended program P is *non-contradictory* if and only if the supported interpretation $I(\emptyset)$ is non-contradictory.

It is worth observing that the non-contradictoriness of a program P does not still ensure the existence of a complete model for P, as illustrated by the following example.

Example 4.3. Let P be the program:

$a \leftarrow not\_b$

$a^{\neg} \leftarrow not\_b$

The Herbrand Base associated with P is: $\mathscr{B} = \{a, a^{\neg}, b, b^{\neg}, not\_a, not\_b, not\_a^{\neg}, not\_b^{\neg}\}$ The program is non-contradictory, since the interpretation supported by the empty set of assumables is simply $\{\}$. The only complete pre-model of P is $\{a, a^{\neg}, not\_b, not\_b^{\neg}\}$, which is not a model of P being contradictory.

Summing up, we defined the semantics of a non-contradictory extended logic program P as the set of complete models for P. We also observed that, even by restricting to non-contradictory programs, there are programs for which no complete model exists because of contradictions. In section 4.3 we shall discuss how contradictions can be dealt with.

4.2. Related work

In this section, we discuss the relations between the semantics defined in Section 4.1 and other approaches to the semantics of extended logic programs. For the moment, we do not consider approaches dealing with contradiction which will be considered in Section 4.4.

First of all, we clarify the form of negation we dealt with (namely, *pseudo negation*) by referring to the paper by Alferes and Pereira [6], where a parameterizable schema is defined which encompasses and characterizes a diversity of proposed semantics for extended logic programs. Among various forms of negation, in this section we consider strong negation, classical negation (and the stationary semantics, in particular, [65]) and explicit negation coping with the *coherence principle* [56]. In particular, we relate our approach with the (well-founded) semantics defined for *explicit negation* by Pereira and Alferes [56].

Moreover, we show that our complete models (definition 4.3) capture both answer set semantics [45] and Przymusinski's 3-valued stable semantics [64] for extended logic programs. Differently from [45, 64], the complete model semantics is defined without any syntactic transformation on program clauses, but through a rather intuitive concept based on the extensibility of Herbrand interpretations.

Finally, we mention the relationships with other semantics of extended logic programs such as the abductive semantics by Brewka and Konolige [19, 18].

4.2.1. Other forms of negation

Alferes and Pereira define in [6] a parameterizable scheme to characterize a variety of semantics proposed for extended logic programs. The scheme is parameterized w.r.t. two parameters: A set of axioms $AX_{\neg}$ defining $\neg$-negation, and a minimality condition defining *not*-negation. They show how different semantics for programs with two kinds of negation [45, 64, 65, 56] can be modeled by suitably instantiating these two

parameters. They also show how other semantics dealing with contradiction removal (e.g., [57]) can be accommodated in the scheme.

It is worth observing that the complete pre-models of an extended program P correspond to the stationary $AX_{\neg}$ expansions of P, where the set of axioms $AX_{\neg}$ is empty. Such a correspondence directly follows from the results of Section 3 and from the definition of stationary $AX_{\neg}$ expansions of [6]. Complete pre-models hence characterize the form of negation called *pseudo negation* in [6]. Moreover, while the semantics for pseudo negation might be inconsistent, complete models define a consistent version of stationary semantics with pseudo negation.

Strong negation is another form of negation defined in [6] by means of a set of AX_{strong} axioms of the form $not\_a \Rightarrow \neg A$, one for each literal A. Complete models do not model properly strong negation, as illustrated by the following example.

Example 4.4. Consider the program P:

$b^{\neg} \leftarrow$

$a \leftarrow not\_a$

$b \leftarrow a$

As shown in [6], no meaning is associated with P when the syntactic negation $\neg$ is interpreted as strong negation. On the other hand, complete models do assign a meaning to P since $\{b^{\neg}, not\_a^{\neg}\}$ is a complete model of P.

It is however worth observing that the enforcement of the totality requirement satisfies the axioms AX_{strong} for strong negation (as well as the *coherence* principle required by *explicit negation* discussed below). There is hence a one-to-one correspondence between the complete *total* models of a program P and the stable AX_{strong} models of P (as well as with the answer sets of P, being these equivalent to stable AX_{strong} models [6]).

Alferes and Pereira [6] point out that some semantics for extended logic programs are not *supportive*. The semantics of *classical* and *weak negation* are examples of non supportive semantics.

Definition 4.5 (Alferes and Pereira [6]). A semantics is necessarily supportive if, for any program P, whenever M is a stationary (resp. stable) model of P then, for every standard literal l if $l \in M$ then there exists at least one rule in P of the form $l \leftarrow b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m$ such that $\{b_1, \ldots, b_n, not\_c_1, \ldots, not\_c_m\} \subseteq M$.

It is easy to observe that our semantics cannot directly reconstruct non supportive semantics, since our interpretations are all supported by definition. Stationary semantics [65] deals with classical negation as well as with negation by default, and it relies on the principles of minimization and predicate circumscription (see Section 3.2.2 for the case of normal logic programs). The equivalence result between complete models and stationary expansions stated in Section 3.2.2 (Theorem 3.9) for normal logic programs does not extend to the case of extended logic programs.

Example 4.5. Consider the program:

$p \leftarrow \neg a$

$p \leftarrow a$

The meaning of this program in our semantics is the complete model $\{not\_a, not\_a^{\neg}, not\_p, not\_\neg p\}$. On the other hand, stationary semantics allows one to conclude that p holds, because of the *tertium non datur* property of classical negation.

One way to reconstruct non-supportive semantics in our framework is to extend the program with rules suitably coping with the axioms of the various semantics. For instance, Pereira and Alferes [6] show that classical negation can be reconstructed by introducing disjunction axioms of the form $a \vee \neg a$, and strong negation axioms of the kind $not\ a \Rightarrow \neg a$.

Pereira and Alferes introduced in [56] another form of negation, called *explicit negation*, where default and explicitly negated literals are related by a *coherence principle*. The coherence principle can be defined by means of inference rules of the form $\neg L \rightarrow not\ L$, one for each standard atom $L \in \mathscr{L}^{\mathscr{C}}$. The authors define a well-founded semantics for extended logic programs in terms of the least extended stable model (WFSX model).

Example 4.6. Let P be the extended program (taken from [56]):

$c \leftarrow a$

$a \leftarrow b$

$b \leftarrow not\_b$

$a^{\neg} \leftarrow$

The only extended stable model of P is $N = \{a^{\neg}, not\_a, not\_c, not\_c^{\neg}, not\_b^{\neg}\}$. Namely, the coherence principle enforces the truth of the default assumption not_a, since the last clause of P makes $a^{\neg}$ true. According to the transformation defined in [56], the first clause is deleted, and no clause for c is left, thus allowing the assumption of the default not_c. In our semantics, instead, P is associated with the (unique) complete supported model $M = \{a^{\neg}, not\_c^{\neg}, not\_b^{\neg}\}$.

The semantics defined in [56] therefore differs from our semantics, since we do not consider the coherence principle. One way to reconstruct the semantics presented in [56] is to enforce the coherence principle, and to perform a suitable program transformation on program clauses so as to replace each standard atom L by the pair $L, not\_L^{\neg}$ [1].

Example 4.7. Let us consider program P of example 4.6, and let us transform its clauses in order to incorporate the coherence principle (see also [5]). We obtain the new program P_3:

$c \leftarrow a, not\_a^{\neg}$

$a \leftarrow b, not\_b^{\neg}$

$b \leftarrow not\_b$

$a^{\neg} \leftarrow$

It is easy to observe that P and P_3 have the same extended stable model, namely

$$N = \{a^{\neg}, not\_a, not\_c, not\_c^{\neg}, not\_b^{\neg}\}$$

Notice also that the complete supported model of P_3 is $S = \{a^{\neg}, not\_c, not\_c^{\neg}, not\_b^{\neg}\}$, and that by enforcing coherence on S we obtain exactly the extended stable model N.

Recently, Alferes et al. [10] analyze the properties of strong and explicit negation, and their relation. They also analyze the relation between strong and explicit negation and default and classical negation, in the general context of knowledge representation and in terms of autoepistemic logic of beliefs. They show that any logic program P can be transformed into a belief theory, and that there exists a one-to-one correspondence between the stationary models of P [65] and the consistent static autoepistemic expansions of the corresponding belief theory [10]. Theorem A.2 of [10] and Theorem 3.9 establish the correspondence between our complete models of a program P and the consistent static autoepistemic expansions of the belief theory obtained from P. Notice that, in a logic programming setting, the results presented in [10] also show that strong and explicit negation coincide with the Gelfond–Lifschitz "classical" negation for (extended) logic programs under the stable (or answer set) semantics [45]. This result applies smoothly to our *total* complete models, for which both strong and explicit negation coincide.

4.2.2. 3-Valued stable semantics

Przymusinski extends in [64] the framework of 3-valued stable models to the case of logic programs with two kinds of negation. For normal logic programs, 3-valued stable models are defined through an extended Gelfond–Lifschitz's transformation and a stability requirement. The framework is adapted to the case of extended logic programs by viewing negated literals $\neg a$ as new atoms $a^{\neg}$, as we have done so far. An extended logic program P is then transformed into a normal logic program P', and the 3-valued stable semantics of the latter is exploited to provide the former with a 3-valued stable semantics.

Let M' be a 3-valued stable model of P'. The corresponding candidate 3-valued model M of P is obtained as follows:

(i) If a (resp. $a^{\neg}$) is true in M', then a (resp. $\neg a$) is true in M;

(ii) If a (resp. $a^{\neg}$) is false in M', then *not* a (resp. *not* $\neg a$) is true in M, that is a (resp. $\neg a$) is false by default;

(iii) if a (resp. $a^{\neg}$) is undefined in M', then a (resp. $\neg a$) is undefined in M.

If the resulting set M is non-contradictory then M is a 3-valued stable model of P, it is discarded otherwise.

Each complete model of P corresponds to a 3-valued stable model of P and vice-versa, as stated by the following proposition.

Proposition 4.1. *Let P be an extended program. Then: M is a complete model of $P \Leftrightarrow M$ is a 3-valued stable model of P.*

Proof. The proof follows from the results given in Section 3 and in [65], and by viewing P as a normal logic program.

($\Rightarrow$) Straightforward, from Theorem 3.9 and results in [65]. Indeed (non-contradictory) complete pre-models correspond to the stationary expansions of P and therefore to the (non-contradictory) 3-valued stable models of P.

($\Leftarrow$) Straightforward, since the 3-valued stable models of P (when viewed as a normal logic program) correspond to the complete models of P. □

Example 4.8. Let us consider the following program P:

$a \leftarrow not\_b$
$b \leftarrow not\_a$
$a^{\neg} \leftarrow not\_c$
$b^{\neg} \leftarrow not\_c$

$\{a^{\neg}, b^{\neg}, not\_c, not\_c^{\neg}\}$ is the only complete model of P, and it coincides with the unique 3-valued stable model of P.

4.2.3. Answer set semantics

Gelfond and Lifschitz present in [45] the answer set semantics as a generalization of the (2-valued) stable model semantics for extended programs. It is easy to show that the answer sets of an extended program P are stable models of the program obtained from P by viewing negated literals $\neg a$ as new positive atoms. Using our notation, an answer set of P is an interpretation $S \subseteq \mathscr{L}^{\mathscr{C}}$ such that S is a stable model of the normal logic program obtained from P by replacing each literal $\neg a$ with $a^{\neg}$. If S contains a pair of complementary literals (i.e., S is contradictory), then S is mapped into the whole set of standard atoms $\mathscr{L}^{\mathscr{C}}$.

In Section 3.2, we have shown that total consistent supported interpretations are indeed complete and that they coincide with the stable models of a normal logic program. In the case of extended logic programs a similar result holds.

Proposition 4.2. *Let P be an extended program. Then:*

(i) *any total model M of P corresponds to an answer set of P;*

(ii) *any non-contradictory answer set of P corresponds to a total model of P.*

Proof. Trivial from Proposition 4.1. □

Example 4.9. Let us consider the following program P:

$a \leftarrow not\_b$
$b \leftarrow not\_a$
$a^{\neg} \leftarrow$

The only total model of P is $\{b, a^{\neg}, not\_a, not\_b^{\neg}\}$, and it coincides with the answer set of P.

4.2.4. Scenario semantics with coherence principle

Alferes et al. present in [5] a unifying framework for studying *explicit* negation, based on the notion of admissible scenarios [35] and on the *coherence principle* [56]. They introduce an "ideal skeptical semantics", along with its well-founded semantics counterpart.

The *coherence principle* implies that there are some *mandatory* hypotheses that must be assumed. More formally, the set of mandatory hypotheses with respect to a scenario $P \cup H$ is

$$Mand(H) = \{not\_L | P \cup H \cup \{not\_L \leftarrow L^{\neg} | L \in \mathscr{L}^{\mathscr{C}}\} \vdash not\_L\}.$$

Alferes et al. consider *consistent* programs, where an extended logic program P is consistent iff $P \cup Mand(\{\})$ is a consistent scenario. Any consistent program has at least one admissible scenario.

The ideal skeptical semantics is defined as the largest set of hypotheses H satisfying the following condition "For each admissible scenario $P \cup K$: $P \cup K \cup H$ is also admissible". The well-founded semantics is defined as the *grounded* part of the ideal skeptical semantics. More precisely, the well-founded (skeptical) semantics of a program P is the least fixpoint of a bottom-up process, which starts with the empty scenario, extends it at each step with mandatory hypotheses belonging to the ideal skeptical scenario. Besides the skeptical semantics, Alferes et al. characterize in [5] more credulous semantics in terms of complete scenaria.

While our framework does not tackle the skeptical semantics addressed in [5], we can briefly discuss the relations between our framework and the more credulous semantics. The notions of admissibility and completeness we presented can be made equivalent to that of [5] provided that mandatory hypotheses are added to our interpretations. More precisely, mandatory hypotheses may be accommodated in our framework by means of a suitable syntactic transformation over program clauses (as suggested for dealing with explicit negation).

4.2.5. Abductive semantics

Brewka and Konolige presented in [19] an abductive semantics for generalized propositional logic programs by defining the meaning of a program in terms of its extensions. Default literals are modeled as *abductive* hypotheses, and a logic program is considered as the background theory. A criterion for defining the acceptable hypotheses, called *extension bases*, is introduced. Their semantics applies also to programs with two kinds of negation, and disjunctions in clause heads.

Given a program P and a set of hypotheses H (i.e., default literals), the *P-closure* of H is the interpretation supported by H if it is both consistent and non-contradictory.

Otherwise the *P-closure* of H is the overall Herbrand base. Given a P-closure C of a set of hypotheses H for a program P, the *P-cover* of C is then defined as

$$COV_P(C) = H \cup \{not\_a \in \mathscr{L}^{\mathscr{D}} \mid a \in C\}.$$

Then C is an extension of P if it is consistent and non-contradictory (according to our terminology) and there is no other set of hypotheses H' whose P-closure C' is such that $COV_P(C) \subset COV_P(C')$. The set of hypotheses H supporting C is called an extension base of P.

The preference criterion adopted by Brewka and Konolige models the intuition that undefinedness should be minimized. From our perspective, Brewka and Konolige select those interpretations that are consistent and non-contradictory, and that assign a truth value to as many literals as possible. In this respect, their semantics is more credulous than ours or others semantics such as well-founded or stationary semantics. As in our case, the resulting semantics is defined for any non-contradictory program.

The semantics defined in [19] is equivalent to the stable model semantics for programs having at least one stable model. This property can be easily accommodated in our framework by enforcing a preference criterion for *total* models (if they exist) over partial models.

The main drawback of the semantics in [19] is however that, when no stable model exists, it notably departs from 3-valued stable model semantics [64] and stationary semantics [65]. On the other hand, it solves for instance the "floating conclusion" problem of well-founded semantics and stationary semantics.

Example 4.10. Consider the program P:

$a \leftarrow not\_b$
$b \leftarrow not\_a$
$c \leftarrow a$
$c \leftarrow b$

The well-founded semantics does not conclude c, whereas abductive semantics is defined by two extension bases, namely $\{not\_b\}$ and $\{not\_a\}$, both of which derive c, being their P-closures $\{a, not\_b, c\}$ and $\{not\_a, b, c\}$, respectively.

It is worth mentioning that Brewka in [18] adopts the same approach but also introduces the coherence principle of Pereira and Alferes [56] thus taking into account explicit negation.

4.3. Dealing with contradiction

The introduction of a second form of negation may lead to inconsistencies in the models of an extended program, namely, the presence of both A and $\neg A$. There are several different ways of handling such inconsistencies.

(1) Inconsistencies may be allowed to *trivialize* the knowledge represented by a program, so that any conclusion can be derived, as for instance in the answer set semantics [45].

(2) *Paraconsistent* semantics can be adopted, that is, semantics allowing the derivation of inconsistencies without trivialization (for a recent, general reference to paraconsistency in Logic Programming see [29]). Intuitively speaking, paraconsistent semantics aim at modeling the situation in which a program contains incorrect information due to a human error. This approach may be employed for program diagnosis and intelligent debugging (e.g., see [61]).

(3) Belief revision techniques may be employed to perform *contradiction removal.* Intuitively speaking, inconsistencies stem from assumptions which are no longer warranted rather than from erroneous information. Such assumptions are then removed (possibly in a minimal way) in order to restore consistency.

As pointed out in Section 4.1, when considering two kinds of negation, the semantics based on complete pre-models may lead to contradictions. Pre-models are *de facto* paraconsistent models, and they correspond to a paraconsistent version of the semantics proposed by Przymusinski [64]. Non-contradictory programs have, by definition, a supported interpretation that is always admissible and non-contradictory (viz., the interpretation supported by the empty set of assumables). As shown in Example 4.3, however, there may be problems when the completeness requirement imposed on admissible models interferes with the non-contradictoriness requirement.

We now discuss a *contradiction removal* approach to give meaning to any non-contradictory program. The intuition underlying the semantics with contradiction removal (see Section 4.4) is, roughly speaking: "if an assumption supports a contradiction, then take back that assumption". In our approach, the admissibility criterion is taken as the basis for defining the meaning of negation by default. The completeness requirement can be then relaxed so as to identify as intended models of a program P the admissible supported interpretations that contain as many assumables as possible, still being consistent and non-contradictory.

Intuitively speaking, given a contradictory complete pre-model, the idea is to withdraw from it as few assumables as possible, in order to restore non-contradictoriness. The intended models of an extended program P are then those admissible models which are maximal subsets of its complete pre-models.

Definition 4.6. Let P be a non-contradictory extended program. A supported interpretation M is a *contradiction-removal model* of P iff it is an admissible model and is a maximal (w.r.t. set inclusion) subset of some complete pre-model N of P.

Example 4.11. Let P be the program:

$a \leftarrow not\_b$

$a^{\neg} \leftarrow not\_c$

P is clearly non-contradictory, but its only complete pre-model $\{a, a^{\neg}, not\_b, not\_c, not\_b^{\neg}, not\_c^{\neg}\}$ is contradictory. The contradiction-removal models of P are $M_1 =$

$\{a, not\_b, not\_b^{\neg}, not\_c^{\neg}\}$ and $M_2 = \{a^{\neg}, not\_c, not\_b^{\neg}, not\_c^{\neg}\}$, which are obtained by removing *not_c* and *not_b*, respectively.[5]

The contradiction-removal semantics is defined for any non-contradictory extended program, as stated by the following proposition.

Proposition 4.3. *Let P be a non-contradictory extended logic program. Then, the set of contradiction-removal models of P is not empty.*

Proof. Since P is non-contradictory, there is always an admissible model of P ($I(\emptyset)$). From Proposition 3.1(i), there exists always a complete pre-model of P. For each complete pre-model M of P, the set of supported interpretations which are subsets of P and which are admissible models of P is non-empty (at least, $I(\emptyset)$). □

Finally, it is worth observing that the set of contradiction-removal models includes the set of complete models.

Proposition 4.4. *Let P be a non-contradictory extended logic program. Then any complete model of P is a contradiction-removal model of P.*

Proof. Straightforward, since any complete model is indeed admissible and maximal. □

4.4. Dealing with contradiction: related work

The problem of removing contradiction from inconsistent programs has been approached by many authors (e.g., [57, 60, 4, 7–9, 53, 62, 67]).

In this section, we consider other approaches coping with contradiction. In order to give meaning to a program, two main approaches can be followed. One is *contradiction removal*, in which hypotheses are discarded (possibly in a minimal way) in order to restore the non-contradictoriness of a "model" despite its completeness. This is the basic approach we follow.

Alternatively, *contradiction avoidance* can be applied by avoiding to assume an hypothesis (even if acceptable) if contradiction would arise when assuming it.

In the following, we prove that our contradiction-removal semantics (Definition 4.6) is equivalent to complete scenario semantics introduced by Dung and Ruamviboonsuk in [41], even if their approach is based on *contradiction avoidance* rather than on contradiction removal. The work by Dung and Ruamviboonsuk originated from the argumentation view of negation in logic programming introduced in [35]. A further refinement (dealing with contradiction) of the argumentation approach has been defined in [38].

[5] Notice that, for extended programs, it may happen that there is no least model. However, there is one single admissible model which is the meet of the contradiction removal models in the lower semi-lattice made of Contradiction Removal Models ordered by their supportive hypotheses.

Besides relating our work with these semantics, we will also relate our approach with other semantics considering explicit negation rather than pseudo negation and based on contradiction removal [57, 60, 58, 4, 9] or contradiction avoidance [7, 8].

4.4.1. Argumentation semantics

Dung shows in [39] that the most of the approaches to non-monotonic reasoning in artificial intelligence and logic programming can be expressed as different forms of argumentation. This idea is applied in [35] to develop a simple, intuitive framework for the semantics of normal logic programs where many proposed approached to negation by default are unified (see Section 3.2.1). The argumentation framework is then extended further in [38, 41] to deal with the case of extended logic programs.

Let us first consider the work by Dung and Ruamviboonsuk [41]. In order to simplify the comparison, following Section 3.2.1, we restate the definitions given in [41] in terms of argumentation by employing the notion of attack. The semantics presented in [41] can be formulated by means of the notion of attack (Definition 3.6) and scenario (Definition 3.7), and by properly extending the notion of admissibility (see below) so as to take into account a second form of negation.

Definition 4.7. Let P be an extended logic program. A scenario $P \cup H$ is *DR-admissible* if and only if

$$\forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } H \Rightarrow H \text{ attacks } E$$

and $P \cup H$ *is consistent and non-contradictory.*

The definition of (D-)complete scenario (Definition 3.10) remains unchanged as in the case of normal logic programs. We will however talk about *DR-complete scenario* to avoid confusion.

Definition 4.8. Let P be an extended logic program. A DR-admissible scenario $P \cup H$ is *DR-complete* if and only if

$$H = \{not\_a \in \mathscr{L}^{\mathscr{Q}} \mid \forall E \subseteq \mathscr{L}^{\mathscr{Q}}.\ E \text{ attacks } \{not\_a\} \Rightarrow H \text{ attacks } E\}$$

Dung and Ruamviboonsuk [41] follow a *contradiction avoidance* approach by considering non-admissible hypotheses whose assumption leads to a contradiction. The correspondence between complete scenaria and complete models, established in Section 3.2 for the case of normal programs, extends smoothly to the case of extended programs. Namely, we prove that our *contradiction removal* semantics is equivalent to the *contradiction avoidance* approach of [41]. To establish the equivalence between the contradiction-removal models of an extended program P and the DR-complete scenaria of P, we first introduce the following lemma.

Lemma 4.5. *Let P be an extended program and $H \subseteq \mathscr{L}^{\mathscr{Q}}$. Then, $M(H)$ is an admissible model of $P \Leftrightarrow P \cup H$ a DR-admissible scenario of P.*

Proof. $\Leftarrow$: Assume $P \cup H$ is a DR-admissible scenario. Then, it is also D-admissible according to Definition 3.8, and therefore $M(H)$ is an admissible pre-model of P. Being $P \cup H$ a scenario it is non-contradictory, and therefore we have that $M(H)$ is an admissible model of P.

$\Rightarrow$: Assume $M(H)$ is an admissible model of P. Then $P \cup H$ is a D-admissible scenario according to theorem 3.5. Being $M(H)$ a model of P, then the scenario $P \cup H$ is consistent and non-contradictory, and therefore we have that it is also DR-admissible. □

Proposition 4.6. *Let P be an extended logic program, $H \subseteq \mathscr{L}^{\mathscr{D}}$ and $M(H)$ the interpretation supported by H. Then, $M(H)$ is a contradiction-removal model of $P \Leftrightarrow$ $P \cup H$ is a DR-complete scenario.*

Proof. $\Rightarrow$: Assume $M(H)$ is a contradiction-removal model of P. We have to prove that

(1) $M(H)$ is supported $\Rightarrow$ $P \cup H$ is a scenario,

(2) $M(H)$ is an admissible model $\Rightarrow$ $P \cup H$ is a DR-admissible scenario,

(3) $M(H)$ is a maximal subset of some complete pre-model of $P \Rightarrow P \cup H$ is a DR-complete scenario

Statements (1) and (2) are trivial. The proof of (3) is given by contradiction. Assume that $P \cup H$ is not DR-complete. Then, $\exists\ not\_a \in \mathscr{L}^{\mathscr{D}}$ such that $\forall E \subseteq \mathscr{L}^{\mathscr{D}}$. E attacks $\{not\_a\} \Rightarrow H$ attacks E but $not\_a \notin H$ and $H \cup \{not\_a\}$ is non-contradictory. Then, $P \cup H \cup \{not\_a\}$ is a DR-admissible scenario of P, and from Lemma 4.5 $M'(H \cup \{not\_a\})$ is an admissible model of P. But then we get contradiction, since $M(H)$ is not maximal.

$\Leftarrow$: From Lemma 4.5, we have that $M(H)$ is an admissible model of P. The rest of the proof is done by contradiction. Let us suppose that $M(H)$ is not a maximal subset of some complete pre-model of P. Then, $\exists not\_a \in \mathscr{L}^{\mathscr{D}}$ such that $M'(H \cup \{not\_a\})$ is an admissible model of P. But then we get contradiction, since the scenario $P \cup H \cup \{not\_a\}$ is not DR-complete. □

Intuitively speaking, the equivalence between the contradiction-removal models of an extended program P and the DR-complete scenaria of P derives from the fact that, in our framework, the new condition of admissibility (Definition 4.7) is obtained by removing from complete pre-models the default literals introducing contradictions.

Example 4.12. Let P be the program of Example 4.8. The contradiction-removal supported models of P are: $\{a^{\neg}, b^{\neg}, not\_c, not\_c^{\neg}\}$, $\{a, not\_b, not\_c^{\neg}\}$, and $\{b, not\_a, not\_c^{\neg}\}$, and they correspond to the DR-complete scenaria of P.

Propositions 4.1, 4.2 and 4.6 allow us to relate the three semantics for extended logic programs that we have considered. Given an extended program P, let $AS(P)$, $3SM(P)$ and $CS(P)$ denote the non-contradictory answer sets [45], the 3-valued stable models [64] and the DR-complete scenario [41] of P, respectively. The following proposition

shows that DR-complete scenario semantics subsumes 3-valued stable semantics, which in turn subsumes answer set semantics [45]. Indeed total supported models are complete models of P, and complete models are in turn contradiction-removal supported models.

Proposition 4.7. *Let P be an extended logic program. Then:*

$$AS(P) \subseteq 3SM(P) \subseteq CS(P).$$

Proof. Trivial since non-contradictory answer sets of P correspond to total models of P (Proposition 4.2), total models are indeed complete models, and complete models correspond, in turn, to 3-valued stable models (Proposition 4.1). Moreover, complete models are contradiction-removal models (Proposition 4.4), and they, in turn, correspond to DR-complete scenario (Proposition 4.6). □

In [38] Dung presents an argumentation-based semantics for extended logic programs where the notion of attack is further refined by distinguishing between *ground* attacks (g-attacks) and *Reductio Ad Absurdum* attacks (RAA-attacks). Namely, the former notion of attack is the one presented in Section 3.2.1, while the latter is introduced for considering attacks caused by contradictions arising in extended logic programming.

Definition 4.9. Let P be an extended program and let $H, K \subseteq \mathscr{L}^{\mathscr{D}}$ be two sets of assumables supporting non-contradictory interpretations. Then we say that H *RAA-attacks* K if and only if

$$M(H \cup K) \text{ is contradictory.}$$

An attack can be either a g-attack or an RAA-attack. This leads to the following definition of admissible set of hypotheses (scenario).

Definition 4.10. Let P be an extended program. A scenario $P \cup H$ is *admissible* if and only if

$$\forall E \subseteq \mathscr{L}^{\mathscr{D}}.\ E \text{ attacks } H \Rightarrow H \text{ g-attacks } E$$

Notice that, differently from our approach and from [41], the notion of admissibility does not rely on a symmetric notion of attack. Namely the *attacking* agent (i.e., K in Definition 4.10) has more arguments than the *counter-attacking* one (i.e., H). The notion of complete scenario is then introduced in the usual way (see Section 3.2.1).

Both our semantics and the argumentation semantics of [38] have the nice property of being defined for any non-contradictory logic program. Our approach however defines a more credulous semantics, as illustrated by the following example.

Example 4.13. For the following extended program:

$a^{\neg} \leftarrow not\_q$

$a \leftarrow not\_p$

$b \leftarrow not\_r$

$\{not\_r\}$ is a complete scenario but not $\{not\_r, not\_q\}$ and $\{not\_r, not\_p\}$. On the other hand, we have the contradiction-removal models: $\{a^{\neg}, b, not\_r, not\_q\}$ and $\{a, b, not\_r, not\_p\}$. □

4.4.2. Contradiction and explicit negation

In this section, we briefly review some works dealing with contradiction removal and contradiction avoidance semantics for explicit negation. All these works are inspired on the well-founded semantics [72] and on the coherence principle [56].

Pereira et al. [57, 60, 58] define a semantics that extends the *WFSX* semantics for explicit negation [56]. Roughly speaking, the key idea is to remove some contradictions and to identify the models of the program obtained by revising the Closed World Assumptions (*CWA*) supporting such contradictions. More precisely, the *CWA* is relaxed by taking back assumptions (viz., default literals) in a *minimal* way by means of a simple syntactic transformation of the original program. The authors show that the proposed technique of contradiction removal is also adequate to deal with general kinds of contradictions, such as those arising from the violation of integrity constraints.

The contradiction removal technique described in [57] consists of taking back a minimal set of default assumptions that are false under the *CWA*, the so-called *revisable literals*. The definition of revisable literal and the associated notion of *contradiction removal set* – the minimal set of revisable literals supporting contradiction – is further refined in [60, 58].

It is worth observing that the method proposed by Pereira et al. is more selective than ours in taking back assumptions. Namely, they can make undefined only those default literals whose truth under *CWA* does not depend on the truth under *CWA* of other default literals. In our approach, instead, any default literal is – in principle – revisable.

Example 4.14. Consider the program:

$a^{\neg} \leftarrow$
$a \leftarrow not\_b$
$b \leftarrow c$
$c \leftarrow d$

taken from [60]. The only revisable literal is not_d, while in our approach any default literal can be made undefined, such as for instance not_b which depends on d.

The semantics framework employed in [57, 60] is the well-founded semantics for *explicit* negation. On the other hand, we apply contradiction removal to stable semantics for *pseudo* negation. It is worth noting that, in the absence of contradictions, the semantics of [60] coincides with the *WFSX* semantics of [56]. Similarly, when complete pre-models are non-contradictory, our semantics coincides with the 3-valued stable semantics of [64].

Alferes, Damasio and Pereira further generalize in [4] previous approaches to contradiction removal. They consider an extended language in order to deal with a general

form of integrity constraints, beyond denials of the form $\leftarrow L, not\_L$. The revised, non-contradictory version of a contradictory program P is obtained from P without introducing new predicates. The original program is modified by introducing or removing rules for the literals that depend on no other literals. The program is partitioned into two parts: A subprogram that can be changed by adding rules for the revisable literals only, and a subprogram that is left unchanged. The revised, non-contradictory program, is obtained by composing these two parts. In this perspective, the program is view as *open* (see also Section 6). Moreover, while we remove contradiction by modifying the truth value of literals into undefined only, in [4] the truth value can be changed from any value into any other value. The contradiction removal approach proposed in [4] is therefore more general than ours, and suggests an alternative way of removing contradiction in our framework. Namely, we can view programs as *open* as well, and add a variable part to each program so as to change the truth value of revisable literals to any value (not only to *unknown*, as in the original approach). More precisely, the truth value of a literal L (resp., $\neg L$) can be changed to *true* by adding the rule $L \leftarrow$ (resp. $\neg L \leftarrow$), and to *unknown* by adding the rule $L \leftarrow not\_L$ (resp. $\neg L \leftarrow not\_\neg L$). Then, as in [4], the revised program is obtained by composing the fixed and variable components. Notice also that, thanks to the compositionality properties of our approach (see Section 6), the supported interpretations of each composition of programs can be directly obtained by the supported interpretations of the separate sub-programs.

Example 4.15. Consider again program P of Example 4.11:

$a \leftarrow not\_b$

$a^{\neg} \leftarrow not\_c$

The complete pre-model of P

$$\{a, a^{\neg}, not\_b, not\_c, not\_b^{\neg}, not\_c^{\neg}\}$$

is contradictory. The contradiction-removal models of P are

$$M_1 = \{a, not\_b, not\_b^{\neg}, not\_c^{\neg}\}$$

$$M_2 = \{a^{\neg}, not\_c, not\_b^{\neg}, not\_c^{\neg}\}$$

which are obtained by building two revisions of P, one obtained by adding as variable part the program R_1:

$c \leftarrow not\_c$

and the other obtained by adding as variable part the program R_2:

$b \leftarrow not\_b$

The contradiction-removal model M_1 turns out to be a complete model of $P \cup R_1$, and M_2 the complete model of $P \cup R_2$.

Alferes and Pereira propose in [7] a *contradiction avoidance* semantics defined in terms of optative hypotheses. In particular they extend the argumentation framework

defined in [5] where complete scenaria semantics is defined for extended logic programs with negation by default and explicit negation. A program may have no complete scenario, even if it is non-contradictory. Alferes and Pereira in [7] allow the programmer to specify which hypotheses used to build scenaria are *optative*. Optative hypotheses can be accepted or not, and a scenario might not be complete with respect to the optatives. Namely scenaria may be partially complete, that is, complete with respect to non-optatives, but possibly not complete with respect to optatives. The equivalence between contradiction avoidance as defined in [7] and the contradiction removal approach of [9] has been proved in [8, 9].

4.4.3. Other approaches

The problem of removing contradiction from inconsistent programs has been approached by many other authors (e.g., [53, 62, 67]).

In [62] Pimentel and Rodi address the issue of belief revision in terms of *stable revisions*. A stable revision is a minimal modification of a stable model constructed so as to eliminate inconsistencies through the retraction of assumptions. The authors consider the language of symmetric logic programs, including a modal operator M and a form of negation $\neg$, and show that this language can be easily mapped into the language of extended logic programs. Their approach is very similar to ours. They do not trivialize inconsistent answer sets of the program, but rather remove inconsistencies by minimally changing the assumptions of the program corresponding to default literals. They also present an implementation of the technique through a Truth Maintenance System [34].

In [67] Sakama extends the well-founded semantics to the case of paraconsistent logic programs. Contradiction is addressed by identifying suspicious facts, that is, facts whose proofs include some inconsistent information. Moreover, facts supported by suspicious facts are suspicious as well. In our approach only default literals are considered as suspicious, and therefore possibly retracted (together with their conclusions). This explains why we do not provide a solution, for instance, to the program $\{p, \neg p, q\}$ like [41], while Sakama is able to retract one of the two facts (i.e., p and $\neg p$) leading to contradiction.

5. Abduction

In this section, we move to the final extension of the basic language and consider the case of logic programs with negation by default, explicit negation, and abduction with integrity constraints, referred to as *abductive logic programs*. In this way, we define a model-theoretic semantics for abduction in a logic programming setting (see Section 5.2). A correct proof procedure for this wider class of programs is given in Section 7.

Abduction has been recognized as a powerful mechanism for hypothetical reasoning in the presence of incomplete knowledge [27, 28, 25, 42, 48].

Though abduction has been the focus of intensive research, many questions concerning both declarative and operational semantics of abduction still arise. Given a theory T and a formula G, the goal of abduction is to find a set of atoms Δ which together with T entails G, that is $T \cup \Delta \models G$. Operationally, G is derivable in T with "conditional answer" Δ.

In this section, we focus on abductive logic program, that is extended programs where some predicate symbols are identified as *abducibles* and have no definitions in the program. An abducible predicate represents a further source of *incomplete* information, other than negation by default.

5.1. Preliminaries

As in the previous sections, we will directly refer to the *positive version* of abductive logic programs. An abductive logic program is a triple $\langle P, Ab, IC\rangle$ where

(i) P is an extended logic program (as in Section 4);
(ii) Ab is a set of *abducible atoms*, such that for each $a \in Ab$, only a and not_a may occur in P and IC, and a never occurs in the head of any clause of P;
(iii) IC is a set of integrity constraints, that is, a set of denials of the form $\leftarrow l_1, \ldots, l_n$, $n \geqslant 1$.

It is worth noting that, by condition (ii) above, we do not consider the explicit negation of an abducible atom $a^{\neg}$, but only its negation by default. This is justified by the observation that an abducible a is itself a default hypothesis, and it makes no sense assuming the explicit negation of an hypothesis, since such a negation would be anyway a default assumption. To keep the distinction between abducibles and non-abducibles clear, in the sequel we will refer to $\mathscr{L}^+$ as the set of atoms $a \notin Ab$ such that either a, or not_a, or $a^{\neg}$, or $not\_a^{\neg}$ occurs in P. Moreover, the other symbols occurring in $\langle P, Ab, IC\rangle$ will be partitioned into the following sets:

$$\mathscr{L}^{\neg} = \{a^{\neg} \mid a \in \mathscr{L}^+\} \qquad \mathscr{L}^{\mathscr{C}} = \mathscr{L}^+ \cup \mathscr{L}^{\neg} \qquad \mathscr{L}^{\mathscr{A}} = Ab$$
$$\mathscr{L}^{\mathscr{D}} = \{not\_l \mid l \in \mathscr{L}^{\mathscr{C}} \cup \mathscr{L}^{\mathscr{A}}\}.$$

To point out their intuitive meaning, atoms in $\mathscr{L}^+$, $\mathscr{L}^{\neg}$, $\mathscr{L}^{\mathscr{C}}$, $\mathscr{L}^{\mathscr{D}}$, and $\mathscr{L}^{\mathscr{A}}$ will be referred to as *positive*, *negated*, *standard*, *default* and *abducibles* literals, respectively. The whole Herbrand base $\mathscr{B}$ of an abductive logic program is then

$$\mathscr{B} = \mathscr{L}^{\mathscr{C}} \cup \mathscr{L}^{\mathscr{D}} \cup \mathscr{L}^{\mathscr{A}}.$$

Given a Herbrand interpretation $I \subseteq \mathscr{B}$ we will denote by $I^{\mathscr{A}}$ the set of abducibles in I, that is $I^{\mathscr{A}} = I \cap \mathscr{L}^{\mathscr{A}}$.

5.2. Model theory

In order to assign a semantics to an abductive logic program $\langle P, Ab, IC\rangle$ we build upon the model theory for extended logic programs defined in Section 4. We keep following an *abductive* interpretation of negation by default, which means that also in

this case the set of assumables contains the set of default atoms $\mathscr{L}^{\mathscr{D}}$. However, the assumables are now extended to contain also the set of abducible literals $\mathscr{L}^{\mathscr{A}}$, due to their inherent nature of assumables. The whole set of assumables $\mathscr{H}$ we use in this case is then

$$\mathscr{H} = \mathscr{L}^{\mathscr{D}} \cup \mathscr{L}^{\mathscr{A}}.$$

Admissible supported interpretations are still characterized through the notion of conservative extensions, as in Definition 3.3. Let us rephrase here that definition in the case of abductive logic programs.

Definition 5.1. Let $\langle P, Ab, IC\rangle$ be an abductive logic program and let $I(H), J(K) \subseteq \mathscr{B}$ be two supported interpretations. Then, $J(K)$ is a *conservative extension* of $I(H)$ if and only if $K \supseteq H$ and $\overline{K} \cap I = \emptyset$.

Remark 5.1. *Given an abductive logic program $\langle P, Ab, IC\rangle$, a conservative extension $J(K)$ of a supported interpretation $I(H)$ is such that*

(i) *for any standard atom $l \in \mathscr{L}^{\mathscr{C}}$, if not_$l \in K$ then $l \notin I$, and*

(ii) *for any abducible $a \in Ab$, if $a \in K$ (resp. not_$a \in K$) then not_$a \notin H$ (resp. $a \notin H$).*

Notice that (ii) above follows from the fact that abducibles have no definitions in P, and hence an abducible a or *not_a* belongs to a supported interpretation $I(H)$ if and only if it belongs to the set of assumables H. This allows us to keep the same notion of admissible interpretations of Definition 3.4 also in the case of abductive logic programs.

Definition 5.2. Let $\langle P, Ab, IC\rangle$ be an abductive logic program. A supported interpretation $I(H)$ is *admissible* if and only if for any conservative extension $J(K)$ of I, $\overline{H} \cap J = \emptyset$.

Notice that this definition is analogous to Definition 3.4. However, due to Remark 5.1 (ii), the admissibility of a supported interpretation $I(H)$ depends only on the fact that any conservative extension of it is consistent with the set of default hypotheses in H, that is with the set $H \cap \mathscr{L}^{\mathscr{D}}$.

As in the case of extended logic programs, we will not assign any abductive logic program a semantics, due to the fact that an abductive logic program may be inherently contradictory (notice that an extended logic program P can always be seen as the abductive logic program $\langle P, \emptyset, \emptyset\rangle$). However, the notion of non-contradictoriness of an interpretation, and consequently of an abductive logic program $\langle P, Ab, IC\rangle$, has now to take into account also the set of integrity constraints. The role of the integrity constraints IC is in fact to rule out interpretations which do not satisfy them. This is obtained by extending the notion of non-contradictoriness of an interpretation (Definition 2.3 (ii)) as follows.

Definition 5.3. Let $\langle P, Ab, IC\rangle$ be an abductive logic program and $I \subseteq \mathscr{B}$ be an interpretation. Then I is *non-contradictory* if and only if $I \models IC^{\neg}$ and $I \models IC$.

We can now naturally extend Definition 4.4 to abductive logic programs as follows.

Definition 5.4. An abductive logic program $\langle P, Ab, IC\rangle$ is *non-contradictory* if and only if the supported interpretation $I(\emptyset)$ is non-contradictory.

From now onwards we will consider only non-contradictory abductive logic programs, which are assigned a semantics by extending in a natural way the definition of admissible pre-models and complete pre-models of Section 4.

Definition 5.5. Let $\langle P, Ab, IC\rangle$ be a non-contradictory, abductive program. A supported interpretation $I(H)$ is an admissible *pre-model* of $\langle P, Ab, IC\rangle$ if and only if it is admissible and consistent.

Also in this case, we use the term *pre-model* instead of model since admissibility and consistency of a supported interpretation are not sufficient to ensure non-contradictoriness. Similarly, we can extend Definition 4.2 as follows.

Definition 5.6. Let $\langle P, Ab, IC\rangle$ be a non-contradictory, abductive program. An admissible pre-model $I(H)$ is a *complete pre-model* of P if and only if the following property holds:

$$(\forall l \in \mathscr{L}^{\mathscr{C}} \cup \mathscr{L}^{\mathscr{A}}.(not\_l \in H \Leftrightarrow (\forall J \in CE(I) : l \notin J))).$$

As in the case of extended programs, complete pre-models are promising approximations of the intended models for non-contradictory abductive programs. However, non-contradictoriness of complete pre-models with respect to the set of integrity constraints $IC \cup IC^{\neg}$ is not always guaranteed. Nevertheless, we can take here the same approach we have followed for extended logic programs, characterizing the intended models of an abductive logic program as follows.

Definition 5.7. Let $\langle P, Ab, IC\rangle$ be a non-contradictory abductive program. A supported interpretation M is an *abductive model* of $\langle P, Ab, IC\rangle$ iff it is an admissible model and is a maximal (w.r.t. set inclusion) subset of some complete pre-model N of $\langle P, Ab, IC\rangle$.

By definition, any abductive model of $\langle P, Ab, IC\rangle$ is indeed an admissible model.

The abductive semantics is defined for any non-contradictory abductive logic program, as stated by following proposition.

Proposition 5.1. *Let $\langle P, Ab, IC\rangle$ be a non-contradictory abductive logic program. Then, the set of abductive models of $\langle P, Ab, IC\rangle$ is not empty.*

Proof. Being P non-contradictory, we have that there always exists an admissible pre-model of P ($I(\emptyset)$) which is non-contradictory. From Proposition 3.1(i), there always exists a complete pre-model of P.

For each complete pre-model M of P, the set of supported interpretations which are subsets of P and which are admissible models of P is non empty (at least, $I(\emptyset)$ is the single element). □

The abductive semantics embeds the generalized stable model semantics by Kakas and Mancarella [48]. In particular, the following proposition holds.

Proposition 5.2. *Let $\langle P, Ab, IC\rangle$ be an abductive logic program. Then, any generalized stable model of $\langle P, Ab, IC\rangle$ corresponds to an abductive model.*

Proof. The proof follows from the fact that generalized stable models are *total*, and total models are indeed complete (see Proposition 3.1(ii)) and therefore they are also abductive models. □

The semantics introduced in Section 3 for negation by default and in section 4 for explicit negation can be obtained directly as instances of the abductive framework here presented.

A related approach is that by Damasio and Pereira in [30]. Our basic semantics is inspired on 3-valued stable semantics [64] while their approach is inspired on well-founded semantics [72] plus the coherence principle. The framework in [30] is an abductive one defined by a triple $\langle P, A, I\rangle$, where P is the program (possibly extended with a variable part), A the set of abducibles, and I integrity constraints. They allow general integrity constraints to be part of the program, and add further constraints to detect paraconsistency. Consistency (viz. non-contradictoriness in our terminology) is regained by revising the truth value of the revisables that lend to violation of integrity constraints. Various semantics are recovered by relaxing the coherence principle, and possibly varying both the set of abducibles and the set of constraints. In particular, they relate their framework with various abductive and 3-valued semantics for extended logic programs, such as Generalized Stable Models [48], Stable Models [44], Stationary Models [65], Well-Founded Semantics [72].

In our framework contradiction is always removed by changing the truth value of abducibles into undefined only and constraints are denials only (as in [8, 57]). In [30], instead, Damasio and Pereira allow revision by introducing or removing rules, thus they can also change the truth value of a literal from undefined to true or false.

Not surprisingly, our approach and that in [30] coincide when abducibles are default literals only, and totality constraints are added to the program (and thus the WFSX model is total). In this case, both WFSX model and our (unique) complete model coincides with the unique stable model of the program (the coherence principle being always satisfied, in this case). When also arbitrary literals, besides default literals, are

considered as abducibles, the semantics of [30] also recovers (as our semantics does) the Generalized Stable Model semantics.

6. Compositional semantics for extended logic programs

This section is devoted to discuss the issue of defining compositional semantics for extended logic programs. We will show that the model-theoretic semantics based on supported interpretations allows us to obtain compositionality results for all the various semantics that have been considered in this paper.

6.1. Compositionality

The property of compositionality plays a major 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 (see, for instance, [22, 17]).

The definition of a compositional semantics for a programming language induces interesting results on the equivalence of programs (or program parts) which are at the basis of most, if not all, operations on programs (or program parts).

Indeed each method of giving a 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. 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, for instance, 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, for instance, a more efficient version of R, obtained by applying some program transformation technique to R. Now if R' is equivalent to R in the chosen semantics then the property of compositionality ensures that the substitution of R' for R does not affect the meaning of the whole program P.

Let us now consider the case of extended logic programs. First of all, generally speaking, it is by no means clear when two programs should be considered equivalent. Indeed there is no general agreement on what the meaning of an extended logic program should be, as testified by the number of alternative proposals. Moreover, the existing semantics for extended logic programs do not address compositionality issues. In other words, it is not clear how to determine the models of a program from the models of its components.

Example 6.1. Let us consider the case of logic programs with negation as default. It is easy to show, for instance, that the stable model semantics proposed by Gelfond and Lifschitz [44] is not compositional with respect to the union of programs. Consider for instance the two programs $a \leftarrow$ and $a \leftarrow \mathit{not}\ b$ which have the same (unique)

stable model $\{a\}$. If these programs are extended with the clause $b \leftarrow$ 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.

6.2. *Establishing compositionality*

We now show how the model-theoretic semantics which has been presented in this paper can be used to define a compositional semantics for the whole class of extended logic programs.

In Sections 3–5 we have considered three extensions of logic programming: negation by default, pseudonegation and abduction. For each of these extensions, several alternative semantics have been proposed in the literature.

Each semantics can be represented by a mapping, $\mathscr{S}$, associating a meaning, $\mathscr{S}(P)$ – e.g. a set of models – to a program P. Suppose now that the equivalence relation induced by a semantics $\mathscr{S}$ is not compositional with respect to the union of programs. Namely this means that there are three programs P_1, P_2 and Q such that

$$\mathscr{S}(P_1) = \mathscr{S}(P_2) \ \wedge \ \mathscr{S}(P_1 \cup Q) \neq \mathscr{S}(P_2 \cup Q).$$

(Example 6.1 illustrates such a situation in the case of stable models.)

Our objective is then to define a compositional semantics that *preserves* such an intended meaning $\mathscr{S}$ of programs. A standard approach consists of considering a *more distinguishing* semantics $\mathscr{C}$ which preserves the semantics $\mathscr{S}$ and which is a congruence for the set of compositions considered.

In the case of extended logic programs, for a given semantics $\mathscr{S}$, this corresponds to finding a semantics $\mathscr{C}$ such that

- *$\mathscr{C}$ is a congruence for the operation of union of programs.*
  Formally, $\forall P,Q,R: \ \mathscr{C}(P) = \mathscr{C}(Q) \Rightarrow \mathscr{C}(P \cup R) = \mathscr{C}(Q \cup R)$.
- *$\mathscr{C}$ preserves $\mathscr{S}$.*
  That is, $\forall P,Q: \ \mathscr{C}(P) = \mathscr{C}(Q) \Rightarrow \mathscr{S}(P) = \mathscr{S}(Q)$.

Such a compositional semantics $\mathscr{C}$ would allow one to reason about programs and program compositions. For instance, suppose that a program P consists of two parts, Q and R say, that is $P = Q \cup R$. The compositionality result allows one to replace, for instance, R with any program R' having the same semantics $\mathscr{C}$ without affecting the meaning $\mathscr{S}(P)$ of the overall program. Indeed if $\mathscr{C}(R) = \mathscr{C}(R')$ then by definition of congruence we have that $\mathscr{C}(Q \cup R) = \mathscr{C}(Q \cup R')$. Since $\mathscr{C}$ preserves $\mathscr{S}$ we also have that $\mathscr{S}(Q \cup R) = \mathscr{S}(Q \cup R')$.

In the rest of this section, we will show that there exists a semantics $\mathscr{C}$ for extended logic programs which is compositional with respect to the union of programs, and which preserves *any* meaning $\mathscr{S}$ for extended logic programs that has been considered in this paper. More precisely, we will show that such a semantics $\mathscr{S}$ coincides with the well-known logical equivalence on the positive versions of extended logic programs. We will first recall the compositionality of logical equivalence with respect to the union of

(definite) logic programs which has been established, for instance, in [54, 24, 23]. Then we will show how the results presented in this paper imply that logical equivalence on the positive versions of programs preserves the various meanings for extended logic programs which have been previously discussed.

In order to study the property of compositionality for extended logic programs, it is necessary to first fix some suitable notions that are needed for a multi-program setting. When considering a multi-program setting, we assume that the language in which programs are written is fixed. Namely, let V be a *global vocabulary* which contains the set of predicate, function and constant symbols from which the various programs are constructed. Put another way, the Herbrand base of a collection of programs is determined by a set of predicate, function and constant symbols that includes all function and predicate symbols used in the programs. The definition of *positive version* of a program, given in Section 2, is therefore extended to take into account the global vocabulary with which programs are written. Namely, rather than denoting by $\mathscr{L}_P^+$ the set of all ground atoms a such that either a, or *not* a, or $\neg a$, or *not* $\neg a$ occurs in P we have the following more general definition. We denote by $\mathscr{L}^+$ the set of atomic formulae constructed by using predicate, function and constant symbols in V.

6.2.1. Logical equivalence

The semantics for extended logic programs that has been presented in this paper is based on the notion of supported interpretation, which has been introduced in Section 2.2. Intuitively speaking, the idea is to consider a set of assumables hypotheses $\mathscr{H}$. Then we say that an interpretation I for a program P is supported by a set of hypotheses H, where $H \subseteq \mathscr{H}$, if I is the least Herbrand model of the program $P \cup H$. The definition of supported interpretation has been then employed in Sections 3–5 to model logic programs with negation by default, pseudo negation and abduction, respectively. In each case, the notion of supported interpretation has been extended by considering different sets of assumable hypotheses, namely:

- The set of default atoms in the case of negation by default and in the case of pseudo negation ($\mathscr{H} = \mathscr{L}^{\mathscr{D}}$), and
- The set of default atoms and the set of abducible literals in the case of abduction ($\mathscr{H} = \mathscr{L}^{\mathscr{D}} \cup \mathscr{L}^{\mathscr{A}}$).

As already observed in Section 2.2, if the set of assumables coincides with the whole Herbrand base ($\mathscr{H} = \mathscr{B}$) then the notion of supported interpretation coincides with the notion of Herbrand model for definite programs. Indeed, an interpretation M is a model for a program if and only if M is supported by some interpretation.

Lemma 6.1. *Let P be a definite programs and let $M \subseteq \mathscr{B}$.*

$$M \models P \Leftrightarrow \exists H : H \subseteq \mathscr{B} \wedge M = lhm(P \cup H).$$

where lhm stands for least Herbrand model.

Proof. ($\Rightarrow$): Let $H = M$. Then:

$$
\begin{aligned}
& lhm(P \cup H) \\
= & \quad \{ \text{ definition of least Herbrand model } \} \\
& min\{I \mid I \models P \cup H\} \\
= & \quad \{ \text{ lemma 6.2 below } \} \\
& min\{I \mid I \models P \wedge I \models H\} \\
= & \quad \{ \text{ definition of } H \ \} \\
& M
\end{aligned}
$$

($\Leftarrow$): Trivial. □

The equivalence relation induced by the Herbrand model semantics is the well-known logical equivalence. That is, two definite programs are logically equivalent if and only if they have the same Herbrand models. Let $\equiv_{HM}$ denote logical equivalence, i.e.,

$$P \equiv_{HM} Q \ \Leftrightarrow \ HM(P) = HM(Q),$$

where $HM(P)$ denotes the set of Herbrand models of a definite program P. Let now $\equiv_{SI}$ denote the equivalence relation induced by the supported interpretations of a program, i.e.,

$$P \equiv_{SI} Q \ \Leftrightarrow \ SI(P) = SI(Q),$$

where $SI(P)$ denotes the set of supported interpretations for a definite program P:

$$SI(P) = \{I(H) \mid I \subseteq \mathscr{B} \ \wedge \ H \subseteq \mathscr{H} \ \wedge \ I = lhm(P \cup H)\}.$$

By virtue of Lemma 6.1, it is easy to observe that logical equivalence preserves the equivalence relation $\equiv_{SI}$ for *any* choice of the set of assumables $\mathscr{H} \subseteq \mathscr{B}$, i.e.,

$$P \equiv_{HM} Q \ \Rightarrow \ P \equiv_{SI} Q.$$

Moreover, as shown for instance in [54, 24, 23], logical equivalence is compositional with respect to the operation of union of programs. 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 lemma.

Lemma 6.2. *Let P and Q be definite programs and let $M \subseteq \mathscr{B}$.*

$$M \models P \cup Q \ \Leftrightarrow \ M \models P \wedge M \models Q.$$

Proof. ($\Rightarrow$): Immediate.

($\Leftarrow$): Suppose that $M \not\models P \cup Q$. Then, by definition of Herbrand model, $\exists A \leftarrow \boldsymbol{B} \in ground(P \cup Q)$ such that $\boldsymbol{B} \subseteq M \wedge A \notin M$. Suppose that such a clause belongs to $ground(P)$ (the other case is analogous). Then by definition of Herbrand model M is not a model for P. Contradiction. □

6.2.2. Relations with other semantics

We now conclude our discussion by showing how logical equivalence on (the positive version of) programs actually preserves the equivalence relations induced by all the semantics for extended logic programs which have been considered in this paper.

In Sections 3–5 we have considered three main extensions of logic programming: Negation by default, pseudo negation and abduction. In each case we have proposed a model-theoretic semantics whose definition consists of three steps:

(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 those models which characterize the intended meaning of a program.

We have also shown the existing correspondence between these models and the models proposed by other authors. For instance, in Section 3.2, we have shown the correspondence between our complete models and:

- the complete scenario of Dung [35],
- the stationary expansions of Przymusinski [65],
- the stable models of Gelfond and Lifschitz [44], and
- the well-founded semantics of Van Gelder, Ross, and Schlipf [72].

These correspondences can be equivalently described in the following way. For each semantics $\mathscr{S}$ considered, there exists a suitable *projection* function $\psi_{\mathscr{S}}$ which, given the set of supported interpretations $SI(P)$ of (the positive version of) a program P, yields the models $\mathscr{S}(P)$ of the corresponding semantics $\mathscr{S}$:

$$\psi_{\mathscr{S}} : SI(P) \rightarrow \mathscr{S}(P).$$

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

$$P \equiv_{SI} Q \;\Rightarrow\; P \equiv_{\mathscr{S}} Q.$$

By combining the latter observation with the observation that logical equivalence preserves the equivalence relation induced by the supported interpretations of a program, we have the following result.

Proposition 6.3. *Let P and Q be two extended logic programs. If the positive versions of P and Q are logically equivalent, then the two programs have the same meaning $\mathscr{S}$ for each $\mathscr{S}$ considered in the paper.*

Proof. Trivial. □

Consider for instance two logic programs with negation by default. If (the positive versions of) the two programs are logically equivalent then the programs have the same D-complete scenaria [35], the same stationary expansions [65], the same stable models [44], and so on.

The compositionality of logical equivalence suggests its use for reasoning about extended logic programs. For instance, suppose that a program P consists of two parts, Q and R say, that is $P = Q \cup R$. The we can replace R with any program R' such that the positive versions of R and R' are logically equivalent. Indeed, thanks to the compositionality of logical equivalence, we have that the resulting programs have the same meaning, that is $\mathscr{S}(Q \cup R) = \mathscr{S}(Q \cup R')$ for *any* meaning $\mathscr{S}$ which has been considered in this paper.

The application of these results to analyze program transformation techniques, for extended logic programs is scope for future work (see also [11]).

7. Proof procedure

In this section, we present a top-down proof procedure which is correct for the semantics discussed in Sections 3–5. In the following we rely upon the terminology introduced in Section 5 since the abductive semantics defined in that section includes those defined for normal 3 and extended logic programs 4.

The method here described is not effective, and has to be rather considered an abstract interpreter. Therefore, we do not address the problems of loops and termination which are fundamental, instead, having in mind a real implementation.

The proof procedure here presented is grounded on that defined by Kakas and Mancarella [49]. The starting point is to consider an abductive logic program as a positive program (as shown in Section 2) provided that integrity constraints of the kind $\{\leftarrow l, not\_l | \forall l \in \mathscr{L}^{\mathscr{C}}\}$ and $\{\leftarrow a, \neg a | \forall a \in \mathscr{L}^{+}\}$ are added to IC.

The procedure in [49] extended Eshghi and Kowalski's procedure [42] in order to manipulate arbitrarily abducibles. It deals with ground abductive logic programs, and manipulates a class of integrity constraints which are denials containing at least one abducible or default atom in each integrity constraint. Therefore, we assume the same hypotheses.

We accommodate this procedure for our framework by

- Lifting it to the 3-valued case. In particular, we do not consider integrity constraints of the kind $a \vee not\_a$, imposed instead in [49]. In this way we remove the totality requirement. [6]
- Dealing with explicit negation, not considered in [49]. Each negated literal $\neg a$ is viewed as a new positive atom, and the integrity constraint $\leftarrow a, \neg a$ added to the program.

[6] This kind of constraint is enforced in [49], but not used in the procedure. In fact, non-contradictoriness checking only checks integrity constraints which are denials.

Notice that this latter integrity constraint does not satisfy the requirement of having at least one abducible or default atom. However, through the application of unfolding on standard atoms we can possibly reduce to this case[7]

Example 7.1. Let us consider the following program:

$a \leftarrow not\_b$

$a^{\neg} \leftarrow not\_b$

The added integrity constraint $\leftarrow a, \neg a$ is transformed, through unfolding in the following, equivalent one: $\leftarrow not\_b$.

7.1. The basic algorithm

In the following, we recall the main steps of the procedure, which is in practice that defined in [49].

Abductive derivation: An abductive derivation from $(G_1\ \Delta_1)$ to $(G_n\ \Delta_n)$ in $\langle P, Ab, IC\rangle$ via a selection rule R is a sequence

$$(G_1\ \Delta_1),(G_2\ \Delta_2),\ldots,(G_n\ \Delta_n)$$

such that each G_i has the form $\leftarrow L_1,\ldots,L_k$, $R(G_i) = L_j$ and $(G_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:

(1) If L_j is not abducible or default, then $G_{i+1} = C$ and $\Delta_{i+1} = \Delta_i$ where C is the resolvent of some clause in P with G_i on the selected literal L_j;

(2) If L_j is abducible or default and $L_j \in \Delta_i$ then $G_{i+1} = \leftarrow L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k$ and $\Delta_{i+1} = \Delta_i$;

(3) If L_j is abducible or default, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$ and there exists a *consistency derivation* from $(\{L_j\}\ \Delta_i \cup \{L_j\})$ to $(\{\}\ \Delta')$ then $G_{i+1} = \leftarrow L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k$ and $\Delta_{i+1} = \Delta'$.

Steps (1) and (2) are SLD-resolution steps with the rules of P and abductive or default hypotheses, respectively. In step (3) a new abductive or default hypotheses is required and it is added to the current set of hypotheses provided it is non-contradictory.

Consistency derivation: A consistency derivation for an abducible or default literal α from $(\alpha\ \Delta_1)$ to $(F_n\ \Delta_n)$ in $\langle P, Ab, IC\rangle$ is a sequence

$$(\alpha\ \Delta_1),(F_1\ \Delta_1),(F_2\ \Delta_2),\ldots,(F_n\ \Delta_n)$$

where (i) F_1 is the union of all goals of the form $\leftarrow L_1,\ldots,L_n$ obtained by resolving the abducible or default α with the denials in IC with no such goal been empty, $\leftarrow$;

(ii) for each $i > 1$, F_i has the form $\{\leftarrow L_1,\ldots,L_k\} \cup F_i'$ and for some $j = 1,\ldots,k$ $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:

[7] Another solution would be that proposed in [68], where forward evaluation of rules is adopted in order to check any form of integrity constraints. A proof procedure for the abductive semantics dealing with a broader class of integrity constraints but based on forward evaluation of rules is reported in [70].

(C1) If L_j is not abducible or default, then $F_{i+1} = C' \cup F'_i$ where C' is the set of all resolvents of clauses in P with $\leftarrow L_1,\ldots,L_k$ on the literal L_j and $\leftarrow \notin C'$, and $\Delta_{i+1} = \Delta_i$;

(C2) If L_j is abducible or default, $L_j \in \Delta_i$ and $k > 1$, then

$$F_{i+1} = \{\leftarrow L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k\} \cup F'_i \text{ and } \Delta_{i+1} = \Delta_i;$$

(C3) If L_j is abducible or default, $\overline{L_j} \in \Delta_i$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta_i$;

(C4) If L_j is abducible or default, $L_j \notin \Delta_i$ and $\overline{L_j} \notin \Delta_i$, and there exists an *abductive derivation* from $(\leftarrow \overline{L_j}\ \Delta_i)$ to $(\leftarrow \Delta')$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta'$.

In case (C1) the current branch splits into as many branches as the number of resolvents of $\leftarrow L_1,\ldots,L_k$ with the clauses in P on L_j. If the empty clause is one of such resolvents the whole consistency check fails. In case (C2) the goal under consideration is made simpler if literal L_j belongs to the current set of hypotheses Δ_i. In case (C3) the current branch is already non-contradictory under the assumptions in Δ_i, and this branch is dropped from the consistency checking. In case (C4) the current branch of the consistency search space can be dropped provided $\leftarrow \overline{L_j}$ is abductively provable.

Given a query L (atomic, for the sake of simplicity), the procedure succeeds, and returns the set of abducibles Δ if there exists an abductive derivation from $(\leftarrow L\{\})$ to $(\leftarrow \Delta)$.

Example 7.2. For the program of Example 7.1 the algorithm would produce no abductive explanation for a since there not exists any consistency derivation for the default literal *not_b*. If no unfolding is performed on integrity constraints, the algorithm would produce, instead, $\{not\_b\}$ as an abductive explanation for a, and this explanation is clearly not right.

In [49], Kakas and Mancarella have stated the soundness of this procedure with respect to generalized stable model semantics when the program P is call-consistent.

Theorem 7.1 (Kakas and Mancarella [49]). *Let $\langle P, Ab, IC\rangle$ be an abductive logic program where P is call-consistent and a literal L an observation. If $(\leftarrow L\{\})$ has an abductive derivation to $(\leftarrow \Delta)$, then the subset $\Delta' \subseteq \Delta$ such that $\Delta' = \Delta \cap \mathscr{L}^{\mathscr{A}}$ is an abductive explanation of L.*

A literal L has an abductive explanation $\Delta \subseteq \mathscr{L}^{\mathscr{A}}$ if and only if there exists a (generalized) stable model M of $P \cup \Delta$ such that $L \in M$ and $M \models IC$ [49].

However, the following example shows the unsoundness of this procedure even for call-consistent programs.

Example 7.3. Let us consider the following (call-consistent) abductive program, where P

$a \leftarrow$

$b \leftarrow c$

$Ab = \{c\}$ and $IC = \{\leftarrow not\} \cup \{\leftarrow c\}$. There exists an abductive derivation from $(\leftarrow a\{\,\})$ to $(\leftarrow \{\,\})$, but no generalized stable models exists. In fact, the only complete total interpretations, $\{a, not\_b, not\_c\}$ and $\{a, b, c\}$, do not satisfy the integrity constraints. Our semantics, instead, gives meaning to this program in terms of the abductive model $\{a\}$.

The correctness of the proof procedure with respect to our abductive semantics is established by Theorem 7.3 and Theorem 7.4, stating, respectively, the soundness and completeness results.

In order to prove these theorems, we use the following lemma which is an extension of Lemma A4 in [35].

Lemma 7.2. *Let us consider a* non-contradictory *abductive logic program* $\langle P, Ab, IC\rangle$, *and let* $L \in \mathscr{L}^{\mathscr{C}}$ *be a standard atom.*

(i) *Assume there exists an abductive derivation from* $(\leftarrow L\{\,\})$ *to* $(\leftarrow \Delta)$. *Then, the supported interpretation* $I(\Delta)$ *of* $\langle P, Ab, IC\rangle$ *is non-contradictory.*

(ii) *Assume there exists a non-contradictory supported interpretation of* $\langle P, Ab, IC\rangle$, $I(H)$, *such that* $L \in I(H)$. *Suppose that every selection of rules in the proof procedure for L terminates with either success or failure. Then, there exists an abductive derivation from* $(\leftarrow L\ \{\})$ *to* $(\leftarrow \Delta)$ *such that* $\Delta \subseteq H$.

Theorem 7.3 (Soundness). *Let us consider a* non-contradictory *abductive logic program. Let L be a standard atom. If there exists an abductive derivation from* $(\leftarrow L\{\,\})$ *to* $(\leftarrow \Delta)$ *then there exists an abductive model M such that* $L \in M$ *and* $\Delta \subseteq M^{\mathscr{A}} \cup M^{\mathscr{D}}$.

Proof (*sketch*). From Lemma 7.2(i), $I(\Delta)$ is non-contradictory. From Proposition A in [35] and Theorem 3.5, we also have that $I(\Delta)$ is admissible. Therefore, from Definition 5.7, there exists an abductive model M such that $I(\Delta) \subseteq M$. □

Theorem 7.4 (Weak completeness). *Let us consider an abductive logic program. Let L be a standard atom. Suppose that every selection of rules in the proof procedure for L terminates with either success or failure. If there exists an abductive model M such that* $L \in M$, *then there exists a selection of rules such that the procedure succeeds for L returning* Δ, *where* $\Delta \subseteq M^{\mathscr{A}} \cup M^{\mathscr{D}}$.

Proof (*sketch*). Straightforward from Lemma 7.2 (ii) since any abductive model is non-contradictory by definition. □

Therefore, the procedure here defined is sound with respect to the abductive semantics, and it is complete when no predicate in the program depends on itself, i.e., the program contains no loop.

From Theorem 7.4, we obtain the following corollary for finite failure.

Corollary 7.5. *Let us consider an abductive logic program. Let L be a standard atom. If the procedure finitely fails for L then no abductive model M exists such that $L \in M$.*

From this corollary directly follows that we can use finite failure to check if a literal L is *unknown* in all abductive models, since the finite failure of the procedure for both L and $\overline{L}$ means that every abductive model does not contain either L or $\overline{L}$.

7.2. Related work

Eshghi and Kowalski [42] firstly introduced a top-down proof procedure (further refined in [35]) to compute negation as failure through abduction which is sound with respect to admissible scenaria [35]. When considering normal logic programs, the procedure here presented becomes, in practice, the abductive proof procedure by Eshghi and Kowalski. Therefore, Theorem 7.4 states a weak completeness result, in the case of finite failure, for admissible scenarios.

In the following, we mainly focus on proof procedures for abductive or extended logic programs. Thus, we do not consider procedures defined for normal logic programs (see [14] for a survey).

In Proposition 4.7, we have shown that, for extended logic programs, our semantics embeds the 3-valued stable semantics which, in turns, subsumes the answer set semantics. Therefore, our proof procedure is weakly complete (under the conditions of Theorem 7.4) with respect to the answer set semantics for extended logic programs, but of course not sound. In fact, the totality requirement of answer set semantics is not enforced in our procedure.

To provide a query evaluation method for abduction, Kakas and Mancarella [49] extended Eshghi and Kowalski's procedure to manipulate arbitrarily abducibles. The procedure in [49] is top-down and maintains the backward evaluation of integrity constraints as [42]. Satoh and Iwayama [68] pointed out that this proof procedure is unsound with respect to generalized stable model semantics, and presented a new query-based procedure which can be regarded as an improvement of Kakas and Mancarella's one, and is correct with respect to Generalized Stable Model semantics. This is obtained at the expense of a more complex integrity checking involving forward evaluation of rules for handling (general) integrity constraints and implicit deletion of rules [66].

As pointed out in Section 7, the procedure by Kakas and Mancarella [49] is correct with respect to our abductive semantics. Neither checking for implicit deletion nor forward evaluation of rules is therefore needed to equip our semantics with a correct, query-based proof procedure.

A proof procedure for our abductive framework could be also obtained from that defined by Satoh and Iwayama [68]. The extension would concern the check for implicit deletion of rules. There exists, in fact, an intimate bond between implicit deletion checking and the totality requirement of (2-valued) stable semantics. The check for implicit deletion in [68] actually verifies that the model under consideration gives an

absolute truth value (true or false) to each atom. Since abductive models are not required to be total, removing the check for implicit deletion from the procedure in [68] ensures a correct behavior with respect to our abductive semantics (see [70] for details).

In [71], Teusink presents a proof procedure for extended logic programs correct with respect to a proof-theoretic semantics. As pointed out by the author, the implemented semantics does not always give the intuitive expected results. This semantics does not guarantee the global non-contradictoriness (which is, instead, guaranteed in our proof procedure) since considered too complex to be tested, but only some form of local non-contradictoriness.

In [31], Denecker and De Schreye introduce a proof procedure for normal abductive logic programs by extending the SLDNF resolution to the case of abduction. The resulting proof procedure (SLDNFA) is correct with respect to the completion semantics. A crucial property of this abductive procedure is the treatment of *non-ground* abductive goals not considered, instead, in our procedure limited to the ground case. In [31], the authors do not consider general integrity constraints, but only constraints of the kind $\leftarrow a, not\_a$. To overcome this limitation, in a later work [32], they consider the treatment of general integrity constraints but in a quite inefficient way. In practice, they check all the integrity constraints at the end of the proof for a query, i.e., only when the overall set of abductive hypotheses supporting the query has been computed. As pointed out by Poole [63], it is better to check consistency as soon as the hypotheses are generated. In our proof procedure, we check consistency in an incremental way and this is easily done since we have no variable in hypotheses.

The problem of incremental consistency checking for the generated hypotheses (possibly containing variables) is considered in [63] where Poole develops a default and abductive reasoning system called *Theorist* based on a first-order language and an abductive resolution-based proof-procedure for it. In *Theorist*, consistency is immediately checked for those hypotheses with no variables and delayed until the end for hypotheses containing variables at the time they are generated. An alternative solution could be to allow the consistency check to return constraints on the value of variables. In [2] the authors define *SLX*, a top-down derivation procedure for WFSX semantics [56] based on the semantic AND-tree characterization of WFSX presented in [3]. The derivation procedure has been generalized in [4] to the paraconsistent version of *WFSX* taking into account general forms of integrity constraints for removing contradiction. Basically, the derivation procedure relies on two kinds of derivations: T-derivations, proving truth, and TU-derivations proving non-falsity. Shifting from one to the other is required for proving a default literal *not L*: the T-derivation of *not L* succeeds iff the TU-derivation of L fails; the TU-derivation of *not L* succeeds iff the T-derivation of L fails. Moreover, T-derivation of *not L* succeeds if T-derivation of $\neg L$ succeeds (thus taking into account the *coherence principle*), and TU-derivation of L fails if T-derivation of $\neg L$ succeeds.

The procedure is amenable to a simple implementation. To guarantee termination, suitable rules are introduced that prune the search space and eliminate both cyclic

positive recursion and cyclic negative recursion. This procedure has several similarities with the procedure procedure by Eshghi and Kowalski [42] and therefore ours, even if these procedures do not treat positive recursion and non-cyclic negative recursion. In fact, abductive and consistency derivation resemble T-derivation and TU-derivation, respectively. However, two big differences can be pointed out: first, our procedure does not take into account the coherence principle and, second, does not compute the WFSX, but the abductive models.

8. Conclusions

We presented a semantics which uniformly integrates abduction and two kinds of negation in a logic programming setting.

A model-theoretic semantics is defined in terms of Herbrand models, and equipped with an interpretation in terms of argumentation. A model-theoretic semantics based on Herbrand models is a widely accepted concept in the logic programming community, which also accounts for interesting properties such as compositionality and program equivalence.

A distinguishing property of our semantic framework is that it is defined for any non-contradictory program, and is strictly related to well-known semantics for the considered extensions, namely abduction, negation by default and pseudo negation, when taken separately. The case of logic programming extended with negation by default is recovered by considering an abductive logic program $\langle P, Ab, IC\rangle$ where P is a normal logic program viewed as a positive one, $Ab = \{\ \}$ and $IC = \{\}$. In this case, any interpretation is non-contradictory and the abductive semantics here defined coincides with the 3-valued stable model semantics. The resulting semantic framework is that presented in [20]. The set of abductive models correspond to the set of 3-valued stable models [64] (or stationary expansions [65], or complete scenaria [35], being they all equivalent as proved in [20]). All the other results presented in [20] hold. In particular, total complete supported models correspond to Gelfond-Lifschitz's stable models [44].

Extended logic programs can be dealt with in the abductive framework by viewing negated literals $\neg a$ as new positive atoms $a^{\neg}$, and enforcing a suitable *non-contradictoriness* condition to capture the meaning of pseudo negation. In particular, the case of extended logic programming can be recovered by considering an abductive logic program $\langle P, Ab, IC\rangle$ where P is an extended logic program viewed as a positive one, $Ab = \{\ \}$ and $IC = \{\leftarrow a, a^{\neg} | \forall a \in \mathscr{L}^{+}\}$.

We prove that for an extended logic program P, the set of its complete models captures 3-valued stable semantics [64], and that the contradiction-removal semantics coincides with the complete scenarios semantics for extended logic programs of [41]. Moreover, the total complete models capture the answer set semantics [45].

Therefore the abductive semantics given here can be considered a common ground where two kinds of negation and abduction – and their associated semantics – can be formally compared and uniformly integrated.

We equipped our abductive framework with a query-based, correct proof-procedure in order to provide a schema for an abstract interpreter for the language. This procedure is, in practice, that defined by Kakas and Mancarella, and coincides with that defined by Eshghi and Kowalski when normal logic programs are considered. Of course, in order to transform the procedure to a real algorithmic issues such as search and loop checking should be treated. This would be the subject for future work.

Acknowledgements

We thank the anonymous referees for their useful and pertinent comments on the first version of this paper. This work has been partially supported by MURST 40% Project.

References

[1] J.J. Alferes, personal communication, 1993.

[2] J.J. Alferes, C.V. Damasio and L.M. Pereira, Top-down query evaluation for well-founded semantics with explicit negation, In: A. Cohn, ed., *Proc. ECAI94* (Morgan Kaufmann, Los Altos, CA, 1994) 140–144.

[3] J.J. Alferes, C.V. Damasio and L.M. Pereira, SLX – A top-down derivation procedure for programs with explicit negation, in: M. Bruynooghe, ed., *Proc. Internat. Symp. on Logic Programming ILPS94* (MIT Press, Cambridge, MA, 1994).

[4] J.J. Alferes, C.V. Damasio and L.M. Pereira, A logic programming system for non-monotonic reasoning, *J. Automat. Reasoning* **14** (1995) 93–147.

[5] J.J. Alferes, P.M. Dung and L.M. Pereira, Scenario semantics of extended logic programs. in: L.M. Pereira and A. Nerode, eds., *Proc. 2nd Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, MA, 1993) 334–348.

[6] J.J. Alferes and L.M. Pereira, On logic program semantics with two kinds of negation, in: K. Apt, ed., *Proc. Internat. Joint Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 574–588.

[7] J.J. Alferes and L.M. Pereira, Optative reasoning with scenario semantics, in: D.S. Warren, ed., *Proc. 10th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1993) 601–615.

[8] J.J. Alferes and L.M. Pereira, Contradiction: when avoidance equals removal – Part I, in: R. Dychoff, ed., *Proc. 4th Internat. Workshop on Extensions of Logic Programming*, Lecture Notes in Computer Science, Vol. 798 (Springer, Berlin, 1994) 11–23.

[9] J.J. Alferes and L.M. Pereira, Contradiction: when avoidance equals removal – Part II, in: R. Dychoff, ed., *Proc. 4th Internat. Workshop on Extensions of Logic Programming*, Lecture Notes in Computer Science, Vol. 798 (Springer, Berlin, 1994) 268–281.

[10] J.J. Alferes, T.C. Przymusinski and L.M. Pereira, "Classical" negation in non-monotonic reasoning and logic programming, in: *Proc. Int. Workshop on Artificial Intelligence and Mathematics*. Fort Laudersdale, Florida, January 1996.

[11] C. Aravindan and P.M. Dung, On the Correctness of unfold/fold transformation of normal and extended logic programs, *J. Logic Programming* **24** (1995) 201–217.

[12] K.R. Apt, Logic programming, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. B (Elsevier, Amsterdam, 1990) 493–574.

[13] K.R. Apt and M. Bezem, Acyclic programs, *New Generation Computing* **9** (Springer, Berlin, 1991) 335–363.

[14] K.R. Apt and R.N. Bol, Logic programming and negation: a survey. *J. Logic Programming* **19/20** (1994) 9–71.

[15] C. Baral and M. Gelfond, Logic programming and knowledge representation, *J. Logic Programming* **19/20** (1994) 73–148.

[16] A. Bondarenko, F. Toni and R.A. Kowalski, An Assumption-based framework for non-monotonic reasoning, in: L.M. Pereira and A. Nerode, eds., *Proc. 2nd Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, 1993) 171–189.

[17] A. Bossi, M. Gabbrielli, G. Levi and M.C. Meo, Contributions to the semantics of open logic programs, in: *Proc. Internat. Conf. on Fifth Generation Computer Systems FGCS92* (ICOT, 1992) 570–580.

[18] G. Brewka. An abductive framework for generalized logic programs, in: L.M. Pereira and A. Nerode, eds., *Proc. 2nd Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, MA, 1993) 349–364.

[19] G. Brewka and K. Konolige, An abductive framework for general logic programs and other nonmonotonic systems, *Proc. Internat. Joint Conf. on Artificial Intelligence IJCAI93* (AAAI, 1993) 9–15.

[20] A. Brogi, E. Lamma, P. Mancarella and P. Mello, Normal logic programs as open positive programs, in: K. Apt, ed., *Proc. Internat. Joint Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 783–797.

[21] A. Brogi, E. Lamma, P. Mancarella and P. Mello, An abductive framework for extended logic programming, in: *Proc. 3rd Internat. Workshop on Logic Programming and Non-Monotonic Reasoning*, Lecture Notes in Computer Science, LNAI Series, Vol. 928 (Springer, Berlin, 1995) 330–343.

[22] A. Brogi, E. Lamma and P. Mello, Compositional model-theoretic semantics for logic programs, in: *New Generation Comput.* **11** (Springer, Berlin, 1992) 1–21.

[23] A. Brogi and F. Turini, Fully abstract compositional semantics for an algebra of logic programs, *Theoret. Comput. Sci.* **149** (1995) 201–229.

[24] M. Bugliesi, E. Lamma and P. Mello, Modularity in logic programming, *J. Logic Programming* **19–20** (Amsterdam, 1994) 443–502.

[25] W. Chen and D.H. Warren, Abductive logic programming, Technical Report, Dept. of Computer Science, State University of New York, 1989.

[26] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Databases* (Plenum Press, New York, 1978).

[27] L. Console, D. Theseider Duprè and P. Torasso, Abductive reasoning through direct deduction from completed domains models, in: *Methodologies for Intelligent Systems*, Vol. 4 (North Holland, Amsterdam, 1989) 175.

[28] P.T. Cox, and T. Pietrzykowski, Causes for events: their computation and applications, in: *Proc. CADE-86* (1986) 608.

[29] C.V. Damasio Paraconsistent Logic Programming with Constraints. Ph. D. Thesis. Universidade Nova de Lisboa, 1996.

[30] C.V. Damasio and L.M. Pereira, Abduction over 3-valued extended logic programs, in: *Proc. 3rd Internat. Workshop on Logic Programming and Non Monotonic Reasoning*, Lecture Notes in Computer Science, LNAI Series, Vol. 928 (Springer, Berlin, 1995) 29–42.

[31] M. Denecker and D. De Schreye, SLDNFA: an Abductive procedure for normal abductive programs. in: K. Apt, ed., *Proc. Int. Joint Conf. and Symp. on Logic Programming*, 686–700. The MIT Press, (1992).

[32] M Denecker and D. De Schreye, Representing incomplete knowledge in abductive logic programming, in: *Proc. Internat. Logic Programming Symp. ILPS93* (MIT Press, Cambridge, MA, 1993) 147–163.

[33] J. Dix, Semantics of logic programs: their intuitions and formal properties, in: A. Fuhrmann and H. Roth, eds., *Logic, Action and Information* (de Gruyter, Berlin, 1994).

[34] J. Doyle, A truth maintenance system, *Artificial Intelligence* **12** (1979) 231–272.

[35] P.M. Dung, Negation as hypothesis: an abductive foundation for logic programming, in: K. Furukawa, ed., *Proc. 8th Internat. Conf. on Logic Programming ICLP91* (MIT Press, Cambridge, MA, 1991) 3–17.

[36] P.M. Dung, On the Relations between stable and well-founded semantics of logic programs, *Theoret. Comput. Sci.* **105** (1992) 7–25.

[37] P.M. Dung, On the acceptability of arguments and its fundamental role in non-monotonic reasoning, in: *Proc. Internat. Joint Conf. on Artificial Intelligence IJCAI93* (AAAI, 1993) 852–857.

[38] P.M. Dung, An argumentation semantics for logic programming with explicit negation, in: D.S. Warren, ed., *Proc. 10th Internat. Conf. on Logic Programming ICLP93* (MIT Press, Cambridge, MA, 1993) 616–630.

[39] P.M. Dung, On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and N-persons games, *Artificial Intelligence* **77** (1995) 321–352.

[40] P.M. Dung, An argumentation-theoretic foundation for logic programming, *Logic Programming* **22** (1995) 151–171.

[41] P.M. Dung and P. Ruamviboonsuk, Well-founded reasoning with classical negation, in: *Proc. 1st Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, MA, 1991) 120–132.

[42] K. Eshghi and R.A. Kowalski, Abduction compared with negation by failure, in: G. Levi and M. Martelli, eds., *Proc. 6th Internat. Conf. on Logic Programming ICLP89* (MIT Press, Cambridge, MA, 1989) 234–254.

[43] C. Evans, Negation-as-failure as an approach to the Hanks and McDermott Problem, in: *Proc. 2nd Internat. Symp. on Artificial Intelligence* (Monterrey, Mexico, 1989).

[44] M. Gelfond and V. Lifschitz, The stable model semantics for logic programming, in: R.A. Kowalski and K.A. Bowen, eds., *Proc. 5th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1988) 1070–1080.

[45] M. Gelfond and V. Lifschitz, Logic programs with classical negation, in: D.H.D. Warren and P. Szeredi, eds., *Proc. 7th Internat. Conf. on Logic Programming ICLP90* (MIT Press, Cambridge, MA, 1990) 579–597.

[46] M. Gelfond and V. Lifschitz, Representing actions in extended logic programming, in: K. Apt, ed., *Proc. Internat. Joint Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 559–573.

[47] A.C. Kakas, R.A. Kowalski and F. Toni, Abductive logic programming, *J. Logic and Comput.* **2** (1993) 719–770.

[48] A.C. Kakas and P. Mancarella, Generalized stable models: a semantics for abduction, in: *Proc. 9th European Conf. on Artificial Intelligence ECAI90* (Pitman, London, 1990) 385–391.

[49] A.C. Kakas and P. Mancarella, On the relation between truth maintenance and abduction, in: *Proc. PRICAI90* (1990).

[50] A.C. Kakas and P. Mancarella, Preferred extensions are partial stable models, *J. Logic Programming* **14** (North-Holland, Amsterdam, 1992) 341–348.

[51] A.C. Kakas and P. Mancarella, Stable theories for logic programs, in: *Proc. Internat. Symp. of Logic Programming* (MIT Press, Cambridge, MA, 1991) 85–100.

[52] R.A. Kowalski, Problems and promises of computational logic, in: J.W. Lloyd, ed., *Computational Logic – Symp. Proc.* (Springer, Berlin, 1990) 1–36.

[53] K. Inoue, Extended logic programs with default assumptions, in: K. Furukawa, ed., *Proc. 8th Internat. Conf. on Logic Programming ICLP91* (MIT Press, Cambridge, MA, 1991) 490–504.

[54] M. Maher, Equivalences of logic programs, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, CA, 1988) 627–658.

[55] P. Mancarella and D. Pedreschi, An algebra of logic programs, in: R.A. Kowalski and K.A. Bowen, eds., *Proc. Fifth Internat. Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1988) 1006–1023.

[56] L.M. Pereira and J.J. Alferes, Well founded semantics for logic programs with explicit negation, in: *Proc. ECAI92* (Wiley, New York, 1992) 102–106.

[57] L.M. Pereira, J.J. Alferes and J.N. Aparicio, Contradiction removal within well-founded semantics, in: *Proc. 1st Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, MA, 1991) 105–119.

[58] L.M. Pereira, J.J. Alferes and J.N. Aparicio, Contradiction removal semantics with explicit negation, in: M. Masuch, J.N. Aparicio and J.J. Alferes, eds., *Knowledge Representation and Reasoning under Uncertainty*, Lecture Notes in Artificial Intelligence, Vol. 808, (Springer, Berlin, 1994) 91–106.

[59] L.M. Pereira, J.N. Aparicio and J.J. Alferes, Derivation procedures for extended stable models, in: *Proc. IJCAI91* (Morgan Kaufman, Los Altos, CA, 1991) 863–868.

[60] L.M. Pereira, J.N. Aparicio and J.J. Alferes, Non-monotonic reasoning with logic programming, *J. Logic Programming* **17** (1993) 227–263.

[61] L.M. Pereira, C. Damasio and J.J. Alferes, Debugging by diagnosing assumptions, in: P.A. Fritzson, ed., *Proc. Automatic Algorithmic Debugging AADEBUG93*, Lecture Notes in Computer Science, Vol. 749 (Springer, Berlin, 1993) 58–74.

[62] S.G. Pimentel and W.L. Rodi, Belief revision and paraconsistency in a logic programming framework, in: *Proc. 1st Internat. Workshop on Logic Programming and Non-Monotonic Reasoning* (MIT Press, Cambridge, MA, 1991) 228–242.

[63] D. Poole, Compiling a default reasoning system into prolog, in: *New Generation Comput.* **9** (Springer, Berlin, 1991) 3–38.

[64] T.C. Przymusinski, Extended stable semantics for normal and disjunctive programs, in: D.H.D. Warren and P. Szeredi, eds., *Proc. 7th Internat. Conf. on Logic Programming* (MIT Press, Cambridge, MA, 1990) 459–477.

[65] T.C. Przymusinski, Semantics of disjunctive logic programs and deductive databases, in: *Proc. DOOD'91* (1991).

[66] F. Sadri and R.A Kowalski, A theorem-proving approach to database integrity, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan-Kaufmann, Los Altos, CA, 1988) 313–362.

[67] C. Sakama, Extended well-founded semantics for paraconsistent logic programs, in: *Proc. Internat. Conference on Fifth Generation Computer Systems FGCS92* (ICOT, 1992) 592–599.

[68] K. Satoh and N. Iwayama, A query evaluation method for abductive logic programming in: K. Apt, ed., *Proc. Internat. Joint Conf. and Symp. on Logic Programming* (MIT Press, Cambridge, MA, 1992) 671–685.

[69] D. Saccà and C. Zaniolo, Stable models and non-determinism for logic programs with negation, *Proc. ACM SIGMOD-SIGACT Symp. on Principles of Database Systems* (1990) 205–217

[70] A. Sørli, Algorithms for treating abductive logic programming based on stable model semantics, ERASMUS Thesis, University of Bologna (1993).

[71] F. Teusink, A proof procedure for extended logic programs, in: *Proc. Internat. Logic Programming Symp. ILPS93* (MIT Press, Cambridge, MA, 1993) 235–249.

[72] A. Van Gelder, K.A. Ross and J.S. Schlipf, Unfounded sets and the well-founded semantics for general logic programs, in: *Proc. ACM SIGMOD-SIGACT, Symp. on Principles of Database Systems* (1988).

[73] J.H. You and L.Y Yuan, On the equivalence of semantics for normal logic programs, *J. Logic Programming* **22** (1995) 211–222.