Asserting Lemmas in the Stable Model Semantics

Stefania Costantini
Gaetano Aurelio Lanzarone
Giuseppe Magliocco
Dipartimento di Scienze dell'Informazione
Universita' degli Studi di Milano
via Comelico 39/41, I-20135 Milano (Italy)
costanti@dsi.unimi.it
lanzarone@hermes.mc.dsi.unimi.it
Ph. +39(2)55006.259/324
Fax +39(2)55006.253

Keywords:
Non–monotonic reasoning, semantics and foundations, stable model semantics

Abstract
From a logic programming point of view, the stable model semantics for normal programs has the problem that logical consequences of programs cannot, in general, be stored as lemmas. This is because the set of stable models of the resulting program may change. In fact, logical consequence under the stable model semantics does not enjoy an important property required of non–monotonic entailment relations, i.e. *cumulativity*. We argue that it is possible to assert a conclusion A as a lemma in the stable model semantics, if asserting at the same time a set of facts supporting the conclusion (that we call a *base set* for A). The effect on the meaning of the program is that of selecting some of the stable models containing A. The collection of all base sets for A generates all the stable models containing A. We formalize this intuition by reformulating the definition of cumulativity accordingly. We propose a characterization of base sets that identifies the minimal ones, i.e. the fewest and smallest base sets for A. Any proof procedure for the stable model semantics (including the abductive ones) should be able, with slight modifications, to return the base sets, by applying the criteria that we propose.

1 Introduction

The stable model semantics [10], originated from the work on autoepistemic logic [9], appears presently as the main proposal for dealing with reasoning by cases, multiple alternative situations, and abduction.

Stable models do not exist for all normal programs, and are in general not unique. The definition of stable models is not constructive, and involves a generate–and–test process. In fact, verifying the existence of a stable model is NP–complete (see [11] and the references therein). Nevertheless, there have been several proposals for stable model computation (see [2] and the references therein, and [3]), whose major issue is to riduce the search space and to avoid expensive guessing as far as possible.

The motivations of this paper are as follows. From a logic programming point of view, the stable model semantics has an important drawback: if a given fact is a consequence of a program w.r.t. the stable model semantics, it is in general not possible to store it as a lemma. In fact, as it is well–known, the set of stable models of the resulting program may change, even for programs with a unique stable model, like for instance the following.
$q : -not\, r.$
$r : -not\, q.$
$p : -not\, p.$
$p : -not\, r.$

This program P has the unique stable model $\{p, q\}$. Even though p is a consequence of P, its addition to the program alters the semantics, since the resulting program has the two stable models $\{p, q\}$ and $\{p, r\}$.

The problem has been formalized in [5] [6], [7], where it is shown that logical consequence under the stable model semantics does not enjoy an important property required of non–monotonic entailment relations, i.e. *cumulativity*.

We may notice however that, on the basis of autoepistemic logic, the stable models represent coherent sets of beliefs that can be derived from themselves. Inconsistencies do not arise because facts in a stable model are *supported* by other facts, that we can see as *hypotheses*. In the example, p is supported by q (or, symmetrically, by $not\, r$), since the clause $p : -not\, p$ is contradictory. In fact, if we add both p and q as facts, (or, symmetrically, we add p and cancel the premise $not\, r$ from every clause where it occurs, which means that we assume $not\, r$ as true), then it is easy to see that the stable model of the resulting program is unchanged, i.e. it is still $\{p, q\}$.

A similar intuition underlies the formulation of negation–as–failure as abduction [8] where every conclusion is considered to be supported, w.r.t. each stable model, by a set of negative facts, that is considered as the *abductive explanation* of the conclusion.

The main contributions of this paper are twofold. In the first part of the paper, we assume that it is possible to assert a conclusion A as a lemma in the stable model semantics, if at the same time a set of facts supporting the conclusion (that we call a *base set* for A) is also asserted. In our view, the effect on the meaning of the program should be that of selecting some of the stable models containing A. In fact, A may have in principle several base sets (at most as many as the stable models to which A belongs), or even none. We also argue that, actually, not every atom in a stable model needs base sets, but only those involved in some odd cycle (cycle involving an odd number of negative dependencies). We formalize this intuition by reformulating the definition of cumulativity accordingly,

in both a weak and a strong form (we call the new property *extended cumulativity*). In the strong form, it is required that, if considering all the base sets for A, all the stable models entailing A are obtained. As a main difference w.r.t. [5], [6], [7], we propose a concept of cumulativity suitable for the case where there are several stable models or, more generally, for any semantics which is not based on a unique model. In the above papers instead, there is the explicit intended restriction to semantic approaches based on a unique model. At this point, it is easy to show that the stable model semantics enjoys extended cumulativity in the strong form. In fact, the sets of abductive explanations are base sets for A, although in general they are quite redundant.

In the second part of the paper, we propose a characterization of the base sets, aimed at identifying the minimal ones. This characterization is intended to be of practical usability, since any proof procedure for the stable model semantics (including the abductive ones) should be able, with slight modifications, to return the support sets, by applying the criteria that we propose. Preliminarly, we have to introduce a novel representation for the syntactic structure of a program, that makes explicit the existing cycles, and their connections. This is because the dependency graph, which is usually adopted for studying properties of programs, is not suitable w.r.t. the stable models semantics, since there are programs with the same dependency graph, but different stable models. On the basis of this representation, from the structure of the program we are able to identify very restricted base sets for a given atom A. We characterize the minimal ones, in the sense that it does not exist any other collection consisting of fewer and smaller base sets for A.

2 Preliminary Definitions and Observations

In this paper we consider normal logic programs, i.e. Horn-clause programs with negation. As basic references for semantics of normal logic programs we take [1] and [12], which contain a systematic description and comparison of the various proposed semantics, and where the reader can find all the original references corresponding to the definitions that we report below.

Clauses in a normal program have an atom as conclusion, and a conjunction of literals as conditions, where a literal is either an atom (positive literal) or the negation of an atom (negative literal). We assume that the order of literals in the conditions of clauses is irrelevant. When saying "a program" we mean (unless explicitly specified differently) a normal program. In the following, let P be a program. Negation is indicated by *not*. We consider Herbrand interpretations and Herbrand models only. Since an Herbrand interpretation is a model of a program P if and only if it is a model of its ground instantiation, we will assume that every program P has already been instantiated. By H_P and B_P we indicate respectively the Herbrand Universe and the Herbrand Base of P. Let also be $B_P^{ext} = B_P \cup \{not\, A : A \in B_P\}$, i.e. the set af all positive and negative literals that can be built from the alphabet of P.

The dependency graph D_P for P is a directed graph with signed edges, whose nodes are the atoms which occur in P. Given a clause in P of the form
$A : -B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m,\ n, m \geq 0$,
D_P contains n *positive* edges, namely (A, B_i), $i \leq n$, and m *negative* edges, namely (A, C_j), $j \leq m$. We say that A depends *positively* (resp. *negatively*) on B if there is a path in D_P

from A to B with only positive edges (resp. at least one negative edge). We say that A depends *evenly* (resp. *oddly*) on B if there is a path in D_P from A to B with an even (resp. odd) number of negative edges. P is called *call–consistent* if no relation depends oddly on itself. A *local stratification* for P is a function $\mathcal{S}$ from B_P to the countable ordinals, which is extended to the negative literals $not\, A$, $A \in B_P$, by assuming that $\mathcal{S}(not\, A) = \mathcal{S}(A) + 1$. P is called *locally stratified* if there exists a local stratification $\mathcal{S}$ for P such that for every clause $A : -Conds$ in P, and for every literal L occurring in $Conds$, $\mathcal{S}(A) \geq \mathcal{S}(L)$.

A program P may have in general several stable models [10], defined as follows, which are among the minimal models of P.

(The Gelfond–Lifschitz Operator). *Let I be a 2-valued interpretation of P. A GL-transformation of P modulo I is a new program P/I obtained from P by performing the following two reductions:*

1. *removing from P all clauses which contain a negative premise $notA$ such that $notA$ is false in I (equivalently, $A \in I$);*
2. *removing from the remaining clauses those negative premises $notA$ such that $notA$ is true in I (equivalently, $A \notin I$).*

P/I is clearly a positive Horn-clause program, with least Herbrand Model J. Let $\Gamma(I) = J$. A 2-valued interpretation I of P is called a Stable Model of P if $\Gamma(I) = I$.

The stable model semantics (SMS for short) defines the meaning of P as the set of its stable models. If P is locally stratified, than it has a unique stable model. If P is call–consistent, than it has stable models.

The well–founded model $WFM_P = \langle T(P); F(P) \rangle$ of P [14] is unique, and is in general 3-valued. $T(P) \subseteq B_P$ is the set of atoms which are true w.r.t. the WFM, $F(P) \subseteq B_P$ the set of atoms which are false. All atoms belonging to $U(P) = B_P - (T(P) \cup F(P))$, have truth value undefined. Whenever WFM of P is total (i.e. two-valued) it coincides with the unique stable model of the program, and P is called *effectively stratified*; local stratification is a special case of effective stratification. Notice however that there are programs with a unique stable model, but a three-valued WFM. In any case, all stable models of a program *extend* its WFM, since for every stable model M of P, $T(P) \subseteq M$ and $M \subseteq T(P) \cup U(P)$.

Consequently, the relevant information about the stable models of programs that are not effectively stratified is limited to (a subset of) the clauses whose conclusion is undefined in WFM_P [3] [13]. Let us consider a subprogram of P, obtained by canceling from P all clauses with conclusion true/false in WFM_P, and all clauses with conclusion undefined and at least one of the conditions false in WFM_P. Let us also assume to cancel from every clause of this subprogram all the literals which are true in WFM_P, and call PU the resulting program. Let $PU\hat{}$ be the unfolded version of PU, where negative dependencies become explicit. I.e., $PU\hat{}$ is the (possibly infinite) program obtained by repeatedly substituting, in all possible ways, every positive literal in the conditions of each clause with the conditions of the clauses where it occurs as the conclusion. Let us select the (finite, possibly empty) subprogram P* of $PU\hat{}$, composed of all clauses without positive conditions.

For the sake of simplicity and without loss of generality, in the rest of this paper, when mentioning a program P, we implicitly refer to its reduced version P*. In fact, as

will be easily seen, we claim that all the reasoning performed in the following sections on P* could be similarly performed on the original program P, at the expense of additional complications due to considering positive and acyclic dependencies.

2.1 Dividing a Program into Cycles

The dependency graph is not a suitable representation of a program for investigating about its SMS [4]. In fact, there are programs that have the same dependency graph, but different sets of stable models. Therefore, in [4] a different representation for the syntactic structure of a program is introduced, that makes explicit the cycles in the program and the different kinds of connections between them.

In the definitions below, let P be a program, let $\{a, a_1, \ldots, a_n\} \subseteq B_P$, $n \geq 1$, let the B_i's, B_{i_h}'s, D_i's, D and E be (possibly empty) conjunctions of negative literals. Given positive integers g and m, by $g \, mod \, m$ we mean the remainder of the integer division of g by m.

Definition 2.1 *A cyclic set of clauses CS is a set of clauses of the form:*
$a_1 : -not\, a_2, B_1.$
$a_2 : -not\, a_3, B_2.$
$\vdots$
$a_n : -not\, a_1, B_n.$

More precisely, a clause composing a cyclic set is of the form:
$a_i : -not\, a_{(i+1) \bmod n}, B_i.$
where $1 \leq i \leq n$, $n \geq 2$.

We admit the special case of a single–clause cyclic set of the form:
$a_1 : -not\, a_1, B_1.$

We also admit the special case where for each of the a_i's, $i \leq n$, there are h clauses, $h > 1$, of the form:
$a_i : -not\, a_{(i+1) \bmod n}, B_{i_h}.$
We say that C consists of *$\{a_1, \ldots, a_n\}$ or that it* contains *the a_i's, which* belong to *it; the composing clauses* are in*, or* belong to*, C, which* contains *them. We say that a cyclic set C is* even *(resp.* odd*) if n is even (resp. odd).*

Notice that it is not required that the B_j's (and the B_{i_h}'s) do not contain the a_i's. Also, notice that the same clause may belong to more then one cyclic set.

Definition 2.2 *Let CS be a cyclic set of clauses. We call* auxiliary clause *of CS a clause, say K, of the form:*
$a_i : -D.$
where: a_i belongs to CS; and K is not in CS.

Definition 2.3 *We call* isolated clause *a clause of the form:*
$a : -E.$
which does not belong to any cyclic set, and is not auxiliary to any cyclic set.

Notice that an isolated clause is not supposed to have auxiliary clauses. I.e., there may be several isolated clauses with the same conclusion, but they are seen as unrelated. It is convenient to consider an isolated clause as a special case of a cycle.

Definition 2.4 *A* cycle *C is either a cyclic set of clauses (*proper cycle*), or an isolated clause. For a proper cycle, the auxiliary clauses are those of the cyclic set it consists of.*

Definition 2.5 *Let C be a cycle. For every clause K in C, of the form:*
$a_i : -not\, a_{i+1}, B_i.$
B_i is called an AND handle *of K; by the definition of a cycle, B_i may be empty. The literal $not\, a_{i+1}$ is called the* main condition*. In the special case of isolated clauses, the main condition is empty. For every auxiliary clause X of C, of the form:*
$a_i : -D.$
D is called an OR handle *of X. With respect to the whole cycle, the B_i's are called the* AND *handles of C, and the D's the* OR *handles. We say that the composing literals* belong to *or* occur in *the handle. Let $\{D_1, \ldots, D_r\}$, $r \geq 0$, be the (possibly empty) set of the OR handles of C. We call H_C the (possibly empty) set of all the handles of C, namely $\{B_1, \ldots, B_n, D_1, \ldots, D_r\}$*

We say that a cycle C *has* an AND (resp. OR) handle if at least one of the B_i's, $i \leq n$ (resp. D_j's, $j \leq r$) is nonempty. More generally, we say that a cycle C is *unconstrained* (resp. *constrained*) if its set of handles H_C is empty (resp.nonempty).

Given clause K in cycle C, of the form:
$a_i : -not\, a_{i+1}, B_i.$
if i is even (resp. odd) then, w.r.t. C, K is called an even (resp. odd) clause, a_i an even (resp. odd) atom, and B_i an even (resp. odd) handle.

It is easy to see that a program P can be partitioned in a unique way into a set of cycles $\{C_1, \ldots, C_k\}$, $k \geq 1$, each with its auxiliary clauses.

In [4] the relationship between this partition and the existence of stable models is widely discussed. In this paper, the decomposition into cycles will be useful for identifying the base sets.

3 Properties of the Stable Model Semantics

It is useful to classify the various semantics of non-monotonic formalisms on the basis of the relevant structural properties that a nonmonotonic entailment relation should satisfy. This approach has been applied to logic programming in [5], [6], [7].

One main such property is *cumulativity* (*CM* for short), according to which an entailment relation ($\models$) defined by a logic programming semantics should satisfy, given a normal program P, the condition

$$\text{If } P \models A \text{ then } P \models B \text{ iff } P \cup A \models B$$

It is well–known that SMS violates this property, even for programs with a unique stable model. This means that if we store the conclusion A as a lemma, we run the risk of modifying the set of consequences of the original theory.

We will argue that the property is to be violated because atoms belonging to stable models are by no means independent of the hypotheses supporting them. Then, in order to assert A, we also have to assert a set $H(A)$ of hypotheses supporting A, as follows.

Definition 3.1 ($\mathcal{S}$–Reduction) *Let $A \in B_P$, $H(A) \subseteq B_P^{ext}$.*
An $\mathcal{S}$–Reduction of P w.r.t. A and H
is a new program $P \cup_{H(A)}(A)$ obtained from P by adding both A and every positive literal $B \in H(A)$ as unit clauses, and removing from the clauses of P those negative premises $notC \in H(A)$.

Differently from [5], [6], [7], we take the position of accepting as reasonable semantics also those semantics which possibly associate a program with several intended interpretations (like of course SMS), that we call *multimodel semantics*. Thus, for the notion of entailment w.r.t. a multimodel semantics S, we take the credulous position, stating that A is a consequence of a program P w. r. t. a semantics S if A belongs to some of the intended interpretations of P under S. That is, if $A \in B_P$, and $\mathcal{M}(P)$ is the set of the intended interpretations of P given by S, $P \models_S A$ iff $\exists I \in \mathcal{M}(P)$ such that $A \in I$. Let $\mathcal{M}^A(P)$ be the set of those intended interpretations of P which contain the atom A. By abuse of notation, we will say that A is true in P w.r.t. S if $P \models_S A$.

Assume to assert as a lemma an atom A belonging to one of the intended interpretations of P under S. The effect, in our view, should simply correspond to restricting the meaning of P to those intended interpretations of P containing A. This is done, as argued above, at the condition of asserting also a suitable base set for A.

Then, we propose the following reformulation of the property of cumulativity, and we call the new property "Extended Cumulativity" (ECM).

Definition 3.2 (Condition for *Weak ECM* of a semantics S)
Let $M \in \mathcal{M}(P)$ under S, and let $A \in M$. There exists a set $H(A) \subseteq B_P^{ext}$ such that $\mathcal{M}(P \cup_{H(A)}(A)) \subseteq \mathcal{M}^A(P)$.
The set $H(A)$ is called a base set for A.

If S produces a single intended interpretation, then the above definition reduces to a revision of the original definition.

We say that $H(A)$ *selects* the set of interpretations $\mathcal{M}(P \cup_{H(A)}(A))$. That is, we say that a semantic S is cumulative if, for any given intended interpretation M of P under S, and for any A in M, it is possible to find a set of literals $H(A)$ which supports A in M. I.e., performing an $\mathcal{S}$–reduction of P modulo A and $H(A)$ corresponds to selecting a subset of the intended interpretations containing A. The motivation of this weak condition is that the atom A may have several base sets, as shown by the following example.

Example 3.1 *Consider the following program P.*
q:-not r.
r:- not q.
a:-not b.
b:- not a.
p:- not p.

p:- not a.
p:- not r.
P has the stable models {p, q, a}, {p, q, b}, {b, p, r}. Both $H_1 = \{b\}$ and $H_2 = \{q\}$ are base sets for p (or, symmetrically, we can consider {not a} and {not r}). According to the definition of weak CM, H_1 selects the models {p, q, b} and {b, p, r},while H_2 selects the models {p, q, a} and {p, q, b}. Notice that the union of the set of models selected by H_1 with the set of models selected by H_2 gives the whole set of stable models of P.

We can formulate a strong version of ECM, requiring exactly that the union of the set of intended interpretations selected by the different base sets corresponds to the whole set of intended interpretations, according to the given semantics. Formally, we can give the following definition.

Definition 3.3 (Condition for *Strong ECM* of a semantics S)
Let $M \in \mathcal{M}(P)$ under S, and let $A \in M$. There exist sets $H_1(A), \ldots, H_n(A)$, $H_i(A) \subseteq B_P^{ext}$, $i \leq n$, such that
$\bigcup_{i=1,\ldots,n} \mathcal{M}(P \cup_{H(A)}(A)) = \mathcal{M}^A(P)$.

If a semantics enjoys strong ECM, we call a collection of base set *complete* if it is able to select, according to the definition, all the stable models containing A.

The question now is whether the stable model semantics enjoys ECM. The answer, as suggested by the examples, is positive, for ECM in the strong form.

In the following, let A be true in P w.r.t. SMS. Notice that this assumption must be based on the assumption of having some proof procedure for the stable model semantics (see for instance [2]). With respect to each of the stable models to which it belongs, A will be supported by a base set, that characterizes A w.r.t. that particular model.

If A is true in the well–founded model of the program, then A has always an empty base set, since A belongs to every stable model, and thus does not need to be supported by any hypotheses. Then, as mentioned in Section 2, we refer to the reduced version of the program which contains all atoms that are undefined in the well–founded model.

If A belongs to some even cycle, then it has an empty base set, since (see [4] for a discussion) the existence of stable models of even cycles does not depend on the rest of the program (at most, some of the models can be excluded). Then, the problem of finding the base sets concerns only those atoms that do not belong to any even cycle.

Since we assume A to be true w.r.t. SMS, in order to find the base set of A, we can consider (in the terminology of [5], [6], [7]) the subprogram P(A) = rel_rul(P,A), consisting of all rules that could contribute to A's derivability, i.e. the set of rules concerning all atoms on which A depends (positively/ negatively, directly/ undirectly).

If instead we should *establish* whether A is true/false, then we should consider a wider subprogram, since SMS does not satisfy *relevance*, i.e. it is not true that the truth value of A in P is equal to the truth value of A in P(A). This is for two reasons. First, in some cases P(A) has stable models, while P has none. Second, A could be derivable in P(A), but not derivable in P, since its truth would make P inconsistent.

Let $S(A)_1, \ldots, S(A)_k, k \geq 0$, be the stable models for $P(A)$ containing A. Let $S(A)_i{}^- = \{B : B \in U(P) - S(A)_i\}, i \leq k$ be the *negative counterpart* of the $S(A)_i$'s, containing, for $i \leq k$, the negation of every atom which is undefined in the well-founded model but does

not belong to $S(A)_i$. The $S(A)_i{}^-$'s, that are exactly the abductive explanations for A in $P(A)$, are *candidate base sets* for A. In fact, they must be checked for consistency w.r.t P–P(A), for the reasons shown in the following example.

Example 3.2 *Consider the program in Example 3.1, and assume to add the clause:* $g : -not\, g, not\, a$

The resulting program has the unique stable model $\{a, p, q\}$. *In fact, a must be true in order to make g false. Otherwise, the new clause would be equivalent to the uncoherent clause* $g : -not\, g$. *Then,* $\{b\}$ *is no longer a base set for p. From P(p), alone, which coincides with the clauses of the previous program, we are not able to detect this situation.*

In general, there may be a problem if some atom D in a base set BS occurs in the handle of some clause K of a cycle C which is not in P(A). In this case, for every such clause, it is necessary to perform the following

Coherency check

Consider the conclusion G of K. Find the truth value of G in P (by means of a proof procedure whatsoever). Assuming the truth value of D in BS, check that the truth value of G is still the same. Discard BS if it is not.

After the consistency check on the $S(A)_i^-$'s, let $BS(A)_1, \ldots, BS(A)_m$, $m > 0$, be the remaining sets.

Proposition 3.1 *The sets* $BS(A)_r$, $r \leq m$, *form a complete collection of base sets for A.*

In fact, it is easy to see that every stable model containing A is selected by some of the $BS(A)_r$'s. Therefore, SMS enjoys ECM in the strong form, and the sets of coherent abductive hypotheses form a complete collection of base sets.

In the next section we will show however that it is possible to identify a complete collection of fewer and smaller base sets.

4 Finding a Minimal Complete Collection of Base Sets

Consider an atom A, that we assume to be true w.r.t. SMS. It is useful to make explicit, in terms of cycles and handles, what are the necessary and sufficient conditions for an atom to be true or false w.r.t. SMS.

First, notice that an handle is true (resp. false) iff every composing conjunct is true (resp., if some of the composing conjuncts is false).

Let us consider a cycle, of the form:

$a_1 : -not\, a_2, B_1.$
$a_2 : -not\, a_3, B_2.$
$\vdots$
$a_n : -not\, a_1, B_n.$

For a_i, $i \leq n$, to be true by means of this cycle, there are two possibilities:

- there exists an auxiliary clause for a_i, of the form $a_i : -B$, and B is true.
- $a_{(i+1) mod\, n}$ is false, and the AND handle B_i is true. Notice that, if $n = 1$, then this condition cannot hold, since the same atom a_1 cannot be both true and false.

Instead, for a_i, $i \leq n$, to be false, both the following conditions must hold:

- there exists no auxiliary clause for a_i, of the form $a_i : -B$, where B is true.
- either $a_{(i+1) mod\, n}$ is true, or the AND handle B_i is false. Notice that, if $n = 1$, then the former cannot hold, since the same atom a_1 cannot be both true and false.

There is a difference between an even and an odd cycle if the cycle is unconstrained. In fact, an unbounded even cycle has always two stable models (one composed of the even atoms, and one composed of the odd atoms), and then a_i will be true in one of them. Instead, an unconstrained odd cycle has no stable models: this means that, for an atom to be true/false in an odd cycle, we must assume that the cycle is not unconstrained.

It is easy to see that A is true iff it is true by means of some of the cycles to which it belongs. Conversely, A is false iff it is false in all the cycles to which it belongs. For a discussion of the impact on truth/falsity of the interactions among cycles, see [4].

Below, we arrange clauses of $P(A)$ (or, equivalently, the search space for A) into a tree, so as to make the previous considerations formal, and thus be able to find the base sets. The following preliminary definitions are in order.

We say that a cycle has an AND handle for A if $A = a_i$, $i \leq n$, and B_i is nonempty. We say that a cycle has an OR handle for A if there exists an auxiliary clause $A : -B$.

Definition 4.1 *Given a clause corresponding to an OR (resp. AND) handle of some cycles, an OR (resp. AND)–restricted version of the clause is obtained by canceling all conjuncts of the handle, except one.*

Then, for every clause corresponding to an OR (resp. AND) handle, there are as many OR(resp. AND)–restricted clauses, as there are conjuncts in the handle.

Definition 4.2 *Given a cycle C, and an atom A, such that C has an OR handle for A, an OR–restricted version of the cycle w.r.t A is obtained by canceling all the AND handles in the composing clauses, and by canceling all the auxiliary clauses, except one with an OR handle for A. This one is then substituted by an OR–restricted version of the clause itself.*

Definition 4.3 *Given cycle C, and an atom A, such that C has an AND handle for A, an AND–restricted version of the cycle w.r.t A is obtained by canceling all the auxiliary clauses, and all the AND handles, except the one for A. This one is then substituted by an AND–restricted version of the clause itself.*

Clearly, there are several OR (resp. AND)–restricted versions of the same cycle, depending on the number of handles and conjuncts. These definitions reflect the fact that, in checking the truth/falsity of A, we can consider the handles one by one and the composing conjuncts in turn one by one.

We will now define how to build a tree, called T–tree, to represent the conditions for A to be true. The T–tree may have as subtree an F–tree, to represent the conditions for some atom B to be false. The nodes of the tree have the following possible labels. **A:true**, **A:false**, where A is the atom whose truth, or resp. falsity we want to represent. **alt**, which indicates an alternative among different conditions. **un**, standing for "union", which indicates the fact that a number of different conditions must hold together.

Definition 4.4 (T–tree for A) *The nodes of the T–tree can be labeled with* **B:true** *or* **B:false**, $B \in B_P$, *or with* **alt**, *or with* **un**, *or with a set of clauses (program node).*
The T–tree has the following structure:

- *The root, labeled with* **A:true**, *has a single child, labeled with* **alt**, *which has one child for every cycle C containing A. If C is unconstrained, then the node corresponding to C is a program node, labeled with the clauses of the cycle. Otherwise, the node is labeled with* **alt**, *and has the following children, from left to right:*
  - *node labeled with* **alt**, *which is the root of the OR–subtree;*
  - *node labeled with* **un**, *which is the root of the AND–subtree.*
- *OR–subtree corresponding to a cycle C.*
  The root has as many children as the number of the OR handles for A, each one corresponding to a clause $A : -not\, B_1, \ldots, not\, B_n$. *Each child (corresponding to one OR handle) is labeled with* **un**, *and has as many children, say* $OC_1, \ldots, OC_k$, *as the number of OR–restricted versions of C w.r.t. that handle, each with atom* B_i, $i \leq k$ *as the remaining atom in the handle. Each* OC_i *is labeled with the clauses of the OR–restricted version, and has one child, which is the F–tree for* B_i.
- *AND–subtree corresponding to a cycle C.*
  Let $A : -not\, B, not\, D_1, \ldots, not\, D_r$ *be the clause CA with conclusion A in C. The root has the following children:*
  - *one child, omitted if A=B, labeled with the clauses of C, where however CA is substituted by* $A : -B$, *which has in turn one child, i.e. the F–tree for B, omitted if the resulting cycle is unbounded.*
  - *as many children, say* $AC_1, \ldots, AC_r$, *as the number of AND–restricted versions of C w.r.t. the handle for A, each with atom* D_i, $i \leq r$ *as the remaining atom in the handle. Each* AC_i *is labeled with the clauses of the AND–restricted version, and has one child, which is the F–tree for* D_i.

Definition 4.5 (F–tree for A) *The nodes of the F–tree can be labeled with* **B:false** *or* **B:true**, $B \in B_P$, *or with* **alt**, *or with* **un**, *or with a set of clauses (program node). The F–tree has the following structure:*

- *The root, labeled with* **A:false**, *has a single child, labeled with* **un**, *which has one child for every cycle C containing A. If C is unconstrained, then the node corresponding to C is a program node, labeled with the clauses of the cycle. Otherwise, the node is labeled with* **un**, *and has the following children, from left to right:*
  - *node labeled with* **un**, *which is the root of the OR–subtree;*
  - *node labeled with* **alt**, *which is the root of the AND–subtree.*
- *OR–subtree corresponding to a cycle C.*
  The root has as many children as the number of the OR handles for A, each one corresponding to a clause $A : -not\, B_1, \ldots, not\, B_n$. *Each child (corresponding to one OR handle) is labeled with* **alt**, *and has as many children, say* $OC_1, \ldots, OC_k$, *as the*

number of OR–restricted versions of C w.r.t. that handle, each with atom B_i, $i \leq k$ as the remaining atom in the handle. Each OC_i is labeled with the clauses of the OR–restricted version, and has one child, which is the T–tree for B_i.

- *AND–subtree corresponding to a cycle C.*
  Let $A : -not\, B, not\, D_1, \ldots, not\, D_r$ be the clause CA with conclusion A in C. The root has the following children:
  - *one child, omitted if A=B, labeled with the clauses of C, where however CA is substituted by $A : -B$, which has in turn one child, i.e. the T–tree for B, omitted if the resulting cycle is unbounded.*
  - *as many children, say $AC_1, \ldots, AC_r$, as the number of AND–restricted versions of C w.r.t. the handle for A, each with atom D_i, $i \leq r$ as the remaining atom in the handle. Each AC_i is labeled with the clauses of the AND–restricted version, and has one child, which is the T–tree for D_i.*

After building the tree, it is convenient to cancel all the **alt** and **un** nodes without children. In this way, all leaves are program nodes.

The T–tree for A consists, in summary, in a hierarchy of T–trees and F–trees, each referring to the truth/falsity of an atom, and associated with a restricted cycle containing that atom. Each tree at the bottom ends in a leaf of the whole tree. Assuming that there are h bottom trees, let $L_1, \ldots, L_h$ be the atoms to which they correspond, and $K_1, \ldots, K_h$ the associated restricted cycles.

The union of all clauses of the program nodes occurring in a path from the root to each K_j, $j \leq h$, is a program, that we call $SP_j(A)$. Notice that every K_j must be an unbounded even cycle. In fact, if the cycle is bounded, then the costruction of the tree does not stop here. If it is odd and unbounded, then it means that P(A), and thus P, has no stable models, which contradicts the hypothesis that A is true (the reader may refer to [4] for a discussion of these topics).

Then, K_j, like every unconstrained even cycle, has two stable models $M^1_{K_j}$ and $M^2_{K_j}$. One of these two models makes the atom L_j true/false, as required. Let us associate this model, say ML_{K_j}, with this leaf, and call it a *base model*. It is convenient, for now, to represent ML_{K_j} as the set of literals true w.r.t. this model.

If adding to $SP_j(A)$, as facts, the atoms which occur positively in ML_{K_j}, $SP_j(A)$ may become locally stratified, with A true in its unique stable model. Otherwise, it may be the case that the resulting program has no stable model. In this case, it is necessary to cancel ML_{K_j}, as well as any other base model of which ML_{K_j} is a subset.

Example 4.1 *Consider the following program P.*
$p : -not\, p, not\, h.$
$p : -not\, a.$
$p : -not\, b.$
$h : -not\, h, not\, e.$
$e : -not\, f.$
$f : -not\, e.$
$a : -not\, b.$
$b : -not\, a.$

The program has the two stable models $\{e, a, p\}$ *and* $\{e, b, p\}$. *Let us consider atom* p. $P(p)$ *coincides with* P. *The abductive hypotheses for* p *are* $\Delta_1 = \{not\, f, not\, b\, not\, h\}$ *and* $\Delta_2 = \{not\, f, not\, a, not\, h\}$

The T–tree for p has the following structure:

- *Root* **p:true**, *with child* **alt**.
- *OR–subtree: since there are two OR handles, the root (label* **alt***) has two nodes, each one (labeled with* **un***) with only one child, which is respectively:*
  - *program node*
    $p : -not\, p.$
    $p : -not\, a.$
    with a child which is the F–tree for a.
  - *program node*
    $p : -not\, p.$
    $p : -not\, b.$
    with a child which is the F–tree for b.

  .
- *AND–subtree: the root (label* **un***) has only one child (the first one has been omitted), i.e. the program node*
  $p : -not\, p, not\, h.$
  with a child which is the F–tree for h.

The F–tree for h has the following structure:

- *Root* **h:false**, *with child* **alt**.
- *OR–subtree: empty*
- *AND–subtree: the root (label* **alt***) has only one child (the first one has been omitted), i.e. the program node*
  $h : -not\, h, not\, e.$
  with a child which is the F–tree for e.

The F–tree for a, b, and e has the following structure:
root **a:false**, **b:false** *and* **e:false** *respectively, with child* **un** *and only one gradchild, i.e. the program node*
$a : -not\, b.$
$b : -not\, a.$
for a and b (let us call this cycle K_1*), and*
$e : -not\, f.$
$f : -not\, e.$
for e. Let us call this cycle K_2.

Let us consider the program $SP_2(p)$, *corresponding to the path from the root to* K_2, *which is:*
$p : -not\, p, not\, h.$
$h : -not\, h, not\, e.$

$e : -not\, f.$
$f : -not\, e.$

The stable model $ML_{K_2} = \{e, not\, f\}$ makes h false, as required by the tree structure. However, if e is added as a fact to $SP_2(p)$, the resulting program has no stable models. Therefore, ML_{K_2} must be excluded.

Finally, let us proceed bottom–up in the tree. This time, starting from the remaining base models, we consider only the **alt** and **un** nodes. For every **alt** node, we keep separate the sets coming from its subtrees. For every **un** node, we make the union of the sets coming from its subtrees, in every combination, if some subtree conveys alternative sets. Finally, we obtain the resulting sets $RS_1, \ldots, RS_n$, $n > 0$.

Proposition 4.1 *$RS_1, \ldots, RS_n$, $n > 0$ is a complete collection of base sets for A in P(A).*

$RS_1, \ldots, RS_n$ can be further reduced, by a simple check: if, for some atoms D, E, we have that $\{D, not\, E\} \in RS_i$, and $\{E, not\, D\} \in RS_j$, $i, j \leq n$, then $D, E, not\, D, not\, E$ can be canceled from both $RS_i and RS_j$. In fact, if a support set for A is obtained independently of the truth value of $DandE$, then no assumption on them is really needed, as shown by the following example.

Example 4.2 *Consider the program of Example 4.1. It is easy to see that the resulting sets for p are $\{a, not\, b\}$ and $\{b, not\, a\}$. Then, any truth value for a and b can support p, and then no assumption is needed. In fact, it is easy to see that the empty set is a support set for p in P.*

Example 4.3 *Consider the following program P.*

$q : -not\, p.$
$p : -not\, p, not\, a.$
$a : -not\, b, not\, c.$
$b : -not\, a.$
$c : -not\, d.$
$d : -not\, c.$

The unique stable model is $\{q, a, d\}$. The corresponding abductive hypothesis for q is $\Delta = \{not\, p, not\, b, not\, c\}$

The T–tree for q depends on the F-tree for p, which has the following structure.
Root **p:false**, *with single child* **un**, *OR–subtree empty, and AND–subtree with first child omitted, and second child which is the T–tree for a, with the following structure:*
root **a:true**, *OR–subtree empty, AND–subtree with root* **un** *and two children, the first one corresponding to the program node*
$a : -not\, b.$
$b : -not\, a.$
with no children, since this cycle is unbounded, and the second one corresponding to the F–tree for c, depending on the program node:
$c : -not\, d.$
$d : -not\, c.$

Then, the base models coming from the two leaves are $\{a, not\, b\}$ *and* $\{d, not\, c\}$ *respectively. Consequently, the unique base set for* q *is* $\{a, d\}$.

Now, it is necessary to perform on the resulting sets the coherence test defined in the previous section. At this point, we can choose to select either the positive or the negative part of the remaining RS_i's, thus obtaining the final sets, $BS_1, \ldots BS_j$, $j \geq 0$.

Theorem 4.1 *$BS_1, \ldots BS_j$ is a complete collection of base sets for A in P.*

Theorem 4.2 *$BS_1, \ldots BS_j$ is a minimal complete collection of base sets for A in P, in the sense that there does not exist any complete collection consisting of fewer and fewer and smaller sets.*

5 Concluding Remarks

In this paper we have discussed the problem of lemma generation with respect to a multimodel semantics, in particular with respect to the stable model semantics.

We have shown that criteria for storing conclusions as lemmas can be defined. However, there is a main difference with respect to asserting lemmas in first–order–logic, or in logic programming, with respect to a single–model semantics: the resulting program is not, in general, equivalent to the original one.

In fact, while asserting a conclusion, a choice is made, and consequently some possibilities are discarded. The particular choice is represented by the base set associated with that conclusion, which in a sense represents the "explanation" for that choice. The meaning of the resulting program corresponds to a restriction of the meaning of the original one.

We have noticed that the lemma generation problem for the stable model semantics is related to abduction, since abductive hypotheses for a conclusion are base sets for that conclusion. We have discussed however the possibility of finding smaller and fewer base sets, only for that kind of conclusions that really need a support (in fact, we show that atoms that are true in the well–founded model of the program, and atoms which belong to some even cycle do not really need explanations).

However, what is the relationship among the lemma generation problem and topics such as abduction, theory–revision and hypothetical reasoning remains to be explored, and is a main future topic of this research.

References

[1] Apt, K. R. and Bol, R., Logic programming and negation: a survey, *J. Logic Program.* 19/20 (1994).

[2] Chen, W. and Warren, D. S., Computation of Stable Models and its Integration with Logical Query Processing. IEEE Trans. on Knowledge and Data Engineering, forthcoming.

[3] Costantini, S., *Contributions to the Stable Semantics of Logic Programs with Negation.* Theoretical Computer Sc. 149 (1995). Extended abstract in:. A. Nerode (ed.), *Logic Programming and Non-Monotonic Reasoning*, Proceedings of the Second International Workshop. The MIT Press, 1993.

[4] Costantini, S., *On the Existence of Stable Models of Unstratified Programs.* Technical Report TR–158–96, Computer Science Dept., Univ. of Milano, January 1996.

[5] Dix, J., *Classifying Semantics of Logic Programs.* Proceedings of the *International Workshop on Logic Programming and Non-Monotonic Reasoning.* Washington, D.C., 1991.

[6] Dix, J., *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties.* Fundamenta Informaticae 22(3), 1995.

[7] Dix, J. *A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties.* Fundamanta Informaticae, forthcoming

[8] Eshghi, K., and Kowalski R.A., Abduction compared with negation as failure, in: G. Levi and M. Martelli (eds.), *Logic Programming, Proceedings of the Sixth International Conference*, MIT Press, Cambridge, MA,1989.

[9] Gelfond, M. *On Stratified Autoepistemic Theories.* Proceedings of AAAI'87, 1987.

[10] Gelfond, M., and Lifschitz, V., *The Stable Model Semantics for Logic Programming.* K. Bowen and R.A. Kowalski (eds.), *Logic Programming*, Proceedings of the Fifth Symposium, The MIT Press, 1988.

[11] *Proceedings of the Workshop on Structural Complexity and Recursion–Theoretic Methods in Logic Programming.* Washington, D.C., November 1992.

[12] Przymusinska, H., and Przymusinski, T. C., *Semantic Issues in Deductive Databases and Logic Programs.* R.B. Banerji (ed.) *Formal Techniques in Artificial Intelligence, a Sourcebook*, Elsevier Science Publisher B.V. (North Holland), 1990.

[13] Subrahmanian, V. S., Nau, D., and Vago, C., *WFS + Branch and Bound = Stable Models* CS-TR-2935 UMIACS-TR-92_82, July 1992 (submitted to IEEE Transactions on Knowledge and Data Engineering).

[14] Van Gelder, A., Ross, K. A., and Schlipf, J. S., *The Well-Founded Semantics for General Logic Programs.* J. of the ACM 38(3), 1991 (abstract in Proceedings of the Symposium on Principles of Database Systems, ACM SIGACT-SIGMOD, 1988).