In this paper, we propose a semantics for logic programs with negation as failure, the Finite Failure Stable Model semantics (FF-SM semantics), which is a three-valued extension of Gelfond and Lifschitz' Stable Model semantics. FF-SM semantics is defined in the style of Gelfond and Lifschitz Stable Model semantics, but it builds on an underlying Kripke/Kleene semantics, in which loops causing nonterminating computations are modeled by means of the truth-value undefined. It is different from the eXtended Stable Model (XSM) semantics defined by Przymusinski, since it does not capture infinite failure. We also introduce an abductive proof procedure which is an abductive extension of SLDNF-resolution based on the ideas underlying Eshghi and Kowalski's abductive procedure. We prove that our procedure is sound and complete with respect to FF-SM semantics. We compare the FF-SM semantics with the XSM semantics, and provide a reconstruction for it within the bilattice-based framework proposed by Fitting. In the paper, we deal with the propositional case.

1. INTRODUCTION

Several efforts have been recently devoted to extending logic programming to perform abductive reasoning [15]. Abduction is a form of reasoning which allows to compute explanations for observations. Moreover, it is a form of nonmonotonic reasoning, since explanations which are consistent in a given context may become inconsistent when additional information is added. In fact, it is well known that abduction provides an alternative formalization of default reasoning [23].

Address correspondence to Dipartimento di Informatica, Università di Torino, Corso Svizzera 185 10149 Torino, Italy.
Received May 1994; accepted April 1995.

THE JOURNAL OF LOGIC PROGRAMMING
© Elsevier Science Inc., 1996
655 Avenue of the Americas, New York, NY 10010
0743-1066/96/$15.00
SSDI 0743-1066(95)00065-R
In the context of logic programming, nonmonotonic reasoning is usually performed by making use of negation as failure (NAF). Eshghi and Kowalski have first recognized the strong relationship between abduction and negation as failure. In [8], they have proposed an abductive semantics for negation as failure in logic programming, which is equivalent to the Stable Model semantics [12]. In the abductive approach, negative literals are regarded as abducibles (assumptions). Besides, Eshghi and Kowalski have defined an abductive proof procedure which is an extension of SLDNF. In addition to the usual yes/no answer of SLDNF, the abductive procedure also provides an abductive explanation \(\Delta\); in this way, alternative abductive explanations are feasible for a given query.

Starting from Eshghi and Kowalski's seminal work, extensions of logic programming have been proposed which support more general forms of abduction. This has led to what is called Abductive Logic Programming (ALP) [15]. Semantics and proof procedures for ALP have been defined in [7, 5, 16], where abduction is used for both hypothetical reasoning and NAF.

In this paper, we will rather focus on the use of abduction to deal with negation as failure, and, in particular, on the relationships between stable models and abductive procedures like the one proposed by Eshghi and Kowalski. It is well known that, although Eshghi and Kowalski's procedure is correct with respect to the Stable Model semantics for call-consistent logic programs, it is not correct in general (in [8], some simple examples are given to point out this fact). To face this problem, two solutions have been pursued. The first one consists of modifying the proof procedure in order to be sound with respect to the Stable Model semantics. A goal-directed proof procedure of this kind has been defined in [28]. This procedure is defined for every general logic program with integrity constraints, and can be regarded as a combination of a modified model-elimination [21] and a forward evaluation of rules to check consistency of "implicit deletion" [27].

The second solution is to modify the semantics so as to fit Eshghi and Kowalski's procedure. This approach lies in a more general line of research, in which several semantics [25, 16, 26, 17, 6] have been proposed as generalizations of the Stable Model semantics of Gelfond and Lifschitz [12], applicable to any logic program. The need for an extension of stable models mainly comes from the fact that several programs have an evident intended meaning, but they do not have any stable model. This is due to two main characteristics of this semantics, i.e., the totality property of stable models (each atom must either be true or false in a given stable model) and their nonmodularity (the meaning of one part of a program can be modified by another unrelated part of it). The Well-Founded semantics [30] has the property of being defined for any program, but it only allows skeptical forms of reasoning.

Dung [6] has defined the Preferential Semantics, a declarative semantics for abduction where negation in logic programs is treated as a form of hypothesis. In this setting, the given logic program is seen as an abductive program, following the line introduced in [8], and its semantics is expressed by means of the preferred extensions of the abductive program. The preferred extensions semantics is a three-valued semantics, in which the totality requirement characterizing the Stable Model semantics is replaced by a maximality condition. Moreover, Dung has shown that Eshghi and Kowalski's abductive procedure is sound with respect to the Preferential semantics.
Neither Eshghi and Kowalski in the original version of their procedure [8], nor Dung in the revised one [6], explicitly mentioned the meaning of loops possibly occurring during the computation (and in the examples they proposed, no loop occurs). In particular, there are cases in which Eshghi and Kowalski's procedure loops indefinitely when applied to atoms belonging to a preferred extension of the given abductive program; therefore, if no loop checking is adopted, the procedure lacks the completeness property, not only with respect to the Stable Model semantics, but also with respect to the Preferential semantics.

Since Eshghi and Kowalski's procedure has been proposed as an extension of the Negation as Failure, it seems natural to regard failure in the procedure as finite failure. Under this view, a loop in the consistency phase of the procedure makes the overall procedure to loop.

In this paper, we define an abductive proof procedure, for finite propositional programs, which extends SLDNF-resolution and is based on the ideas underlying Eshghi and Kowalski's procedure. In our approach, failure is intended as finite failure. Besides, we define a three-valued semantics (the Finite Failure Stable Model Semantics, FF-SM) with respect to which our abductive procedure is sound and complete. This semantics is a three-valued generalization of the Stable Model semantics [12]. It is different from the eXtended Stable Model (XSM) semantics [25] since it does not capture infinite failure. The FF-SM semantics is defined in the style of Gelfond and Lifschitz Stable Model semantics, but it builds on an underlying Kripke/Kleene semantics [9, 18], in which loops causing nonterminating computations are modeled by means of the truth-value undefined.

The FF-SM semantics of a program is defined by first transforming the program with respect to a given three-valued interpretation $I$, and then by checking if the Kripke/Kleene fixed-point semantics of the transformed program coincides with $I$ (stability condition). If they coincide, $I$ is an FF-Stable model of the program. As a difference with the Gelfond and Lifschitz transformation, the transformation we apply to the program still gives a possibly negative program.

A comparison of our semantics with the eXtended Stable Model (XSM) semantics [25] is also given, based on the strong similarities in style between the definition of the semantics proposed in this paper and the definition of the XSM semantics proposed in [24, 22]. In both cases, first a transformation is applied to the program with respect to a three-valued interpretation, and then a stability condition is checked by computing the iteration at $\omega$ of a given immediate consequences operator $T_p$.

It is possible to see that our definition can be restated so that the transformation applied to the program is precisely the same as the one used in the definition of the XSM semantics. Hence, since the operator $T_p$ used in the two definitions is also the same, the only difference between our semantics and the XSM semantics is that we compute the iteration at $\omega$ starting from the interpretation in which every atom is undefined, while in the XSM semantics, the starting interpretation is the one in which all atoms are false. In essence, the least fixed point of the immediate consequence operator of the transformed program is computed in the two cases with respect to different orderings. We will make this clearer by reformulating the FF-SM semantics in the bilattice-based framework presented in [11]. While XSM semantics captures infinite failure (as the Well-Founded semantics does), the semantics we propose does not, since it is based on a Kripke/Kleene
semantics. For this reason, FF-SM semantics we propose is suitable to model our abductive proof procedure, which is an extension of SLDNF procedure.

The outline of the paper is the following. In Section 2, we introduce some notation and recall some results that will be used along the paper. In Sections 3 and 4, we define the FF-SM semantics and the abductive proof procedure, respectively. Moreover, we state soundness and completeness results of the abductive procedure with respect to the FF-SM semantics. In Section 5, we compare FF-SM semantics with XSM's semantics, and in Section 6 we give a bilattice-based restatement of our semantics. Finally, in Section 7, we give some conclusions. The Appendix contains the proof of the theorems stated in the paper.

2. PRELIMINARIES AND NOTATIONS

In the following, we will make use of concepts and notations standard in logic programming. We will consider finite propositional normal programs, i.e., finite sets of clauses of the form

\[ C \leftarrow A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m, \quad n \geq 0, m \geq 0 \]

where \( C, A_i, B_j \) are atomic propositions and the negation in the body is the negation as failure.

As usual, a goal is a clause of the form

\[ \leftarrow A_1, \ldots, A_n, \sim B_1, \ldots, \sim B_m, \quad n \geq 0, m \geq 0. \]

In [9, 18, 19], a Kripke/Kleene semantics has been defined for general logic programs, which makes use of Kleene's strong three-valued logic. The three truth-values are \( \text{true}, \text{false}, \) and \( \text{undefined} \), where the value \( \text{undefined} \) is intended to model computations which fail to return an answer. In the propositional case, a partial interpretation is any (total) function \( I \) from the set of all propositions into \( \{ \text{true}, \text{false}, \text{undefined} \} \). On the truth-values, an ordering relation \( <_k \) is defined as follows: \( \text{undefined} <_k \text{false} \) and \( \text{undefined} <_k \text{true} \).

From this ordering among truth-values, a partial ordering among interpretations is defined as follows: \( I <_k J \) (\( J \) extends \( I \) with respect to \( <_k \)) iff, for all propositions \( A \), \( I(A) <_k J(A) \) or \( I(A) = J(A) \).

Usually, a partial interpretation is represented as a pair \( I = \langle T, F \rangle \), where \( T (F) \) is the set of all the propositions \( \alpha \) such that \( I(\alpha) = \text{true} (I(\alpha) = \text{false} \), respectively.\)

The notion of partial interpretation can be naturally extended to a function \( \hat{I} \) defined on all conjunctions of literals by following Kleene's truth tables. In particular, we have that:

- \( \hat{I}(\sim A) = \text{true} (\text{false}) \text{ iff } I(A) = \text{false} (\text{true}) \);
- \( \hat{I}(L_1, \ldots, L_n) = \text{true} \text{ iff } \hat{I}(L_i) = \text{true}, \forall i = 1, \ldots, n; \)
  \( \hat{I}(L_1, \ldots, L_n) = \text{false} \text{ iff there exists an atom } L_i \text{ such that } \hat{I}(L_i) = \text{false} ; \)
  \( \hat{I}(L_1, \ldots, L_n) = \text{undefined}, \text{ otherwise.} \)

In the following, since there is no risk of ambiguity, we will also use the simpler notation \( I \) (instead of \( \hat{I} \)) to represent the extended interpretation function.
For each program \( P \), an operator \( T_P \) is defined, mapping interpretations to interpretations. The mapping \( T_P \), applied to a partial interpretation \( I \), gives another partial interpretation \( J = T_P(I) \) such that for each atomic proposition \( A \):

1. \( J(A) = \text{true} \) iff for some clauses \( A \leftarrow L_1, \ldots, L_n \) in the given program \( P \), \( I(L_1, \ldots, L_n) = \text{true} \);
2. \( J(A) = \text{false} \) iff for all clauses \( A \leftarrow L_1, \ldots, L_n \) in the given program \( P \), \( I(L_1, \ldots, L_n) = \text{false} \);
3. \( J(A) = \text{undefined} \), otherwise.

It is worth noting that the function \( T_P \) does not have the property of being increasing with respect to the ordering relation \( \leq_k \), that is, \( I \leq_k T_P(I) \) need not hold, but it is monotone, i.e., \( I \leq_k J \) implies \( T_P(I) \leq_k T_P(J) \).

Since the space of all interpretations is a complete semilattice under the \( \leq_k \) ordering, \( T_P \) has a least fixed-point, \( I^\infty = T_P^\infty \), which can be constructed by transfinite recursion by defining \( I^0 = T_P^0 = (\emptyset, \emptyset) \), \( I^\mu + 1 = T_P(I^\mu) \), and taking limits at limit ordinals.

Since we only deal with finite propositional programs, the fixed-point \( I^\infty \) is equal to \( I^n \) for some \( n < \omega \). Hence, \( \text{lfp}(T_P) = T_P^n \).

The operator \( T_P \) will play a fundamental role in the definition of our semantics. In the following, we will make use of the results of soundness and completeness of SLDNF-resolution (whose definition will be recalled in Section 4) with respect to the fixed-point semantics (when the iteration of the \( T_P \) operator is cut at a finite ordinal \( n \)). We will refer to the results in [18] for soundness and completeness of SLDNF in the propositional case. Completeness of SLDNF in the first-order case has been proved in [19] under the allowedness condition.

In the propositional case, this result can be stated as follows: given a program \( P \) and a goal \( G \),

- \( G \) succeeds via SLDNF in \( P \) iff \( T_P^n(G) = \text{true} \);
- \( G \) finitely fails via SLDNF in \( P \) iff \( T_P^n(G) = \text{false} \).

3. THE FF-STABLE-MODEL SEMANTICS

In this section, we define the Finite Failure Stable Model (FF-SM) semantics, which is obtained by extending the notion of Stable Model [12], building on the basis of a Fitting/Kunen semantics. In the next section, we will define an abductive proof procedure which is an extension of SLDNF quite similar to Eshghi and Kowalski's procedure and is sound and complete with respect to FF-SM semantics.

Let us first recall the definition of the Stable Model semantics.

**Definition 3.1.** [The Stable Model semantics [12]] Given a ground logic program \( P \), for any set \( I \) of atoms let \( P(I) \) be the (positive) program obtained by deleting from \( P \):

- each clause containing a negative literal \( \sim A \), with \( A \in I \),
- all negative literals in the remaining clauses.

\(^1\)We specialize to the propositional case the definition given in [18] for the first-order case.

\(^2\)A program \( P \) is allowed if, for each clause \( A \leftarrow L_1, \ldots, L_n \) in \( P \), each variable \( X \) which occurs anywhere in that clause occurs in at least one positive literal \( L_i \) in its body [19].
If the minimal (Herbrand) model of \( P(I) \) coincides with \( I \), then \( I \) is a Stable Model of \( P \).

In the style of Stable Model semantics, to introduce our notion of FF-Stable Model, we first define how a program is transformed, given a three-valued interpretation \( I \). Then, we define the stability condition that must be satisfied for the interpretation \( I \) to be a FF-Stable Model.

**Definition 3.2.** [The transformed program] Given a finite propositional program \( P \) and an interpretation \( I = \{ T, F \} \), we define the transformed program \( P_I \) as the program obtained from \( P \) by removing from the body of its clauses all the negative literals \( \neg B \) such that \( B \neg F \).

Notice that, as a difference with respect to the transformation of Gelfond and Lifschitz [12] and its extension proposed by Przymuzinski [25], our transformed program \( P_I \) possibly contains negative literals: the transformation only removes the negative literals \( \neg B \) which are true in the given interpretation, and retains all the other negative literals and clauses. Notice also that we require that the propositional program \( P \) is finite.

To define the FF-Stable-Model semantics, given the transformed program \( P_I \), we will make use of the \( T_e \) operator [18], which has been recalled in Section 2. A given interpretation \( I \) is an FF-Stable-Model of \( P \) iff the least fixed-point (in the ordering \( \leq_k \)) of the operator \( T_{P_I} \) coincides with \( I \). In particular, the atoms which are false in \( I \) must be false in \( T_{P_I} \).

**Definition 3.3.** [The FF-Stable-Model] Given a finite propositional program \( P \) and an interpretation \( I = \{ T, F \} \), we say that \( I \) is a FF-Stable-Model of \( P \) iff

\[
I = \text{lfp}(T_{P_I}) = T_{P_I}^w.
\]

**Example 1.** Given the program

\[
P = \begin{cases}
(1) & w \leftarrow t \\
(2) & t \leftarrow \neg s \\
(3) & s \leftarrow \neg w \\
(4) & a \leftarrow r, \neg p \\
(5) & r \leftarrow r \\
(6) & p
\end{cases}
\]

let \( I = \langle \{ p, w, t \}, \{ a, s \} \rangle \) be an interpretation. In order to verify whether \( I \) is an FF-Stable Model of \( P \), we transform \( P \) into \( P_I \):

\[
P_I = \begin{cases}
(1) & w \leftarrow t \\
(2) & t \\
(3) & s \leftarrow \neg w \\
(4) & a \leftarrow r, \neg p \\
(5) & r \leftarrow r \\
(6) & p
\end{cases}
\]

and then we compute the least fixed-point of \( T_{P_I} \):

\[
T_{P_I}^0 = \langle \emptyset, \emptyset \rangle, \quad T_{P_I}^1 = \langle \{ p, t \}, \emptyset \rangle, \\
T_{P_I}^2 = \langle \{ p, t, w \}, \{ a \} \rangle, \quad T_{P_I}^3 = \langle \{ p, t, w \}, \{ a, s \} \rangle = T_{P_I}^4 = T_{P_I}^w = I.
\]

Since \( \text{lfp}(T_{P_I}) = I \), \( I \) is an FF-Stable-Model of \( P \). Another FF-SM of \( P \) is \( I' = \langle \{ p, s \}, \{ a, w, t \} \rangle \). In both of these models, \( r \) is undefined.

**Example 2.** Let us consider the program

\[
P = \begin{cases}
(1) & q \leftarrow \neg r \\
(2) & r \leftarrow r
\end{cases}
\]
$P$ has a unique FF-SM $I = \langle \emptyset, \emptyset \rangle$ in which $q$ and $r$ are both undefined; on the contrary, $P$ has a unique preferred extension $P \cup \{ \neg r \}$ [6] in which $q$ is true and $r$ is false. Similarly, the interpretation $\langle \{q\}, \{r\} \rangle$ is the unique XSM [25] for $P$. In Section 6, we will compare more carefully FF-SM semantics with the XSM semantics.

**Example 3.** To see the difference between the FF-SM semantics and the Stable Model semantics, consider the following program,

\[
P = \begin{cases} 
(1) & r \leftarrow \neg r \\
(2) & r \leftarrow q \\
(3) & p \leftarrow \neg q \\
(4) & q \leftarrow p
\end{cases}
\]

which was used in [8] to show that Eshghi and Kowalski's procedure is not correct, in general, with respect to Stable Model semantics. $P$ has two FF-Stable models: $M_1 = \langle \{r, q\}, \{p\} \rangle$ and $M_2 = \langle \{p\}, \{q\} \rangle$. The first one coincides with the (unique) stable model of $P$. The second one is not a stable model, since $r$ is undefined in it.

It is easy to prove that every program $P$ has at least one FF-Stable model, which is the model $M = \text{lfp}(T_P) = T_P^\omega$. Such a model is the least one with respect to the $\preceq_k$ ordering relation.

4. THE ABDUCTIVE PROOF PROCEDURE

In [8], Eshghi and Kowalski's procedure has been proposed as an extension of NAF in logic programming. Logic programs are given an abductive interpretation in which negative literals are considered as abductive hypotheses. These hypotheses can be assumed if they are consistent with the program, given a set of integrity constraints.

A given program $P$ is transformed into an abductive framework $\langle P^*, I, A \rangle$, where $P^*$ is obtained from $P$ by replacing each negative literal $\neg B$ with a new literal $B^*$ (called abducible), $A$ is the set of the new abducible symbols introduced, and $I$ is a set of constraints sanctioning the mutual exclusion of $B$ and $B^*$ ($\neg B \wedge B^*$) and the totality requirement ($B \vee B^*$), according to which if $B$ cannot be proved, then $B^*$ must be assumed. Given a goal $G$ and an abductive framework $\langle P^*, I, A \rangle$, an abductive solution (explanation) for $G$ is a set of abducibles $\Delta \subseteq A$ such that $P^* \cup \Delta$ derives $G$ and satisfies the constraints $I$. It has been shown [8] that there is a one-to-one correspondence between the stable models of a program $P$ and the abductive solutions of the associated abductive framework. In this setting, the original program $P$ is given a semantics in terms of abductive solutions $\Delta$ of the abductive framework $\langle P^*, I, A \rangle$: a conclusion $G$ holds in $P$ if there exists an abductive solution $\Delta$ for $G$ in $\langle P^*, I, A \rangle$.

Eshghi and Kowalski's abductive procedure is proposed by the authors as an effective method to compute abductive explanations for a given goal. It applies to the transformed program $P^*$, and consists of two interleaved phases of computation. The first (abductive) one is an SLD resolution phase which reasons backward, looking for a refutation of the current goal and collects the required (negative) hypotheses. The second one, the consistency phase, checks consistency with respect to the integrity constraints of the collected hypotheses. Eshghi and Kowalski have pointed out that the abductive procedure is not correct in general, that is, it may compute explanations for a goal which do not correspond to any abductive solution. It is correct for call-consistent programs.
In the following, we give a slight variant of Eshghi and Kowalski's procedure which is sound and complete with respect to the FF-Stable-Model Semantics introduced in the previous section. In its original version, Eshghi and Kowalski's procedure is formulated by means of two mutually recursive procedures, for the abductive and the consistency phase, respectively. Here, we adopt a definition based on the notions of success derivation, corresponding to the abductive phase, and finitely failing derivation, corresponding to the consistency phase. This formulation makes it clearer that our abductive procedure is an extension of SLDNF.

Let us first recall the notions of SLDNF success derivation and SLDNF finitely failing derivation, and introduce some notation. The notion of rank is the same as the one used in [20].

**Definition 4.1. [SLDNF success derivation]** Given a program $P$, a goal $G$, and a computation rule $R$, an SLDNF success derivation for $G$ (via $R$) of rank $r$ in $P$ is a sequence of goals:

$$G_1, \ldots, G_h$$

where $G_1 = G$, $G_h = \Box$, and, for all $k = 1, \ldots, h - 1$, $G_{k+1}$ is obtained from $G_k$ by means of one of the following rules.

Let $G_k = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$, and let $L_i$ be the literal in $G_k$ selected by $R$.

- $(R A^*_1)$ If $L_i = A$ and there exists in $P$ a rule
  
  $$A \leftarrow B_1, \ldots, B_m, (m \geq 0)$$

  then $G_{k+1} = L_1, \ldots, L_{i-1}, B_1, \ldots, B_m, L_{i+1}, \ldots, L_n$.

- $(R \sim A^*_1)$ If $L_i = \sim A$, and there exists an SLDNF finitely failing derivation for \{A\} (via $R$) of rank $r' < r$, then $G_{k+1} = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$.

**Definition 4.2. [SLDNF finitely failing derivation]** Given a program $P$, a set of goals $F$, and a computation rule $R$, an SLDNF finitely failing derivation for $F$ (via $R$) of rank $r$ is a sequence

$$F_1, \ldots, F_h$$

where $F_1 = F$, $F_h = \{\}$, and, for all $k = 1, \ldots, h - 1$, $F_{k+1}$ is obtained from $F_k$ by means of one of the following rules. Let $G = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$ be a goal in $F_k$, and let $L_i$ be the literal in $G$ selected by $R$.

- $(FA^*_1)$ If $L_i = A$, let
  
  $$A \leftarrow B_1^i, \ldots, B_k^i, (j = 1, \ldots, m)$$

  be all the clauses defining $A$ in $P$.

  We define

  $$F_{k+1} = F_k \setminus \{G\} \cup \{G_1, \ldots, G_m\},$$

  where

  $$G_j = L_1, \ldots, L_{i-1}, B_1^j, \ldots, B_k^j, L_{i+1}, \ldots, L_n$$

  $(j = 1, \ldots, m)$. 

(FA^*_1) If \( L_i = A \) and there is no definition for \( A \) in \( P \) (i.e., if \( m = 0 \)), then \( F_{k+1} = F_k \setminus \{G\} \).

\( F \sim A^*_1 \) If \( L_i = \sim A \) and there exists an SLDNF success derivation for \( A \) (via \( R \)) of rank \( r' < r \), then \( F_{k+1} = F_k \setminus \{G\} \).

**Definition 4.3.** Given a program \( P \) and a goal \( G \), we say that \( G \) **succeeds** via SLDNF in \( P \) if there exist a computation rule \( R \) and a rank \( r \) such that \( G \) has an SLDNF success derivation via \( R \) of rank \( r \).

**Definition 4.4.** Given a program \( P \) and a goal \( G \), we say that the goal \( G \) **finitely fails** via SLDNF in \( P \) if there exist a computation rule \( R \) and a rank \( r \) such that \{\( G \)\} has an SLDNF finitely failing derivation via \( R \) of rank \( r \).

Let us move to the definition of our abductive procedure. We have already mentioned that, as a difference with Eshghi and Kowalski’s abductive procedure, its main peculiarity is the interpretation of failure as finite failure. To stay closer to SLDNF, in the following algorithm we will keep on using \( \sim B \) instead of \( B^* \).

**Definition 4.5.** [Abductive success derivation] Given a program \( P \), a goal \( G \), a computation rule \( R \), and a set of negative assumptions \( A \), an abductive success derivation for \((G, A)\) (via \( R \)) of rank \( r \) with computed answer \( A' \) is a sequence

\[
(G_1, \Delta_1), \ldots, (G_h, \Delta_h)
\]

where \( G_1 = G \), \( \Delta_1 = \Delta \), \( G_h = \Box \), \( \Delta_h = \Delta' \), and, for each \( k = 1, \ldots, h - 1 \), \((G_{k+1}, \Delta_{k+1})\) is obtained from \((G_k, \Delta_k)\) by means of one of the following rules.

Let \( G_k = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n \), and let \( L_i \) be the goal in \( G_k \) selected by \( R \).

\( RA_1 \) If \( L_i = A \) and there exists in \( P \) a clause

\[
A \leftarrow B_1, \ldots, B_m \quad (m \geq 0)
\]

then \( G_{k+1} = L_1, \ldots, L_{i-1}, B_1, \ldots, B_m, L_{i+1}, \ldots, L_n \) and \( \Delta_{k+1} = \Delta_k \).

\( RA_2 \) If \( L_i = \sim A \) and \( \sim A \in \Delta_k \), then \( G_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n \) and \( \Delta_{k+1} = \Delta_k \).

\( RA_3 \) If \( L_i = \sim A \) and \( \sim A \notin \Delta_k \) and there exists an abductive finitely failing derivation (via \( R \)) of rank \( r' < r \) for \((\{A\}, \Delta_k \cup \{\sim A\})\) with computed answer \( \Delta_f \), then \( G_{k+1} = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n \) and \( \Delta_{k+1} = \Delta_f \).

**Definition 4.6.** [Abductive finitely failing derivation] Given a program \( P \), a set of goals \( F \), a computation rule \( R \), and a set of negative assumptions \( \Delta \), an abductive finitely failing derivation for \((F, \Delta)\) (via \( R \)) of rank \( r \) with computed answer \( \Delta' \) is a sequence

\[
(F_1, \Delta_1), \ldots, (F_h, \Delta_h)
\]

such that for all \( k = 1 \ldots h \), the set of goals \( F_k \) does not contain the empty clause (i.e., the goal true), \( F_1 = F \), \( \Delta_1 = \Delta \), \( F_h = \emptyset \), \( \Delta_h = \Delta' \), and for each
$k = 1, \ldots, h - 1$, $(F_{k+1}, \Delta_{k+1})$ is obtained from $(F_k, \Delta_k)$ by means of one of the following rules.

Let $G = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$ be a goal in $F_k$, and let $L_i$ be the literal in $G$ selected by $R$.

$(FA_1)$ If $L_i = A$, let

$$A \leftarrow B_{j_1}, \ldots, B_{j_m}, (j = 1, \ldots, m)$$

be all the clauses defining $A$ in $P$.

We define $F_{k+1} = (F_k \setminus \{G\}) \cup \{G_1, \ldots, G_m\}$, and $\Delta_{k+1} = \Delta_k$, where for all $j = 1, \ldots, m$

$$G_j = L_1, \ldots, L_{i-1}, B_{j_1}, \ldots, B_{j_m}, L_{i+1}, \ldots, L_n.$$ 

$(FA_2)$ If $L_i = A$ and $A$ is not defined in $P$ (i.e., $m = 0$), then $F_{k+1} = F_k \setminus \{G\}$, and $\Delta_{k+1} = \Delta_k$.

$(FA_3)$ If $L_i = \neg A$ and there exists an abductive success derivation (via $R$) of rank $r' < r$ for $(A, \Delta_k)$ with computed answer $\Delta_i$, then $F_{k+1} = F_k \setminus \{G\}$ and $\Delta_{k+1} = \Delta_i$.

$(FA_4)$ If $L_i = \neg A$ and $\neg A \in \Delta_k$ then

$$F_{k+1} = F_k \setminus \{G\} \cup \{G'\}$$

where $G' = L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, and $\Delta_{k+1} = \Delta_k$.

Notice that if in case $(FA_1)$ some $G_j$ is the empty clause, then the corresponding derivation cannot terminate with $F_h = \{\}$ because of the empty clause which will not be eliminated.

Both in the abductive success derivations and in the abductive finitely failing derivations, the sequence $\Delta_1, \Delta_2, \ldots$ is, by construction, monotonically increasing.

It is clear that in a derivation of rank 0, the rules $(FA_1)$ and $(FA_2)$ are never applied.

Definition 4.7. Given a program $P$, a goal $G$, and a set of negative assumptions $\Delta$, we say that the pair $(G, \Delta)$ succeeds with computed answer $\Delta'$ if there exist a computation rule $R$ and a rank $r$ such that $(G, \Delta)$ has a success derivation, via $R$, of rank $r$ with computed answer $\Delta'$.

Definition 4.8. Given a program $P$, a goal $G$, and a set of negative assumptions $\Delta$, we say that the pair $(G, \Delta)$ finitely fails with computed answer $\Delta'$ if there exist a computation rule $R$ and a rank $r$ such that $(G, \Delta)$ has a finitely failing derivation, via $R$, of rank $r$ with computed answer $\Delta'$.

The abductive procedure above is very similar to SLDNF: indeed, it differs from SLDNF only for the fact of remembering assumptions, and for the presence of the additional rules $(FA_1)$, in the definition of the success derivation, and $(FA_2)$, in the definition of the finitely failing derivation.
In this formulation of both the SLDNF resolution and the abductive proof procedure, the symmetry between success derivations and finitely failing derivations is clear. In a success derivation, the selection of the literal in the goal (by R) represents a "don't care" nondeterministic choice, and the selection of the clause to be resolved with the goal represents a "don't know" nondeterministic choice. In a finitely failing derivation, the selection of the literal in the goal (by R) is a "don't know" choice, and the selection of the goal G to be falsified from F_k is a "don't care" one (all the goals in F must be falsified).

Notice that in the case of the abductive derivation, both success and finitely failing derivations return a set of abducibles as computed answer. It follows that, as a difference with the case of SLDNF, given a goal G and a set of negative assumptions Δ, there can be both a success derivation for (G, Δ) and a finitely failing derivation for ((G), Δ). Also, the fact that (G, Δ) succeeds from P via R, with computed answer Δ', does not imply that (G, Δ) succeeds via any computation rule R' which gives a fair computation. We will come back to this point later in this section.

The abductive success derivation and the finitely failing one can be naturally related to a dynamic notion of computation. Given a program P, a goal G, and a set of negative assumptions Δ, an abductive computation is an attempt to construct a success derivation for (G, Δ) or a finitely failing derivation for ((G), Δ). As an example, given the simple program P:

\[
P = \begin{cases} 
(1) & a \leftarrow \neg b \\
(2) & b \leftarrow \neg a \\
(3) & c \leftarrow a \\
(4) & c \leftarrow b 
\end{cases}
\]

consider the computation for (c, ∅). First, rule (RA_1) is applied, resulting in a resolution step on the given goal c, using a clause of P having c as its head. Figure 1 shows the computation obtained for the goal c when the third clause in P is used: the procedure succeeds, and it "motivates" the answer yes to the query with the set of negative assumptions Δ_1 = {¬ b} supporting the success.

If the second clause defining c were used in the first resolution step, the other alternative support to the answer would be found, i.e., Δ_2 = {¬ a}. Δ_1 and Δ_2 correspond to the two FF-Stable models of P, ⟨{a, c}, {b}⟩ and ⟨{b, c}, {a}⟩, which coincide with Gelfond and Lifschitz' two-valued stable models of P, M_1 = {a, c} and M_2 = {b, c}.
Let us come back to Example 3. We have already mentioned that the program $P$ was given by Eshghi and Kowalski in [8] to show that their procedure is not correct (in the general case) with respect to Gelfond and Lifschitz' Stable Model semantics. The same example shows that the abductive procedure we have described above is not correct with respect to Gelfond and Lifschitz' Stable Model semantics.

The program

$$
P = \begin{cases} 
(1) & r \leftarrow \neg r \\
(2) & r \leftarrow q \\
(3) & p \leftarrow q \\
(4) & q \leftarrow \neg p
\end{cases}
$$

has a unique stable model \{r, q\}, but the abductive computation of \((p, \emptyset)\) succeeds with computed answer $\Delta' = \{\neg q\}$.

We have already seen, however, that the program has an FF-Stable model \(\langle\{p\}, \{q\}\rangle\) which corresponds to the abductive solution found by the operational semantics. Another FF-Stable model of $P$ is \(\langle\{r, q\}, \{p\}\rangle\), which coincides with the unique Gelfond and Lifschitz' stable model.

Notice that, as for SLDNF, it may happen that, given a goal $G$ and a set of assumptions $\Delta$, neither a success derivation for $(G, \Delta)$ nor a finitely failing derivation for $(\{G\}, \Delta)$ exists. This is the case, for instance, when the computation of $(G, \emptyset)$ enters a loop. Of course, when a part of the computation enters a loop, the overall computation does. As an example, consider again the simple program $P$:

$$
P = \begin{cases} 
(1) & q \leftarrow \neg r \\
(2) & r \leftarrow r
\end{cases}
$$

and the computation of $(q, \emptyset)$. When trying to construct a success derivation, the rules $(RA_1)$ and $(R \sim A_2)$ are applied; therefore, another computation, looking for a finitely failing derivation for $(\{r\}, \{\neg r\})$, is activated. In this case, the “subcomputation” enters a loop: it indefinitely applies the rule $(FA_1)$, resolving the literal $r$ with the second clause of the given program, and then also the external computation loops.

The behavior just mentioned is really similar to the way the SLDNF procedure works: given the same program, in order to prove $q$, a finitely failing proof for $r$ is looked for; this proof cannot be found since the computation enters a loop, and therefore the global proof of $q$ also does not terminate. Correspondingly, we have seen in Section 3 that $P$ has a unique FF-Stable model $I = \langle\emptyset, \emptyset\rangle$ in which both $r$ and $q$ are undefined.

The existence of nonterminating computations is also responsible for the incompleteness of our abductive procedure with respect to the preferential semantics of Dung [6]. The previous example is sufficient to point out this fact. Indeed, the program $P$ has a preferred extension $P \cup \{\neg r\}$ in which the literal $q$ is true, while the computation of $(q, \emptyset)$ loops (as illustrated above).

### 4.1. Comparison with Eshghi and Kowalski's Procedure

As we have already pointed out, the abductive procedure above is not equivalent to Eshghi and Kowalski's procedure; the main difference between the two concerns the definition of the consistency derivation (that we call finitely failing derivation). Indeed, both Eshghi and Kowalski's consistency derivation and our finitely failing one, when activated on a pair $(\{B\}, \Delta)$, look for a finite sequence of pairs, having $(\{B\}, \Delta)$ as its first element, and ending in a pair whose set of goals is empty. As far
as resolution steps, replacing a goal in the current set \( F_i \) by means of other goals (possibly none) are concerned, the two procedures have the same behavior. Both Eshghi and Kowalski's consistency derivation procedure and our finitely failing one remove the chosen goal from the current set \( F_i \) if one of the following two conditions holds: either the selected literal in the current goal is a positive literal which is not defined in the program, or it is a negative literal whose complement has an abductive derivation (abductive success derivation, respectively).

It may be the case that the literal selected from the chosen goal is a negative one, say \( \sim A \), and no success derivation for \((A, \Delta)\) is found: this means that the selected literal \( \sim A \) cannot be used to prove the failure of the current goal. To deal with this case, Eshghi and Kowalski's consistency derivation procedure assumes the falsity of \( A \), and goes on selecting from the current goal another literal, if any, to prove the falsity of the goal itself.

In [6], Dung has pointed out that, in the presence of infinite derivations, Eshghi and Kowalski's procedure lacks the correctness property, even for stratified programs. In particular, he showed that it is not correct to regard an infinite derivation as an infinitely failing one, and he proposed a revised version of the procedure differing from the original one as regards this choice. According to Dung's procedure, when a negative literal is selected from the current goal, and its complement does not have any success derivation, the chosen literal is simply removed from the current goal (while Eshghi and Kowalski's procedure would also assume it to hold), and another literal, if any, is selected from the current goal, trying to prove its falsity. The revised procedure is correct with respect to Dung's preferential semantics.

In order to point out the difference among our procedure, Eshghi and Kowalski's procedure, and Dung's revised version, let us consider the following example, originally given by Dung. Let \( P \) be the following stratified program:

\[
P = \begin{cases} 
a & \leftarrow \sim b \\
b & \leftarrow \sim c, \sim r \\
r & \leftarrow \sim c \\
c & \leftarrow \sim d \\
d & \leftarrow d 
\end{cases}
\]

\( P \) has \( M = \{c, a\} \) as its unique stable model.

Figure 2 shows the search space of a success derivation for the pair \((a,\{\})\) according to Eshghi and Kowalski's procedure. The derivation step marked with (***) is the one that, in the consistency derivation, takes into account the absence of any success derivation for the pair \((c,\{\sim b\})\), due to the fact that, when trying to build such a derivation, a loop is entered repeatedly resolving the goal \( \leftarrow d \) with the last clause of the program. The loop is recognized, and, according to Eshghi and Kowalski's procedure, \( \sim c \) is assumed; therefore, a success derivation for \((a,\{\})\) is built, returning the set \( \{\sim b, \sim c\} \) as computed answer. On the contrary, \( c \) belongs to the unique stable model \( M \) also containing \( a \). Therefore, the computed answer is not correct with respect to the Stable Model Semantics. Dung's revised version of the procedure, instead, does not assume the negative hypothesis \( \sim c \), and returns the set only containing the negative assumption \( \sim b \).

It turns out from the previous example that, in general, recognizing the absence of any success derivation for a pair \((A, \Delta)\) may need some loop detection mechanisms.
Our abductive procedure, like SLDNF resolution [20], does not take into account the case of infinite failure, since it implements the notion of failure as finite failure, and it does not rely on any loop-checking mechanism. Hence, in our procedure, the query \((a, \{\})\) has a looping computation from the program \(P\), and, correspondingly, \(P\) has a unique FF-Stable model in which every proposition is undefined.

Since our abductive procedure is an extension of SLDNF, we will show that, as SLDNF is sound and complete with respect to Fitting/Kunen semantics, our abductive procedure is sound and complete with respect to the FF-SM semantics defined in Section 3, which is obtained by extending stable models, building on the basis of a Fitting/Kunen semantics.

### 4.2. Properties of the Abductive Procedure

In this section, we will introduce some properties characterizing our abductive procedure.

We have mentioned above that, given a program \(P\), a goal \(G\), and a set of negative assumptions \(\Delta\), there can be both a success derivation for \((G, \Delta)\) and a finitely failing derivation for \(\langle G, \Delta \rangle\). In this case, the success and the failure of \(G\) are supported by two different sets of assumption returned by the procedure.

As an example, let us consider the following program \(P\):

\[
P = \begin{cases} 
  d \leftarrow a \\
  a \leftarrow \sim b \\
  b \leftarrow \sim a.
\end{cases}
\]

Let \(G = \leftarrow d\) and \(\Delta = \{\}\) be the given goal and assumption set. There are both a success derivation for \((\leftarrow d, \{\})\), with computed answer \(\Delta' = \{\sim b\}\), and a finitely failing derivation for \(\langle d, \{\} \rangle\), which returns \(\{\sim a\}\). The two derivations are shown in Figures 3 and 4.

Moreover, as mentioned above, the fact that \((G, \Delta)\) has a success derivation from \(P\), via \(R\), with computed answer \(\Delta'\) does not imply that \((G, \Delta)\) succeeds via
any other computation rule $R'$ which gives a fair computation. Consider the following example:

$$p = \{ g \leftarrow \neg a, \neg c \}
\begin{align*}
a &\leftarrow \neg b, \neg c \\
b &\leftarrow \neg c \\
c &\leftarrow \neg b.
\end{align*}$$

In Figure 5, a success derivation for $(g, \emptyset)$ via the computation rule $R_1$, with computed answer $\Delta_1 = \{ \neg a, \neg c \}$, is shown.

$R_1$ selects $\neg a$ from the goal $\neg a, \neg c$, and $\neg b$ from $\neg b, \neg c$. On the other hand, if the computation rule $R_2$, which selects $\neg a$ from $\neg a, \neg c$ and $\neg c$ from $\neg b, \neg c$ is taken, no success derivation is obtained (see the computation in Figure 6).

Note that, in the computation shown in Figure 6, the subgoal $\neg c$ does not succeed since there is no finitely failing derivation for $((c), (\neg a, \neg b))$.

4.3. Soundness and Completeness Results

In this section, we state soundness and completeness of our abductive procedure with respect to the FF-SM semantics. The proof of these results will rely on the soundness and completeness of SLDNF-resolution w.r.t. the Kripke/Kleene semantics (in particular, we will make use of the result by Kunen recalled in Section 2), and will be given in the Appendix.
Theorem 4.1. The abductive proof procedure is **sound** w.r.t. the FF-Stable Model semantics, i.e., given a program P and a goal G,

- if \((G, \emptyset)\) succeeds from P with computed answer \(\Delta\) according to the abductive procedure, then there exists an FF-Stable Model \(I = \langle T, F \rangle\) for P such that \(I(G) = \text{true}\) and \(\Delta \subseteq \Delta_I\), where \(\Delta_I = \sim F\);
- if \((G, \emptyset)\) finitely fails from P with computed answer \(\Delta\) according to the abductive procedure, then there exists an FF-Stable Model \(I = \langle T, F \rangle\) for P such that \(I(G) = \text{false}\), and \(\Delta \subseteq \Delta_I\), where \(\Delta_I = \sim F\).

Theorem 4.2. The abductive procedure is **complete** w.r.t. the FF-SM semantics, i.e., given a program P and a goal G,

- if there exists an FF-Stable Model \(I = \langle T, F \rangle\) for P such that \(I(G) = \text{true}\), then \((G, \emptyset)\) succeeds from P with computed answer \(\Delta\) according to the abductive procedure, and \(\Delta \subseteq \Delta_I\), where \(\Delta_I = \sim F\);

**FIGURE 5.** Success derivation for \((g, \emptyset)\) via \(R_1\).

**FIGURE 6.** Absence of success derivation for \((g, \emptyset)\) via \(R_2\).
if there exists an FF-Stable Model $I = \langle T, F \rangle$ for $P$ such that $I(G) = \text{false}$ then $\langle \{G\}, \emptyset \rangle$ finitely fails from $P$ with computed answer $\Delta$ according to the abductive procedure, and $\Delta \subseteq \Delta_f$, where $\Delta_f = \sim F$.

5. COMPARISON WITH THE XSM-SEMANTICS

As we have shown in Section 3, the FF-SM semantics is a three-valued extension of the Stable Model semantics, which gives different results with respect to the XSM semantics [25]. In this section, we compare FF-SM semantics and XSM semantics more carefully. To this purpose, we will refer to the definition of XSM semantics given in [24]. There, XSMs are defined by first applying a transformation to the program $P$ with respect to a given three-valued interpretation $I$, and then taking the least three-valued model $\Gamma(P/I)$ of the resulting program $P/I$, with respect to the partial ordering relation $\leq_t$ between interpretations (see below). If $\Gamma(P/I)$ coincides with $I$, then $I$ is an eXtended Stable Model. The transformation applied to the program is the following.

Definition 5.1. [Extended GL-transformation [24]] Given a program $P$ and an interpretation $I = \langle T, F \rangle$, the extended GL-transformation of $P$ modulo $I$ is a (nonnegative) program $P/I$ obtained from $P$ by performing the following three operations:

- Removing from $P$ all clauses which contain a negative premise $L = \sim A$ such that $A \in T$;
- Replacing in all remaining clauses those negative premises $L = \sim A$ such that $A \not\in T \cup F$ by $u$;
- Removing from all the remaining clauses those negative premises $L = \sim A$ such that $A \in F$.

Note that the transformed program $P/I$ does not contain any negative literal: it is a nonnegative program with occurrences of $u$ literals, whose value is undefined in any three-valued interpretation. It has been proved that such a program has a unique least three-valued model $\Gamma(P/I)$, taking the ordering $<_t$ among truth-values as follows: false $<_t$ undefined $<_t$ true. From the relation $<_t$, the partial ordering $\leq_t$ among truth-values can be defined as follows: $a \leq_t b$ holds if either $a <_t b$ or $a = b$. The partial ordering $\leq_t$ can be extended to interpretations as usual. While the ordering relation $\leq_t$ compares the “degree of truth” of the truth-values, the ordering relation $\leq_k$ (introduced in Section 2) compares their “degree of knowledge.”

Notice, moreover, that the third step in the definition of the GL-transformation is exactly the transformation in terms of which the FF-SM semantics is defined.

Definition 5.2. [eXtended Stable Model] A three-valued interpretation $I$ is an eXtended Stable Model of $P$ iff $\Gamma(P/I) = I$.

It has been shown [24, 22] that the least three-valued model of $P/I$ (in the $\leq_t$ ordering), $\Gamma(P/I)$, can be given a constructive definition as the iteration at $\omega$ of an immediate consequence operator. Such an operator coincides with the mapping $T_p$ defined by Kunen [18] and recalled in Section 2. As a difference with the
Kripke/Kleene semantics in [18], $\Gamma(P/I)$ is the iteration at $\omega$ of the operator $T_{P/I}$ starting from the initial interpretation $I_0^P = \langle \emptyset, H \rangle$, in which all propositions have value $false$.

Let $T_P \uparrow n$ be the iteration of $T_P$ at the finite ordinal $n$, starting from $I_0^P$, as follows:

- $T_P \uparrow 0 = \langle \emptyset, H \rangle$, where $H$ is the Herbrand base;
- $T_P \uparrow (n + 1) = T_P(T_P \uparrow n)$.

It has been proved that $T_{P/I} \uparrow \omega$ is the least three-valued model of $P/I$ (in the $\leq_t$ ordering). Hence, it holds that: $I$ is an extended Stable Model of $P$ iff $I = T_{P/I} \uparrow \omega$.

Notice that, while we have denoted by $T_e \uparrow n$ the nth iteration of the $T_e$ operator starting from the initial interpretation $I_0^E = \langle \emptyset, H \rangle$, in Section 2 we have denoted by $T_e^\omega$ the nth iteration of the $T_e$ operator starting from the interpretation $I_0 = \langle \emptyset, \emptyset \rangle$.

We have mentioned that $T_e^\omega$ is the least fixed-point of $T_e$ under the ordering $\leq_k$ among the interpretations. On the other hand, if $P$ is a nonnegative program (as the program resulting from the extended GL-transformation above), then $T_P \uparrow \omega$ is the least fixed-point of $T_P$ under the ordering $\leq_t$ among truth-values.

Our definition of FF-SM semantics and this alternative definition of XSMs [24] are very similar in style. They differ in two points: the first one is the transformation of the program with respect to the given interpretation; the second one is the ordering under which the fixed point of the $T_e$ operator is computed.

The next theorem shows that the first difference is not relevant: in fact, in the definition of FF-SM semantics, the transformation defined in [24] can be used instead of the one in Section 3 (i.e., $P_1$ can be replaced by $P/I$), without affecting the semantics itself.

**Theorem 5.1.** Given a program $P$ and an interpretation $I = \langle T, F \rangle$, $I = T_{P_1}^\omega$ (i.e., $I$ is an FF-Stable Model of $P$) iff $I = T_{P/I}^\omega$.

The proof of the theorem is given in the Appendix.

By this theorem, we can say that the only difference between FF-SM semantics and XSM semantics is the fact that, to check the stability condition, in FF-SM semantics the Kripke/Kleene semantics of the transformed program, $T_{P/I}^\omega$, is taken, while in XSM semantics the least fixed point of $T_{P/I}$ with respect to the $\leq_t$ ordering, i.e., $T_{P/I} \uparrow \omega$, is used.

While XSM semantics assigns the truth-value $false$ to the propositions involved in positive loops, and hence captures a notion of infinite failure, FF-SM semantics does not. For instance, the program $P = \{a \leftarrow a\}$ has a unique XSM in which $a$ is false, while it has a unique FF-SM in which $a$ is undefined. This behavior of FF-SM semantics is borrowed from the Kripke/Kleene semantics in [9, 18] on which it is based; it captures finite failure and it makes use of the value $undefined$ to model the fact that, operationally, a query may loop. For this reason, this semantics fits our abductive procedure which is an extension of SLDNF procedure. On the other hand, a top-down procedure to compute XSM semantics has been proposed in [22]. This procedure has some similarities with Eshghi and Kowalski's proof procedure, but it makes explicit use of loop-checking to deal with infinite failure. Also relying on a loop-checking mechanism is the SLX top-down procedure, recently proposed in [2, 1] to compute the Well-Founded Semantics.
We cannot argue that our semantics is better than XSM semantics as regards expressiveness. However, we think that the fact that it models a very natural extension of SLDNF makes it meaningful. This adherence to the proof procedure makes the semantics rather intuitive too.

As we have seen, from the implementation point of view, loop checking to detect positive loops (like \( a \leftarrow a \)) is not required, although negative loops are captured by means of the abductive assumptions. Also, FF-SM semantics has some properties quite similar to XSM semantics, like, for instance, the existence of a least (with respect to \( \leq_k \)) FF-Stable model, which can be computed in linear time (i.e., the Kripke/Kleene semantics of the program, \( T^p_k \)). Moreover, it is easy to see that determining the existence of an FF-Stable model in which a given goal is true is a problem NP-complete.

To further clarify the relationship between XSMs and FF-SMs, in the next section, we will provide a bilattice-based reformulation of FF-SM semantics, similar to the one given in [10, 11] for the Stable Model Semantics [12] and for the Well-Founded Semantics [30].

6. BILATTICE-BASED CHARACTERIZATION

In this section, we show how the FF-SM Semantics defined in Section 3 can be reformulated within the bilattice framework presented in [11].

A bilattice [13] is a structure with two partial orderings which are complete lattices. We consider bilattices in which points are truth-values, and the partial orderings defined on them represent their "degree of truth" and "degree of knowledge," respectively.

**Definition 6.1.** A pre-bilattice is a structure \((B, \leq_t, \leq_k)\) where \(B\) is a nonempty set (of truth-values), and \(\leq_t\) and \(\leq_k\) are partial orderings giving \(B\) the structure of a complete lattice.

Meet and join operators, least and greatest elements are defined on the truth-values belonging to \(B\) with respect to both the ordering relations. Meet and join operators under \(\leq_t\) are denoted by \(\land\) and \(\lor\), while under \(\leq_k\), they are denoted by \(\otimes\) and \(\oplus\); \(\text{false}\) and \(\text{true}\) are the least and the greatest elements under \(\leq_t\), while \(\bot\) and \(\top\) are the least and the greatest elements under \(\leq_k\).

**Definition 6.2.** A distributive bilattice is a pre-bilattice in which all distributive laws hold.

**Definition 6.3.** A pre-bilattice \((B, \leq_t, \leq_k)\) has a negation if there is a mapping \(\neg\) such that:

1. \(x \leq_t y \Rightarrow \neg y \leq_t \neg x\)
2. \(x \leq_k y \Rightarrow \neg x \leq_k \neg y\)
3. \(\neg \neg x = x\).

Belnap's four-valued logic is an example of a distributive bilattice with negation, which uses the orderings in Figure 7.
FIGURE 7. Belnap's four-valued logic.

The truth-value \( T \) represents inconsistency. The truth-value \( \bot \) corresponds to the truth-value *undefined* we have used throughout this paper. Notice that the ordering relations \( \leq_t \) and \( \leq_k \) on the truth-values *false*, *true*, and \( \bot \) are those introduced in Section 2.

**Definition 6.4.** A *B-valuation* on a distributive bilattice with negation is a mapping from ground atoms to members of the bilattice, mapping the atom *false* to the truth-value *false*.

As usual, orderings on truth-values can be straightforwardly extended to orderings on interpretations. Before showing how the FF-SM semantics fits into the bilattice framework, we recall the bilattice-based formulation of XSM semantics as given in [11].

In order to express the fact that the Stable Model semantics distinguishes, in a given program \( P \), the role of positive and negative information, the immediate consequence operator is generalized to accept two \( B \)-valuations as input, assigning meanings to positive and to negative atoms, respectively.

A notion of *B-pseudo-valuation* is introduced, which is a mapping from ground literals to truth-values in the distributive bilattice \( B \). The value assigned to a negative literal \( \neg A \) by a *B-pseudo-valuation* can be independent of the value assigned to its complement.

**Definition 6.5.** Given two *B-valuations* \( v_1 \) and \( v_2 \), the *B-pseudo-valuation* \( v_1 \triangle v_2 \) is defined as follows.

\[
(v_1 \triangle v_2)(A) = v_1(A)
\]

\[
(v_1 \triangle v_2)(\neg A) = \neg v_2(A).
\]

In \( v_1 \triangle v_2 \), \( v_1 \) and \( v_2 \) supply the positive and the negative information, respectively. Given two *B-valuations* \( v_1 \) and \( v_2 \), the *B-pseudo-valuation* \( v_1 \triangle v_2 \) can be naturally extended to conjunctions of literals.

Based on the above definition of *B-pseudo-valuation*, a new operator \( \Psi_p \), splitting apart positive and negative information, is defined.

---

\(^3\) A negative atom is the atom appearing in a negative literal.
Definition 6.6. Given a logic program $P$ and two $B$-valuations $v_1$ and $v_2$, $\Psi_p(v_1, v_2)$ is the $B$-valuation such that for each ground atom $A$

$$\Psi_p(v_1, v_2)(A) = \bigvee \{(v_1 \cup v_2)(\land Body) | A \leftarrow Body \text{ is in } P\}.$$ 

When $v_1$ and $v_2$ are taken as the same $B$-valuation $v$, $\Phi_p(v) = \Psi_p(v, v)$ is precisely the operator which generalizes the definition of the Fitting/Kunen operator $T_p$ to valuations acting on any distributive bilattice with negation (see [10]). In particular, when $B$ is the Belnap logic, $\Phi_p$ restricted to consistent valuations coincides with the operator $T_p$.

The operator $\Psi_p(v_1, v_2)$ is monotonic in both the input $v_1$ and the input $v_2$ under the ordering $\leq_k$, while under the ordering $\leq_t$, it is monotonic in $v_1$ and antimonotonic in $v_2$.

The valuation $v_2$ can be fixed, thus obtaining a monotonic operator (under $\leq_t$) in $v_1$, having (due to its monotonicity) a smallest fixed point. Based on this operator, a derived operator, also called stability operator in [29], is defined as follows.

Definition 6.7. [10]. The derived operator of $\Psi_p$ is the single input mapping $\Psi_p'$, given by: $\Psi_p'(v)$ is the smallest fixed point, in the $\leq_t$ ordering, of the mapping $(\lambda x)\Psi_p(x, v)$.

Under the ordering $\leq_t$, the derived operator $\Psi_p'$ is antimonotonic, while under the ordering $\leq_k$, the derived operator $\Psi_p$ is monotonic. Hence, $\Psi_p'$ has fixed points. As shown in [11], if $B$ is the Belnap four-valued logic [3], the fixed points of $\Psi'$ are the partial stable models or XSMs of $P$ [24]; the two-valued ones, if any, are its (classical) Stable Models [12].

The results in the previous section make clear the similarities and the differences between XSM semantics and FF-SM semantics. Given the above bilattice-based characterization of XSM semantics, we can now define a similar bilattice-based characterization for the FF-SM semantics. Like the Stable Model semantics, the XSM semantics and the Well-Founded semantics, the Finite Failure Stable Model semantics also distinguishes the role of positive and negative information. Such a distinction is expressed by means of the program transformation. As proved above, the definition of FF-SM semantics can be restated so that the transformation applied to the program is precisely the one used in the definition of the XSM semantics [24] (see the previous section). Moreover, the operator $T_p$, used in the definition of the XSM semantics and of the FF-SM semantics is the same. So the only difference between our semantics and the XSM semantics is that, in the definition of the FF-SM semantics, the least fixed point of $T_p$ is computed with respect to the $\leq_k$ ordering, while, in the definition of the XSM semantics, it is computed with respect to the $\leq_t$ ordering.

The bilattice-based characterization of FF-SM semantics is defined by making use of the operator $\Psi_p$, defined above. The valuation function $\Psi_p(v_1, v_2)$ is monotonic under the ordering $\leq_k$ in both inputs $v_1$ and $v_2$. Again, we can fix $v_2$, thus obtaining a monotonic operator (under $\leq_k$) in its input $v_1$. A new derived operator of $\Psi_p(v_1, v_2)$, $\Psi_p''$, is defined as follows.
Definition 6.8. $\Psi^p(v)$ is the smallest fixed point under the ordering $\leq_k$ of the single input mapping $\lambda x \Psi_p(x, v)$.

$\Psi^p(v)$ is monotonic under the ordering $\leq_k$ in its input $v$, and hence, it has fixed points. If $B$ is the Belnap logic [3], the following property holds:

**Proposition 6.1.** Given a program $P$ and a valuation function $v$, $v$ is an FF-SM of $P$ iff it is a fixed point of the derived operator $\Psi^p$, i.e., iff $v = \Psi^p(v)$.

This formulation makes even more explicit the fact that the only significant difference between the eXtended Stable Model Semantics [24] and the Finite Failure Stable Model semantics is the ordering relation with respect to which the least fixed point of the immediate consequence operator is computed. Indeed, in the bilattice-based characterizations, the two semantics only differ in the definition of the derived operators $\Psi^p$ and $\Psi^p_\tau$: while the first one is obtained by taking the least fixed point of $\lambda x \Psi_p(x, v)$ with respect to the $\leq_\tau$ ordering, the second one is obtained by taking the least fixed point of the same $\lambda x \Psi_p(x, v)$ with respect to the $\leq_k$ ordering.

Theorem 5.1 guarantees that when, in the definition of the FF-SM Semantics, a given program $P$ is transformed with respect to a three-valued interpretation $I$, the evaluation of the negative literals appearing in the body of the clauses in $P$ is given by the interpretation $I$ itself. In fact, the transformation of $P$ into $P/I$ [24] completely eliminates the negative literals from the program, and this corresponds to keeping, for each negative literal, its truth-value fixed by the interpretation $I$. In the reformulation above, the role of the interpretation $I$ is taken by the valuation $v$ (the argument of $\Psi^p$), which is fixed to get the single input mapping $\lambda x \Psi_p(x, v)$.

7. CONCLUSIONS

In this paper, we have proposed a three-valued semantics for logic programs with negation, the Finite Failure Stable Model semantics, and an abductive procedure, both sound and complete with respect to it, which is a slight variant of Eshghi and Kowalski’s abductive procedure [8]. Some of the results presented in this paper were already presented in the shorter paper [14]. The FF-SM semantics is a generalization of the Stable Model semantics of Gelfond and Lifschitz [12] which, like the Extended Stable Model (XSM) semantics [24], assigns a meaning to each logic program (while there are programs having no stable model).

FF-Stable Model semantics is defined building on a Kripke/Kleene semantics: the given program is first transformed with respect to a given three-valued interpretation $I$ (giving a program which still possibly contains negative literals), then the equality of $I$ and the Kripke/Kleene fixed-point semantics of the transformed program is checked. A comparison of FF-SM semantics and XSM semantics shows that, in the definition of FF-SM semantics, the transformation applied to a program $P$ (given a three-valued interpretation $I$) can be equivalently replaced by the extended GL-transformation, $P/I$, proposed in [24] to define XSMs. Both the FF-SM semantics and the XSM semantics can be defined by taking a smallest fixed point of the same immediate consequences operator $T_p/I$ associated with the transformed program $P/I$, and checking whether $I$ coincides
with that smallest fixed point. However, the smallest fixed point is computed in the two cases with respect to two different ordering relations: $\preceq_k$ for the FF-SM semantics (i.e., the Kripke/Kleene semantics is used), while $\preceq_t$ for the XSM semantics. Hence, as a difference with XSM semantics, FF-SM semantics does not capture infinite failure: it models the loops causing nonterminating computations by means of the truth-value undefined. The relationship between XSMs and FF-SMs has also been shown by defining a bilattice-based characterization [11] for FF-SM semantics and by comparing it to the one given by Fitting for XSM semantics.

The FF-SM semantics can be naturally related to the Preferential Semantics [6] by means of the equivalence results presented in [4], where it is proved that there exists a one-to-one correspondence between Dung’s complete scenarios and eXtended Stable Models [25] of a given program $P$. Hence, since preferred extensions are maximal (with respect to set inclusion) complete scenarios, each preferred extension has a corresponding eXtended Stable Model. The relation between complete scenarios and FF-SMs is therefore the same as the one between XSMs and FF-SMs.

The FF-SM semantics and the abductive procedure we have defined and discussed in the paper only deal with propositional logic programs. However, since the abductive procedure is an extension of SLDNF, it seems quite natural to extend it to the first-order case.

As usual, in the first-order case, to avoid the selection of nonground negative literals, a safeness restriction can be put on the selection rule $R$. In this way, the set of assumptions $\Delta$ returned by the abductive procedure may contain ground atoms only. Since nonground goals can be evaluated by the (extended) abductive procedure, a notion of answer substitution must be defined, and must be included (in the obvious way) in the definition of the abductive procedure. Hence, when a success derivation is found for a nonground goal, both an answer substitution $\sigma$ (for the variables appearing in the goal) and a set of (ground) negative assumptions $\Delta$ supporting the success are returned (obviously, no answer substitution is returned by the finitely failing derivations, but only a set of assumptions $\Delta$).

The definition of the FF-SM semantics could be extended to the first-order case, by making use of the ground instantiation $\overline{P}$ of the program, as follows: given a first-order program $P$ and an interpretation $I = \langle T, F \rangle$, $I$ is a Finite Failure Stable Model of $P$ iff $I = T_{\overline{P}}^\omega$. Notice that $\overline{P}$ (and hence $\overline{P}_f$) is a possibly infinite propositional program, and the iteration at $\omega$ of $T_{\overline{P}}$ does not necessarily coincide with the least fixed point of $T_{\overline{P}}$.

We argue that soundness and completeness of the abductive procedure with respect to the FF-SM semantics can be proved also in the first-order case, under an allowedness condition on programs and goals (see [19]). This condition will be surely needed to get completeness, since the abductive procedure inherits from SLDNF the problems related to the existence of floundering computations.

**APPENDIX**

We give the proofs of the theorems stated in the paper. First, we prove soundness and completeness of the abductive procedure with respect to the FF-SM semantics (Theorem 4.1 and Theorem 4.2 stated in Section 4). The proof of these results
relies on the soundness and completeness of SLDNF-resolution w.r.t. the Kripke/Kleene semantics. In particular, we make use of the result by Kunen recalled in Section 2. Afterwards, we prove Theorem 5.1.

Before moving to the proof of the theorems, we introduce some definitions and properties of the abductive procedure, which will be referred to in the following proofs.

**Definition A.1.** [Inner Subderivation] Given a program $P$, a goal $G$, a set of assumptions $\Delta$, and a computation rule $R$, let $(G, \Delta_0), (G_1, \Delta_1), \ldots$, where $\Delta_0 = \Delta$, be a possibly infinite sequence of pairs such that, for $i > 0$, $(G_{i+1}, \Delta_{i+1})$ is obtained from $(G_i, \Delta_i)$ by means of a step of the abductive success derivation procedure. For any $i > 0$, if the rule applied to derive $(G_{i+1}, \Delta_{i+1})$ from $(G_i, \Delta_i)$ is $(R \sim A)$, then the literal in $G\sim$ selected by $R$ is a negative one, say $\sim A$, such that there exists a finitely failing derivation for $({A}, \Delta_i \cup \{ \sim A \})$ via $R$. The pair $({A}, \Delta_i \cup \{ \sim A \})$ is called root of an inner finitely failing subderivation.

The root of an inner success subderivation is defined analogously, by considering the steps of a finitely failing derivation in which the rule $(F \sim A)$ is applied.

The following properties formalize some features of the computed answers returned for a given goal by the success and finitely failing abductive derivations.

**Property A.1.** Given a program $P$, a goal $G$, and a set of assumptions $\Delta_0$ such that $(G, \Delta_0)$ has an abductive success derivation (respectively, $({G}, \Delta_0)$ has a finitely failing derivation) from $P$ with computed answer $\Delta$, let $(G', \Delta')$ (respectively, $({G'}, \Delta')$) be the root of an inner subderivation.

(a) If there exists an abductive success derivation for $(G', \Delta')$ with computed answer $\Delta_1$, then there exists an abductive success derivation for $(G', \Delta)$ from $P$ with computed answer $\Delta$;

(b) If there exists an abductive finitely failing derivation for $({G'}, \Delta')$ with computed answer $\Delta_f$, then there exists an abductive finitely failing derivation for $({G'}, \Delta)$ from $P$ with computed answer $\Delta$.

**Property A.2.** Given a program $P$, a goal $G$, and a set of negative assumptions $\Delta_0$, if there exist both a success derivation for $(G, \Delta_0)$ with computed answer $\Delta_s$ and a finitely failing derivation for $({G}, \Delta_0)$ with computed answer $\Delta_f$, then $\Delta_s \neq \Delta_f$.

**Property A.3.** Given a program $P$, a goal $G$, and a set of assumptions $\Delta$, if there exists a success (finitely failing) derivation for $(G, \Delta)$ (respectively, $({G}, \Delta)$) with computed answer $\Delta''$, then for each assumption $\sim A \in (\Delta'' - \Delta)$, there exists a set of assumptions $\Delta'$ such that $\Delta \subseteq \Delta' \subseteq \Delta''$ and $({A}, \Delta')$ is the root of an inner abductive finitely failing derivation (i.e., the rule $(R \sim A)$ has been applied to introduce $\sim A$ in $\Delta''$).

In the proofs of the soundness and completeness results, we make use of the following notation.

Given $P$ and $I = \langle T, F \rangle$ let $\Delta_I = \sim F$, where $\sim F = \{ \sim b | b \in F \}$. To make evident the fact that only the negative component of a given interpretation $I = \langle T, F \rangle$ plays a role in the program transformation $P_I$ (defined in Section 3), in the following proofs, we will equivalently denote $P_I$ with $P_{\Delta_I}$.
From the definition of the program transformation, it immediately follows that, for a given \( \Delta \), there exists a correspondence between the clauses of \( P \) and those of \( P_\Delta \); in particular, each clause
\[
c : A \leftarrow B_1, \ldots, B_m \quad (m \geq 0)
\]
in \( P \) has a corresponding
\[
c_\Delta : A \leftarrow B_{1 \Delta}, \ldots, B_{m \Delta} \quad (m \geq 0) \text{ in } P_\Delta,
\]
where
\[
\begin{align*}
B_{i \Delta} &= \text{true} & \text{if } B_i = \neg B \text{ and } \neg B \in \Delta, \text{i.e., } B_i \in \Delta \\
B_{i \Delta} &= B_i & \text{otherwise.}
\end{align*}
\]
In the first case (\( B_{i \Delta} = \text{true} \)), \( B_{i \Delta} \) is not explicitly mentioned in the clause body. For this reason, it can be the case that two different clauses, \( c \) and \( c' \), which only differ as regards negative literals in their bodies, have the same corresponding clause in \( P_\Delta \), i.e., \( c_{\Delta} = c'_{\Delta} \). As a consequence: each clause of \( P \) has a corresponding clause in \( P_\Delta \), and each clause in \( P_\Delta \) has one or more corresponding clauses in \( P \). The positive literals appearing in the body of \( c \) are exactly those which are present in the body of \( c_{\Delta} \).

In the following, given a normal goal \( G \) and a set of negative literals \( \Delta \), we will also denote by \( G_\Delta \), the goal obtained from \( G \) by removing from it all the negative literals occurring in \( \Delta \).

\section{A.1. Proof of Theorem 4.1}
We have to prove that the abductive proof procedure is \textbf{sound} with respect to the FF-Stable Model semantics, i.e., given a program \( P \) and a goal \( G \),

- if \((G, \emptyset)\) succeeds from \( P \) with computed answer \( \Delta \) according to the abductive procedure, then there exists an \textbf{FF-Stable Model} \( I \) for \( P \) such that \( I(G) = \text{true} \) and \( \Delta \subseteq \Delta \);
- if \( ((G), \emptyset) \) finitely fails from \( P \) with computed answer \( \Delta \) according to the abductive procedure, then there exists an \textbf{FF-Stable Model} \( I \) for \( P \) such that \( I(G) = \text{false} \), and \( \Delta \subseteq \Delta \).

The proof of the soundness theorem makes use of the following lemmas and corollaries. Lemma A.1 and Corollary A.1 give some properties relating negative assumptions, least fixed points of the \( T_p \) associated with the transformed programs, and FF-Stable Models.

\textbf{Lemma A.1.} Given a program \( P \) and an interpretation \( I = (T, F) \), let \( b \) be a positive literal in the language of \( P \), and let \( \Delta = \neg F \). If \( T_{\Delta}^\omega(b) = \text{false} \), then for each atom \( A \) of the language of \( P \), \( T_{\Delta}^\omega(A) = T_{\Delta \cup \{ \neg b \}}^\omega(A) \).

\textbf{Proof.} We prove by induction on \( n \) that, for all \( n \geq 0 \),

\begin{enumerate}
  \item \( T_{\Delta}^n \subseteq T_{\Delta \cup \{ \neg b \}}^n \)
  \item \( T_{\Delta \cup \{ \neg b \}}^n \subseteq T_{P_3}^\omega \)
\end{enumerate}

\textit{Base of induction.} \( T_{\Delta}^0 = T_{P_3 \cup \{ \neg b \}}^0 = (\emptyset, \emptyset) \). Hence, 1 and 2 hold trivially.
Inductive step. By inductive hypothesis, we have that:

1. $T^n_{P_a} \subseteq T^n_{P_a \cup \{\sim b\}}$
2. $T^n_{P_a \cup \{\sim b\}} \subseteq T^n_{P_a}$

We have to prove that:

1'. $T^{n+1}_{P_a} \subseteq T^{n+1}_{P_a \cup \{\sim b\}}$
2'. $T^{n+1}_{P_a \cup \{\sim b\}} \subseteq T^{n+1}_{P_a}$

1'. Consider a generic atom $A$ in the language of $P$. If $T^{n+1}_n(A) = true$, then there exists in $P_\Delta$ a clause $A \leftarrow B_1, \ldots, B_m, (m \geq 0)$, such that $T^n_n(B_i) = true$, $\forall i < m$.

The literals occurring in the body of the corresponding clause in $P_\Delta \cup \{\sim b\}$ are a subset of $\{B_1, \ldots, B_m\}$; each of them satisfies the inductive hypothesis, from which $T^{n+1}_n(A) = true$ follows.

If $T^{n+1}_n(A) = false$, and $A$ is not defined in $P_\Delta$, then $A$ is not defined in $P_\Delta \cup \{\sim b\}$ either; therefore, $T^{n+1}_n(A) = false$. Otherwise, $A$ is defined in $P_\Delta$, but the bodies of all clauses defining it are false $in T^n_n$. Let us consider a generic one.

$A \leftarrow B_1, \ldots, B_m$ \hspace{1cm} ($m > 0$).

Its body contains at least one literal $B_j$ such that $T^n_n(B_j) = false$.

We have that $B_j \neq \sim b$. In fact, if $B_j = \sim b$ and $T^n_n(\sim b) = false$, then $T^n_n(b) = true$, and hence, by the monotonicity of $T^n_n$, $T^n_n(b) = true$. But this contradicts the hypothesis that $T^n_n(b) = false$.

Thus, the same literal $B_j$ also appears in the body of the corresponding clause in $P_\Delta \cup \{\sim b\}$. This body turns out to be false $in T^n_n$, since, by the inductive hypothesis, $T^n_n(B_j) = false$.

Since the same argument applies to each clause defining $A$ in $P_\Delta$, we conclude $T^{n+1}_n(A) = false$.

2'. If $T^{n+1}_n(A) = true$, then there exists in $P_\Delta \cup \{\sim b\}$ a clause $A \leftarrow B_1, \ldots, B_m, m \geq 0$ such that $T^n_n(B_i) = true, \forall i \leq m$.

This clause has in $P_\Delta$ at least one corresponding clause, whose body consists of all the literals $B_1, \ldots, B_m$ and possibly some occurrences of the literal $\sim b$.

By the inductive hypothesis $T^n_n(B_j) = true, \forall i \leq m$. Moreover, $T^n_n(\sim b) = true$, since $T^n_n(b) = false$ by hypothesis. Therefore, $T^n_n(A) = true$.

If $T^{n+1}_n(A) = false$ and $A$ is not defined in $P_\Delta \cup \{\sim b\}$, then $A$ is not defined in $P_\Delta$ either; therefore, $T^n_n(A) = false$. Otherwise, if $T^{n+1}_n(A) = false$ and $A$ is defined in $P_\Delta \cup \{\sim b\}$, then each clause defining $A$ in $P_\Delta \cup \{\sim b\}$ contains in its body at least one literal $B_j$ false $in T^n_n$.

This literal is different from $\sim b$ by definition of the program transformation $\langle \sim \Delta \cup \{\sim b\} \rangle$. Therefore, $B_j$ also appears in the body of the corresponding clause in $P_\Delta$. By inductive hypothesis, $T^n_n(B_j) = false$. Since this holds for all clauses defining $A$, we have that $T^n_n(A) = false$. \]

Corollary A.1. Given a program $P$ and a set $F$ of positive literals in the language of $P$, let $\Delta = \sim F$. If $I = \langle T', F' \rangle = T^n_n$ and $F \subseteq F'$, then $I = T^n_n$ (that is, $I$ is an FF-SM of $P$).
PROOF. We define $F'' = \{b | b \notin F$ and $T_{P_a}(b) = \text{false}\}$, i.e., $F'' = F' - F$. Let $F'' = \{b_1, \ldots, b_k\}$. $F''$ is finite, since the program $P$ is finite propositional. Each atom $b_i \in F''$, $i = 1, \ldots, k$, satisfies the hypothesis of Lemma A.1: $T_{P_a}(b_i) = \text{false}$.

Therefore, each atom in $F''$ also satisfies the hypothesis of Lemma A.1, w.r.t. $P \cup \{\neg b_i\}^k$.

We can repeat $k$ times the same argument, i.e., we can apply $k$ times Lemma A.1, once for each $b_i$, thus obtaining $T_{P_a \cup \{\neg b_i\}} = T_{P_a \cup \{\neg b_i\}}$, $i = 1, \ldots, k$. It follows that $T_{P_a \cup \{\neg b_i\}} = I = \langle T', F' \rangle$, where $T' = \{b | T_{P_a} \models b\}$ and $F' = F \cup F''$. Hence, by definition, $I = \langle T', F' \rangle$, is an FF-Stable Model of $P$. □

The following is the key lemma in the proof of soundness.

**Lemma A.2.** Given a program $P$ and a goal $G$ such that $(G, \emptyset)$ has an abductive success (or $(\{G\}, \emptyset)$ has a finitely failing) derivation with computed answer $\Delta$,

(a) If $(G', \Delta')$ is the root of an inner subderivation of the derivation for $(G, \emptyset)$ (or $(\{G\}, \emptyset)$), and there exists an abductive success derivation for $(G', \Delta')$ from $P$, then $G'$ succeeds from $P_a$ via SLDNF;

(b) If $(\{G'\}, \Delta')$ is the root of an inner subderivation of the derivation for $(G, \emptyset)$ (or $(\{G\}, \emptyset)$), and there exists an abductive finitely failing derivation for $(\{G'\}, \Delta')$ from $P$, then $G'$ finitely fails from $P_a$ via SLDNF.

**PROOF.** In the proof of Lemma A.2, we make use of a notion of “SLDNF-pseudo-derivation,” defined as an SLDNF derivation in which “null steps,” resulting in the occurrence of contiguous identical goals $G$ (or sets of goals $F$), are admitted.

We prove that, for a given success (finitely failing) abductive derivation, a corresponding success (finitely failing) SLDNF-pseudo-derivation can be constructed.

The proof is by induction on the rank $m$ of subderivations, that is, on their nesting level, starting from the innermost ones.

- Base of induction $(m = 0)$

The abductive subderivation for $(G', \Delta')$ from $P$ does not contain any nested subderivation.

(a) Let $(G_1, \Delta_1), \ldots, (G_n, \Delta_n)$, $G_1 = G'$, $\Delta_1 = \Delta'$, $G_n = \emptyset$ be the given abductive subderivation.

Let $G_{k}^{\Delta} = (G_{k})_{\Delta}$ for all $k = 1, \ldots, n$. We show that $G_{1}^{\Delta}, \ldots, G_{n}^{\Delta}$ is an SLDNF-pseudo-derivation for $G_1$. Note that $G_{1}^{\Delta} = G_1$ (being root of a subderivation, $G_1$ is a positive literal), and $G_{n}^{\Delta} = \emptyset$. We have to prove that $\forall k = 1, \ldots, n - 1$ either $G_{k}^{\Delta} = G_{k+1}^{\Delta}$ or $G_{k+1}^{\Delta}$ is obtained from $G_{k}^{\Delta}$ by an SLDNF rule.

Consider the step $(G_{k}, \Delta_{k}), (G_{k+1}, \Delta_{k+1})$, $(1 \leq k < n)$ in the given abductive success subderivation.

Let $G_{k} = L_{1}, \ldots, L_{i-1}, L_{i}, L_{i+1}, \ldots, L_{k}$, and let $L_{i}$ be the selected literal. One of the following cases necessarily happens:

(RA) If $L_{i} = A$, the considered step is a resolution step, resolving $A$ on a clause $c_{A} \in P$, i.e., $G_{k+1} = \text{resolvent}(G_{k}, A, c_{A})$. 


Since $A \in G_k$ and $G_k^\Delta = (G_k)_\Delta$, $A \in G_k^\Delta$.

Given $G_{k+1}^\Delta = (G_{k+1})_\Delta$, it is quite obvious that $G_{k+1}^\Delta = \text{resolvent}(G_k^\Delta, A, \{c_A\} \Delta)$. Hence, $G_{k+1}^\Delta$ is obtained from $G_k^\Delta$ by rule $(RA_1^\Delta)$.

$(R \sim A_1)$ If $L_i = \sim A$ and $\sim A \in \Delta_k$, then $G_{k+1} = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$ and $\Delta_{k+1} = \Delta_k$.

Since $\sim A \not\in G_k^\Delta$, then $G_{k+1}^\Delta = G_k^\Delta$, i.e., a null step.

$(R \sim A_2)$ The rule $(R \sim A_2)$ is never applied since rank $m = 0$.

(b) Let $(F_1, \Delta_1), \ldots, (F_h, \Delta_h)$, $F_1 = \{G'\}$, $F_h = \emptyset$ be the given abductive finitely failing derivation.

As in case (a), we build a corresponding finitely failing SLDNF-pseudo-derivation. The transformation function $(\cdot)_\Delta$ is naturally extended to sets of goals. Let $F_k^\Delta = (F_k)_\Delta$ for all $k = 1, \ldots, h$.

Since $F_1 = \{G'\}$ and $G'$ is an atom (it is the root of an abductive finitely failing derivation), we have $F_1^\Delta = F_1$.

We show that $F_1^\Delta, \ldots, F_h^\Delta$ is a finitely failing SLDNF-pseudo-derivation. Note that $F_h^\Delta = (F_h)_\Delta = \emptyset$.

At step $k$, let $G_k = L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n$ be the chosen goal in $F_k$, and let $L_i$ be the selected literal in $G_k$. One of the following cases happens:

$(FA_1)$ $L_i = A$. $G'_k$ is replaced in $F_k$ by all its resolvents on $A$ with clauses defining $A$ in $P$, so to get the new set of goals $F_{k+1}^\Delta = (F_k \setminus G'_k) \cup \{\text{resolvent}(G'_k, A, \{c_A\}_\Delta)\}$, where $C_A \subseteq P$ is the set of clauses defining $A$ in $P$.

Let $C_A = (C_A)_\Delta$.

It is quite obvious that $F_{k+1}^\Delta = (F_{k+1})_\Delta = (F_k^\Delta \setminus G_k^\Delta) \cup \{\text{resolvent}(G_k^\Delta, A, \{c_A\}_\Delta)\}$. Hence, $F_{k+1}^\Delta$ is obtained from $F_k^\Delta$ by rule $(FA_1^\Delta)$.

$(FA_2)$ $L_i = A$ and $A$ is not defined in $P$. In this case, $A$ is not defined in $P_\Delta$ as well, and the abductive derivation step $(F_k, \Delta_k)(F_k \setminus \{G_k\}_\Delta, \Delta_{k+1})$, has a corresponding SLDNF step $F_k^\Delta, F_k^\Delta \setminus \{G_k^\Delta\}$.

Indeed, if $G_k^\Delta = (G_k')_\Delta$, $F_{k+1}^\Delta = (F_{k+1})_\Delta = F_k^\Delta \setminus \{G_k^\Delta\}$.

$(F \sim A_1)$ The rule $(F \sim A_1)$ is never applied, since $m = 0$.

$(F \sim A_2)$ $L_i = \sim A$, $\sim A \in \Delta_k$; therefore, $\sim A \in \Delta$, and $F_{k+1}$ is obtained from $F_k$ by removing $\sim A$ from $G_k'$.

Since $\sim A \not\in G_k^\Delta = (G_k')_\Delta$, $F_{k+1}^\Delta = F_k^\Delta$, a null step.

- Inductive step ($m > 0$)

By inductive hypothesis, we have that:

---for each pair $(G', \Delta')$ which is a root of an abductive success subderivation of rank $j$, $0 \leq j < m$, from $P$, there is a corresponding SLDNF success pseudo-derivation for $G'$ in $P_\Delta$;

---{resolvent}(G_k^\Delta, A, \{c_A\}_\Delta) is the extension to the set $C_A^\Delta$ of the notion of resolvent.
—for each pair \((G', \Delta')\) which is a root for an abductive finitely failing subderivation of rank \(j\), \(0 \leq j < m\), from \(P\), there is a corresponding SLDNF finitely failing pseudo-derivation for \(G'\) in \(P_\Delta\).

We want to prove the thesis for rank \(m\).

The proof is analogous to the base of induction, as far as it concerns the steps there defined. Some additional cases have to be taken into account: \((R \sim A_2)\) and \((F \sim A_1)\).

\((R \sim A_2)\) If \(L_i = \sim A\), and there exists an abductive finitely failing derivation for \(\langle \{A\}, \Delta_k \cup \{\sim A\} \rangle\), of rank \((m-1)\), \(G_{k+1} = G_k \setminus \{L_i\}\) and \(\sim A \in \Delta\) (from the monotonicity of the sequence \(\Delta_1, \Delta_2, \ldots\)).

Since \(G_{k}^\Delta = (G_k)_{\Delta} \setminus \sim A \in \Delta\); therefore, \(G_{k+1}^\Delta = (G_{k+1})_{\Delta} = G_k^\Delta\) (this is a null step in the SLDNF pseudo-derivation).

\((F \sim A_1)\) If \(L_i = \sim A\) and there exists an abductive success derivation, of rank \((m-1)\), for \((A, \Delta_k)\), then \(F_{k+1} = F_k \setminus \{G_k, A\}\), and it must be that \(\sim A \not\in \Delta\).^5

Therefore, \(\sim A \in G_{k+1}^\Delta = (G_{k+1})_{\Delta}\)

By inductive hypothesis, there exists an SLDNF success pseudo-derivation for \(A\) in \(P_\Delta\). Hence, \(F_{k+1}^\Delta = (F_{k+1})_\Delta = F_k^\Delta \setminus \{G_k, A\}\) is obtained from \(F_k^\Delta\) by rule \((F \sim A_1^\Delta)\).

We have shown that, from a given abductive derivation, a corresponding SLDNF pseudo-derivation can be defined. From it, an SLDNF derivation can be obtained by eliminating null steps. For instance, the SLDNF success derivation corresponding to the considered abductive success derivation \((G_1, \Delta_1), \ldots, (G_n, \Delta_n)\) can be obtained by removing from the sequence \(G_1^\Delta, \ldots, G_n^\Delta\) each goal identical to the previous one. The “pseudo-derivation” ends in the empty clause; thus, the derivation extracted from it is a successful one. Similarly, an SLDNF finitely failing derivation can be obtained by removing contiguous repetitions from the finitely failing pseudo-derivation \(F_1^\Delta, \ldots, F_n^\Delta\).

From the above Lemma A.2, we can easily derive the following corollary.

**Corollary A.2.** Given a program \(P\) and a goal \(G\)

- if \((G, \emptyset)\) has an abductive success derivation from \(P\) with computed answer \(\Delta\), then \(G_\Delta\) succeeds via SLDNF from \(P_\Delta\);
- if \(((G), \emptyset)\) has an abductive finitely failing derivation from \(P\) with computed answer \(\Delta\), then \(\{G_\Delta\}\) finitely fails via SLDNF from \(P_\Delta\).

**Proof.** The proof is very similar to the one of Lemma A.2: both the case of success and the case of finite failure for the abductive derivation are considered, and it is shown that each step of the abductive (finitely failing) derivation for \(G\) (\(\langle G \rangle\)) from \(P\) has a corresponding SLDNF “pseudo-derivation” step for \(G_\Delta\) (\(\langle G_\Delta \rangle\)) from \(P_\Delta\); to conclude the proof, the pseudo-derivation is transformed into an SLDNF derivation.

---

^5This immediately follows from Properties A.1 and A.2
The main difference between the proof of Lemma A.2 and that of its Corollary A.2 is that the calls to the inductive hypothesis in Lemma A.2 are replaced by calls to Lemma A.2 in Corollary A.2.

**Lemma A.3.** Given a program $P$ and a goal $G$,

- if $(G, \emptyset)$ has an abductive success derivation from $P$ with computed answer $\Delta$ and $I = T_P^w$, then $I(G_\lambda) = \text{true}$;
- if $\{G\}, \emptyset$ has an abductive finitely failing derivation from $P$ with computed answer $\Delta$ and $I = T_P^w$, then $I(G_\lambda) = \text{false}$.

**PROOF.** The lemma follows immediately from Corollary A.2 and soundness of SLDNF with respect to the Kripke/Kleene semantics [18] (see Section 2).

By making use of Lemmas A.1, A.2, and A.3, we can prove Theorem 4.1.

**PROOF OF THEOREM 4.1.**

- If $(G, \emptyset)$ succeeds from $P$ with computed answer $\Delta$ according to the abductive procedure, let $I = (T, F) = T_P^w$. By Lemma A.3, $I(G_\lambda) = \text{true}$.

Let us first prove that $I = T_P^w$ is an FF-Stable model. Indeed, $\Delta$ is the abductive computed answer and, according to the definition of the abductive procedure, if $\sim b \in \Delta$, then there exists $\Delta' \subseteq \Delta$ such that $\{(b), \Delta'\}$ is the root of an abductive finitely failing subderivation with computed answer $\Delta_f \subseteq \Delta$ (Property A.3).

Lemma A.2 states that there exists an SLDNF finitely failing derivation for $(b)$ from $P_\lambda$, and therefore $I(b) = \text{false}$, by soundness of SLDNF resolution. It follows that $\Delta \subseteq \sim F$. This means that the hypothesis of Corollary A.1 is satisfied. Thus, $I$ is an FF-Stable model of $P$ and $I(G_\lambda) = \text{true}$.

Given that $\Delta \subseteq \sim F$, we can easily conclude that also $I(G) = \text{true}$. To show this, it suffices to observe that if $G$ contains a literal, say $L$, which does not belong to $G_\lambda$, then $L = \sim b \in \Delta$. Hence, $b \in F$ and $I(\sim b) = \text{true}$.

- The proof the finitely failing case is similar to the previous case.

**A.2. Proof of Theorem 4.2**

We have to prove that the abductive procedure is **complete** with respect to the FF-SM semantics, i.e., given a program $P$ and a goal $G$,

- if there exists an FF-Stable Model $I$ for $P$ such that $I(G) = \text{true}$, then $(G, \emptyset)$ succeeds from $P$ with computed answer $\Delta$ according to the abductive procedure, and $\Delta \subseteq \Delta_f$;
- if there exists an FF-Stable Model $I$ for $P$ such that $I(G) = \text{false}$, then $\{(G), \emptyset\}$ finitely fails from $P$ with computed answer $\Delta$ according to the abductive procedure, and $\Delta \subseteq \Delta_f$.

The proof of the completeness theorem is based on the following lemmas.

**Lemma A.4.** Given a program $P$ and an interpretation $I$, if $I$ is an FF-SM of $P$ and there exists an SLDNF success derivation for $G_\lambda$ from $P_\lambda$, then each goal in such a derivation does not contain any negative literal.
**PROOF.** Let $G_1, \ldots, G_n$ be the given SLDNF success derivation, where $G_1 = G_{\Delta_1}$, $G_n = \Box$.

We assume, by contradiction, that there exists a goal $G_k$, $1 \leq k < n$, such that

$$G_k = L_1, \ldots, L_m, \text{ and } \exists L_i = \sim b, \quad 1 \leq i \leq m.$$  

Each literal occurring in $G_k$ either belongs to the initial goal $G_{\Delta_k}$, or results from a previous resolution step (rule $R_{A_T^*}$), resolving a positive literal with a clause in $P_{\Delta_k}$.

Since the given derivation is a success one, the last goal is the empty clause. This means that for each negative literal occurring in an intermediate goal, there exists a derivation step ($R_{A_T^*}$) which removes it from the current goal. In particular, there exists a $j$th step, $k < j < n$, in the given derivation in which the literal $L_i = \sim b$ is the selected one, and it is dropped because of the existence of an SLDNF finitely failing derivation for $\{b\}$ from $P_{\Delta_k}$.

By hypothesis, $I$ is an FF-SM of $P$, i.e., $I = T_{P_{\Delta_k}}$. Since there exists an SLDNF finitely failing derivation for $\{b\}$ from $P_{\Delta_k}$, $I(b) = \text{false}$ (by soundness of SLDNF), i.e., $b \not\in F$. This contradicts the hypothesis that $L_i$ belongs to the goal $G_k$: if $b \in F$, then $\sim b$ does not belong to $G_k$, nor to $P_{\Delta_k}$.

It follows that $\sim b$ does not belong to any intermediate goal of the given success derivation. \[
\]

**Lemma A.5.** Given a program $P$ and an FF-Stable Model $I$ of $P$, for each goal $G$,

1. if there exists an SLDNF success derivation for $G_{\Delta_k}$ from $P_{\Delta_k}$, then, for all $\Delta' \subseteq \Delta_k$, $(G, \Delta')$ has an abductive success derivation from $P$ with computed answer $\Delta'' \subseteq \Delta_k$;
2. if there exists an SLDNF finitely failing derivation for $\{G_{\Delta_k}\}$ from $P_{\Delta_k}$, then, for all $\Delta' \subseteq \Delta_k$, $\{(G), \Delta'\}$ has an abductive finitely failing derivation from $P$ with computed answer $\Delta'' \subseteq \Delta_k$.

**PROOF.** The proof is by induction on the cardinality of $\Delta'$: decreasing cardinalities are considered, starting from the biggest one.

- Base of induction ($\Delta' = \Delta_k$)

1. Let us assume that there exists an SLDNF success derivation for $G_{\Delta_k}$ from $P_{\Delta_k}$. Then there exists an abductive success derivation for $(G, \Delta_k)$ from $P$.

Let $G_{\Delta_k}^1, \ldots, G_{\Delta_k}^n$ be the SLDNF success derivation, via $R$, for $G_{\Delta_k}$ in $P_{\Delta_k}$.

By taking into account each of its steps, an abductive success derivation for $(G, \Delta_k)$ is built up.

For each step $G_{\Delta_k}^k, G_{\Delta_k}^{k+1}$ in the SLDNF derivation, a corresponding abductive derivation step $(G_k, \Delta_k)$, $(G_{k+1}, \Delta_{k+1})$ is defined, with $\Delta_{k+1} = \Delta_k = \Delta_k$ in such a way that, assuming $G_{\Delta_k}^k = (G_k)_{\Delta_k}$, we get $G_{\Delta_k}^{k+1} = (G_{k+1})_{\Delta_k}$.

We define $(G_1, \Delta_k) = (G, \Delta_k)$, and therefore we have $G_{\Delta_k}^1 = (G_1)_{\Delta_k}$. For each $k = 1, \ldots, n - 1$, $G_{\Delta_k}^{k+1}$ is obtained from $G_{\Delta_k}^k$ by a resolution step resolving a positive literal $L_i \in G_{\Delta_k}^k$ with a clause $c_{\Delta_i} \in P_{\Delta_k}$ (by Lemma A.4).
Let \( c \) be a clause in \( P \) such that \( c_{\Delta_i} = (c)_{\Delta_i} \). In the abductive derivation, we take \( G_{k+1} = \text{resolvent}(G_k, L_i, c) \). It is clear that, assuming \( G_k = (G_k)_{\Delta_i} \), we get \( G_{k+1} = (G_{k+1})_{\Delta_i} \). If \( G_n \) is the empty clause \( \square \), then

\[
(G_1, \Delta_i), \ldots, (G_n, \Delta_i)
\]
is a success abductive derivation for \((G, \Delta_i)\) from \( P \).

Otherwise, \( G_n = \sim b_1, \ldots, \sim b_k \), with each \( b_j \in \Delta_i \). In this case, we get an abductive derivation by applying rule \((R \sim A_i)\) \( k \) times, each time removing from the current goal a negative literal belonging to \( \Delta_i \), thus obtaining, at the end, the pair \((\square, \Delta_i)\).

2. Let us assume that \( F^\Delta_1, \ldots, F^\Delta_n \) is an SLDNF finitely failing derivation for \( \{G_{\Delta_i}\} \) from \( P_{\Delta_i} \). We prove that there exists an abductive finitely failing derivation for \((G, \Delta_i)\) from \( P \). An abductive finitely failing derivation \((F_1, \Delta_i), \ldots, (F_m, \Delta_i)\) such that \( F_m = \{\} \) can be built by a construction similar to the previous one.

We define \( F_1 = \{G\} \), and we have \( F^\Delta_1 = (F_1)_{\Delta_i} \).

Assume \( F^\Delta_k = (F_k)_{\Delta_i} \), and let \( G_{ik} \) and \( L_i \) be the chosen goal in \( F^\Delta_k \) and the selected literal in \( G_{ik} \), respectively.

Then there is a goal \( G_{ik} \) in \( F_k \) such that \( L_i \in G_{ik} \).

— If \( L_i = A \) (positive literal) is defined in \( P_{\Delta_i} \), let \((C_A)_{\Delta_i}\) be the set of clauses defining \( A \) in \( P_{\Delta_i} \).

We define \( F_{k+1} = F_k \setminus \{G_{ik}\} \cup \{\text{resolvents}(G_{ik}, L_i, C_A)\} \). It follows that \( F_{k+1} = (F_{k+1})_{\Delta_i} \).

— If \( L_i = A \) and \( A \) is not defined in \( P_{\Delta_i} \), \( A \) is not defined in \( P \).

We define \( F_{k+1} = F_k \setminus \{G_{ik}\} \), and it follows that \( F_{k+1} = (F_{k+1})_{\Delta_i} \).

— If \( L_i = \sim A \), and there exists an SLDNF success derivation for \( A \) from \( P_{\Delta_i} \), then there exists a corresponding abductive success derivation for \((A, \Delta_i)\) from \( P \), with computed answer \( \Delta_i \) (Part 1 of this proof).

Therefore, the rule \((F \sim A_i)\) can be applied, thus defining \( F_{k+1} = F_k \setminus \{G_{ik}\} \).

For \( k = n \), \( F^\Delta_n = (F_n)_{\Delta_i} \). If \( F^\Delta_n = \{\} \), then \( F_n = \{\} \), that is, the abductive derivation is a finitely failing one.

- Inductive Step

1. The proof is quite similar to that of the base case.

Let \( G^\Delta_1, \ldots, G^\Delta_n \) be the given SLDNF success derivation for \( G_{\Delta_i} \) from \( P_{\Delta_i} \). \( I \) being an FF-Stable model of \( P \), each subgoal \( G^\Delta_k, 1 \leq k \leq n \) does not contain any negative literal (Lemma A.4).

Following the same construction applied in the proof of the base of the induction, the first \( n - 1 \) steps of the abductive derivation for \((G, \Delta')\) are defined:

\[
(G_1, \Delta'), (G_2, \Delta'), \ldots, (G_n, \Delta').
\]

If \( G_n = \square \), the property is proved; otherwise, let \( G_n = L_0, \ldots, L_m \); each \( L_i \) is a negative literal belonging to \( \Delta_j \), and then not appearing in \( G_{\Delta_i} \) nor in \( P_{\Delta_i} \). In order to conclude the abductive derivation for \((G_1, \Delta')\), a selection rule selecting the literals in \( G_n \) according to the order they appear in it can be adopted: at step \( n + i, 0 \leq i \leq m \), the literal \( L_i \) is selected in \( G_{n+i} \).
---If $L_i \in \Delta_{n+i}$, then $G_{n+i+1} = L_{i+1}, \ldots, L_m$ and $\Delta_{n+i+1} = \Delta_{n+i}$.

---If $L_i = \neg A$ and $L_i \notin \Delta_{n+i}$, then the rule $(R \sim A_2)$ is applied. The rule is applicable since $\sim A \in \Delta$, and therefore $A \in F$.

Since $I = \langle T, F \rangle = P_\Delta^\omega$ (by hypothesis), then $T_\Delta^\omega(A) = \text{false}$ and, hence, there exists an SLDNF finitely failing derivation for $\{A\}$ from $P_\Delta$ (from the completeness of SLDNF).

By inductive hypothesis, since $\Delta' \subseteq \Delta_{n+i} \subseteq \Delta_{n+i} \cup \{\sim A\}$, there exists an abductive finitely failing derivation for $(\{A\}, \Delta_{n+i} \cup \{\sim A\})$ with computed answer $\Delta_f$ such that $(\Delta_{n+i} \cup \{\sim A\}) \subseteq \Delta_f \subseteq \Delta$. Rule $(R \sim A_2)$ generates the pair $(G_{n+i+1}, \Delta_{n+i+1})$, where $G_{n+i+1} = L_{i+1}, \ldots, L_m$ and $\Delta_{n+i+1} = \Delta_f \subseteq \Delta_f$.

Dealing this way with each literal in $L_0, \ldots, L_m$ we build an abductive success derivation for $(G, \Delta')$:

$$(G_1, \Delta'), (G_2, \Delta'), \ldots, (G_n, \Delta'), (G_{n+1}, \Delta_{n+1}), \ldots, (G_{n+m+1}, \Delta_{n+m+1}),$$

where $G_1 = G, G_n = L_0, \ldots, L_m, G_{n+i} = L_i, \ldots, L_m, \Delta_{n+i} \subseteq \Delta_{n+i+1} \subseteq \Delta$,

$\forall i = 0, \ldots, m$, and $G_{n+m+1} = \varnothing$.

2. We proceed as in the base case. [\]

Corollary A.3. Given a program $P$ and a goal $G$, let $I$ be an FF-SM of $P$. If $G_{\Delta}$ has an SLDNF success $\langle(G_\Delta)\rangle$ has an SLDNF finitely failing) derivation from $P_\Delta$, then $(G, \varnothing)$ has an abductive success $\langle(G), \varnothing\rangle$ has an abductive finitely failing) derivation from $P$ with computed answer $\Delta''$, where $\Delta'' \subseteq \Delta_f$.

PROOF. This corollary immediately follows from Lemma A.5, for $\Delta' = \varnothing$. [\]

We can now prove the completeness of the abductive procedure.

PROOF OF THEOREM 4.2.

• If there exists an FF-Stable Model $I$ such that $I(G) = \text{true}$, then $I(G_\Delta) = \text{true}$ and, hence (since $I$ is an FF-Stable Model and, thus, $I = T_\Delta^\omega$), there exists an SLDNF success derivation for $G_{\Delta}$ from $P_\Delta$ (by completeness of SLDNF).

By Corollary A.3, there exists $\Delta'' \subseteq \Delta_f$ such that $(G, \varnothing)$ has an abductive success derivation with computed answer $\Delta''$.

• The proof for the finitely failing case is similar to the one for success. [\]

A.3. Proof of Theorem 5.1

We have to prove that given a program $P$ and an interpretation $I = \langle T, F \rangle$, $I = T_\Delta^\omega$ (i.e., $I$ is an FF-Stable Model of $P$) iff $I = T_{P/I}^\omega$.

PROOF. We have to prove that:

1. $I = \langle T, F \rangle = T_\Delta^\omega \Rightarrow I = T_{P/I}^\omega$.
2. $I = \langle T, F \rangle = T_{P/I}^\omega \Rightarrow I = T_\Delta^\omega$.

1. Assume that $I$ is an FF-SM of $P$, i.e., $I = T_\Delta^\omega$. We prove separately the two inclusions: $I \supseteq T_{P/I}^\omega$ and $I \subseteq T_{P/I}^\omega$.

• $I = T_{P/I}^\omega \Rightarrow (\forall n < \omega$, for each literal $A$ in the language of $P, T_{P/I}^\omega(A) = \text{true/false} \Rightarrow T_\Delta^\omega(A) = \text{true/false}$).
The statement is proved by induction on the index $n$:

**Base of induction.** $T^0_{P/I} = T^0_{P} = \langle \emptyset, \emptyset \rangle$.

**Inductive step.** If $T^m_{P/I}(A) = \text{true}$, then there exists in $P/I$ a clause $A: -B_1, \ldots, B_m$ ($m \geq 0$) with $T^m_{P/I}(B_j) = \text{true}$, $j = 1, \ldots, m$.

The considered clause is a positive one. Indeed, the transformed program $P/I$ does not contain any negative literal (by definition), and, since the body of the clause has truth-value $\text{true}$, it does not contain the atom $u$ which would make undefined the value of the body.

Hence, the same clause also belongs to $P_I$; therefore, $T^n_{P_I}(B_j) = \text{true}$, $j = 1, \ldots, m$ (inductive hypothesis), and $T^n_{P}(A) = \text{true}$.

If $T^{n+1}_{P/I}(A) = \text{false}$, we prove that $T^n_{P_I}(A) = \text{false}$. If $A$ is not defined in $P_I$, the equality is trivially proved. If $A$ is defined in $P_I$, all the clauses defining it must be taken into account. For each of them, two cases are possible: either a corresponding clause (resulting from the second step of the transformation $P/I$ which replaces with $u$ some negative literals) belongs to $P/I$ or the clause has been dropped from it. In the first case, the clause in $P/I$ has the form $A: -B_1, \ldots, B_m$, with $m > 0$, and for at least one literal $B_j$, $1 \leq j \leq m$, $T^n_{P/I}(B_j) = \text{false}$.

Because of its truth-value $\text{false}$, $B_j$ cannot be the atom $u$; therefore, it also appears in the body of the considered clause in $P_I$. By the inductive hypothesis, $T^n_{P_I}(B_j) = \text{false}$; thus, the body of the clause is false at $\omega$.

In the second case, the clause has been dropped from $P/I$ since its body contains some literal $\sim B$ such that $B \in T$. Since, by the hypothesis, $\langle T, F \rangle = T^n_{P}$, for the atom $B$ there exists an index $j < \omega$ such that $T^n_{P_I}(B) = \text{true}$ (and $\forall i < j, T^n_{P_I}(B) = \text{undefined}$); therefore, $T^n_{P_I}(\sim B) = \text{false}$. Hence, $T^n_{P_I}(\sim B) = \text{false}$ (by monotonicity of $T^n_{P_I}$).

Having considered all the clauses defining $A$ in $P_I$, and having proved that their bodies are all false at $\omega$, we conclude that $T^n_{P_I}(A) = \text{false}$.

- $I = T^n_{P_I} \Rightarrow (\forall n < \omega, \text{for each literal } A \text{ in the language of } P, T^n_{P_I}(A) = \text{true/false } \Rightarrow T^n_{P/I}(A) = \text{true/false})$.

Also, this statement is proved by induction on the index.

**Base of induction.** $T^0_{P/I} = T^0_{P} = \langle \emptyset, \emptyset \rangle$.

**Inductive step.** If $T^{n+1}_{P_I}(A) = \text{true}$, then there exists in $P_I$ a clause $A: -L_1, \ldots, L_m$, $m \geq 0$, such that $T^n_{P_I}(L_i) = \text{true}$, $i = 1, \ldots, m$. Each literal in the clause is a positive one (indeed, if it were that $L_i = \sim B_i$, from the truth of $L_i$, it would follow that $T^n_{P_I}(B_i) = \text{false}$: due to the monotonicity of $T^n_{P_I}$, $B_i$ would belong to $F$, and the literal $L_i$ would have been dropped from the considered clause).

Hence, the same clause is also present in $P/I$; by the inductive hypothesis, $T^n_{P/I}(L_i) = \text{true}$, and then $T^{n+1}_{P/I}(A) = \text{true}$.

If $T^{n+1}_{P/I}(A) = \text{false}$ and $A$ is not defined in $P_I$, $A$ is not defined in $P/I$ either; then $T^{n+1}_{P/I}(A) = \text{false}$.

Otherwise, all the clauses defining $A$ in $P/I$ must be taken into account. Each of them is equal (modulo the replacement of $u$ to the negative literals not belonging to $T \cup F$) to a clause in $P_I$. Let $A: -L_1, \ldots, L_m$, $m > 0$, be one of
such clauses in $P_I$. From the hypothesis $T^{n+1}_{P_I}(A) = \text{false}$, it follows that there exists $k, 1 \leq k \leq m$, such that $T^n_{P_I}(L_k) = \text{false}$.

$L_k$ is a positive literal (otherwise, if it were that $L_k = \sim B_k$, it would be that $B_k \in T$; this cannot be the case, since the clause we are considering in $P_I$ has a corresponding one in $P/I$, and if it were $B_k \in T$, the corresponding clause in $P_I$ would have been dropped out by the transformation). Hence, the literal is present in the body of the corresponding clause in $P/I$ and the inductive hypothesis can be applied to it. Hence, $T^n_{P/I}(L_k) = \text{false}$.

The same argument applies to each clause defining $A$ in $P/I$, thus obtaining $T^{n+1}_{P/I}(A) = \text{false}$.

2. In order to prove that if $I = T^{n}_{P/I}$ then $I$ is an FF-SM of $P$, we prove the following two inclusions: $I \subseteq T^{n}_{P/I}, I \supseteq T^{n}_{P/I}$.

- $I = T^{n}_{P/I} \Rightarrow (\forall n < \omega, \forall each literal $A$ in the language of $P$, $T^{n}_{P/I}(A) = \text{true/false} \Rightarrow T^{n}_{P/I}(A) = \text{true/false}$).

The proof is by induction on the index $n$.

Let us first consider the case $T^{n}_{P/I}(A) = \text{true}$.

**Base of induction.** $T^0_{P/I} = T^0_{P_I} = (\emptyset, \emptyset)$.

**Inductive step.** If $T^{n+1}_{P/I}(A) = \text{true}$, then there exists in $P/I$ a clause $A: -B_1, \ldots, B_m$ such that $T^{n+1}_{P/I}(B_j) = \text{true}$, for all $j \leq m$. The same clause is also present in $P_I$; by the inductive hypothesis $T^{n}_{P/I}(B_j) = \text{true}$, $j \leq m$, and therefore $T^{n+1}_{P/I}(A) = \text{true}$. Thus, $T^{n}_{P/I}(A) = \text{true}$.

Let us now consider the case $T^{n+1}_{P/I}(A) = \text{false}$.

**Base of induction.** $T^0_{P/I} = T^0_{P_I} = (\emptyset, \emptyset)$.

**Inductive step.** If $T^{n+1}_{P/I}(A) = \text{false}$, $A$ is not defined in $P/I$ and $A$ is not defined in $P_I$, then the falsity of $T^{n+1}_{P/I}(A)$ immediately follows. If $T^{n+1}_{P/I}(A) = \text{false}$, $A$ is not defined in $P/I$ but $A$ is defined in $P_I$, each clause defining $A$ in $P_I$ contains in its body at least one literal $\sim B$, such that $B \in T$. Let $B$ be one of such atoms belonging to $T$. Since $I = (T, F) = T^{n}_{P/I}$ (by hypothesis), $T^{n}_{P/I}(B) = \text{true}$. Hence, for some finite $m$, $T^{m}_{P/I}(B) = \text{true}$, and, by the proof above, $T^{m}_{P/I}(B) = \text{true}$. Therefore, $T^{n}_{P/I}(B) = \text{false}$.

If $A$ is defined in $P/I$, each clause defining it must be considered. Let $A: -L_1, \ldots, L_m, m > 0$ be one such clause. Its body contains at least a literal $L_i$, $1 \leq i \leq m$ such that $T^{n}_{P/I}(L_i) = \text{false}$. Due to its truth-value, this literal cannot be $u$; therefore, it also appears in the body of the corresponding clause in $P_I$, and the inductive hypothesis can be applied.

Having considered all the clauses defining $A$ in $P_I$, the conclusion $T^{n}_{P/I}(A) = \text{false}$ follows.

- $I = T^{n}_{P/I} \Rightarrow \forall n < \omega, \forall each literal $A$ in the language of $P$, $T^{n}_{P/I}(A) = \text{true/false} \Rightarrow T^{n}_{P/I}(A) = \text{true/false}$.

By induction on the index:

**Base of induction.** $T^0_{P/I} = T^0_{P_I} = (\emptyset, \emptyset)$.

**Inductive step.** If $T^{n+1}_{P/I}(A) = \text{true}$, then there exists in $P/I$ a clause $A: -L_1, \ldots, L_m$ such that $T^{n}_{P/I}(L_i) = \text{true}$, $i \leq m$. The same clause also belongs to $P/I$. Indeed, every clause removed from $P/I$ contains in its body a literal $\sim B$ such that $B \in T$, i.e., $T^{n}_{P/I}(\sim B) = \text{false}$, and this is not the
case, since $\mathcal{T}^n_{P/I}(L_i) = \text{true}$ (by the inductive hypothesis), and thus $\mathcal{T}^n_{P/I}(L_i) = \text{true}$ ($\mathcal{T}^\nu_{P/I}$ is monotonic). By applying the inductive hypothesis to each literal in the body of the considered clause, we obtain $\mathcal{T}^n_{P/I}(A) = \text{true}$.

If $\mathcal{T}^n_{P/I}(A) = \false$, then all the clauses defining $A$ in $P_I$ contain in their bodies at least one literal false in $\mathcal{T}^n_p$. Consider one of these clauses, and let $L_i$ be one of the literals in its body such that $\mathcal{T}^n_{P/I}(L_i) = \false$. By the inductive hypothesis, $\mathcal{T}^n_{P/I}(L_i) = \false$. Hence, $\mathcal{T}^n_{P/I}(L_i) = \false$, i.e., $I(L_i) = \false$. If the corresponding clause belongs to $P/I$, then $L_i$ is a positive literal occurring in its body (otherwise, that clause would have been removed from $P/I$). Hence, the body of the clause is false in $\mathcal{T}^n_{P/I}$.

Since every clause in $P/I$ has a corresponding one (equal, modulo $\mathcal{U}$) in $P_I$, all the clauses defining $A$ in $P/I$ have been considered. It follows that $\mathcal{T}^n_{P/I}(A) = \false$. □

We thank Robert Kowalski and the anonymous referees for their helpful comments. This work has been partially supported by CNR—Progetto Finalizzato “Sistemi Informatici e Calcolo Parallelo” under Grant 93.01598.PF69.

REFERENCES