Coherence of Updates in Answer Set Programming*

Stefania Costantini, Benedetto Intrigila
Dipartimento di Informatica, Università degli studi di L'Aquila
L'Aquila, I-67010 (Italy), {*stefcost, intrigil*}*@di.univaq.it*
Alessandro Provetti
Dipartimento di Fisica, Università degli studi di Messina
Messina, I-98166 (Italy), *ale@unime.it*

Abstract

In the context of logic programming under the Answer Set semantics, we propose to introduce a Software Engineering perspective, by discussing the properties that a program should exhibit after an update, i.e., after the addition/deletion of facts and rules during the software development process. In this sense, the property of Strong Equivalence has recently become of interest for its practical implications, by characterizing which program optimizations preserve equivalence w.r.t. arbitrary updates. In order to characterize how the answer sets change after updates that significantly modify the program, we introduce and discuss the new notion of Coherence. In particular, coherence of updates ensures that (some or all) the answer sets of the original program become proper subsets of the answer sets of the updated program. We study coherence in detail for the recently-proposed class of kernel logic programs, which have a simplified and uniform syntax, without loss of generality since every logic program has a kernel counterpart. For kernel programs, we define useful sufficient conditions for coherence.

1 Introduction

This paper aims at studying problems related to the developing, restructuring and updating of logic programs in Answer Set Programming (ASP). A lot of good research has been done for dealing with modifications to a knowledge base represented by a propositional theory under Answer Set Programming (a very good recent review with many references is [Alferes and Pereira, 2002]). The two main approaches are Theory Revision, which deals with incorporating new knowledge about a static world, and Theory Update, which deals with changing worlds. Thus, in these approaches a program changes because the available knowledge about the world changes at "run-time". Although indebted to all these approaches under many respects, we take the different perspective of software development, where a program changes because we are constructing it step by step, or because we are modifying or refining it, or trying to make it more concise and efficient.

To the best of our knowledge, the first attempt of considering Answer Set Semantics under such a perspective has been that of [Dix, 1995a; Dix, 1995b], that analyzes logical consequence under the stable model semantics from the point of view of the properties that a non–monotonic entailment relation should possess. It turns out that, in ASP, logical consequences of programs cannot in general be stored as lemmas. This is because the set of the answer sets of the resulting program may change. In fact, logical consequence under the Answer Set semantics does not enjoy an important property required of non–monotonic entailment relations: *Cumulativity*.

In [Costantini et al., 1996] this problem has been coped with by showing that it is possible to assert a conclusion A as a lemma, by asserting one or more sets of facts supporting the conclusion (called *base sets*). The effect on the meaning of the program is that of selecting the answer sets containing A. This has led to formulating a notion of Extended Cumulativity.

Another important aspect of logical consequence under the Answer Set semantics is the lack of the *Relevance* property: in order to establish whether atom A is true/false in one answer set, it does not suffice to consider the subprogram $P(A) = rel\_rul(P, A)$, consisting of all rules that could contribute to A's derivability, namely the set of rules concerning all atoms on which A depends (positively/ negatively, directly/indirectly). This is for two reasons. First, in some cases $P(A)$ has answer sets, while P has none. Second, A could be derivable in $P(A)$, but not derivable in P, since its truth would make P inconsistent.

The lack of relevance makes it more difficult to understand what happens when a program is updated by adding/deleting facts and rules. Clearly, it is of practical interest to understand under which conditions the updated program is still consistent, and which are the resulting answer sets. In this direction, in a recent set of articles ([Lifschitz et al., 2001], Pearce [Pearce, 1997], Cabalar [Cabalar, 2001], Otero [Otero, 2001],

*We acknowledge the support of MIUR 40% project *Aggregate- and number-reasoning for computing: from decision algorithms to constraint programming with multisets, sets, and maps* and of the *Information Society Technologies programme of the European Commission, Future and Emerging Technologies* under the IST-2001-37004 WASP project.

and others) some authors have introduced and studied the notion of Strong Equivalence of logic programs under Answer Set semantics. Two programs are equivalent if they have the same answer sets, while they are strongly equivalent if the programs resulting from adding the same set of fact and rules to each are still equivalent. In other words, we can see strong equivalence in terms of equivalence w.r.t. arbitrary updates. The rationale for studying strong equivalence is therefore to establish when can we simplify a subprogram, supposedly with the purpose of making it more concise, and thence easier to compute with, and maybe more readable, without altering the overall semantics of the program.

The topic that we discuss in this paper concerns the investigation of how the answer sets of a program change after an update that does not preserve equivalence, as it makes significant extensions/modifications to the given program. We introduce the new notion of *coherence* that relates the answer sets of the given program to the answer sets of the updated program. In particular, suppose one identifies certain conclusions as interesting. The *core answer set*,of an answer set S is defined as the intersection of S with the interesting atoms.

We will call an update *coherent* if it ensures that the core answer sets are *preserved*,as proper subsets of the answer sets of the updated program. Coherence will be defined both in a weak and a strong form.

An important issue is that of identifying sufficient conditions for coherence. As a first step, in this paper we address this issue for the class of kernel logic programs [Brignoli et al., 1999] [Costantini and Provetti, 2002], that due to their simple and uniform syntactic form allow us to understand in many significant cases which are the effects of an update.

The paper is organized as follows: after some preliminary definitions (Section 2), in Section 3 we introduce and discuss the notion of coherence of updates. In Section 4 we define kernel programs, and in Section 5 we introduce for this class of programs significant sufficient conditions for coherence. Finally, in Section 6 we discuss future developments, and in particular we give hints on how to go beyond kernel programs.

2 Background definitions

In this paper we consider the syntax of $DATALOG^{\neg}$ for deductive databases, which is more restricted than traditional logic programming (the reader may refer to [Marek and Truszczyński, 1999] for a discussion). In the following, we will implicitly consider the ground version of $DATALOG^{\neg}$ programs. A rule ρ is defined as usual, and can be seen as composed of a conclusion $head(\rho)$, and a set of conditions $body(\rho)$. The latter can be divided into positive conditions $pos(\rho)$ each one of the form A, and negative conditions $neg(\rho)$, each one of the form *not* A. In what follows, Π will denote a generic logic program. For the sake of simplicity, we assume that no two rules in Π with the same conclusion have the bodies in subset relation.

The answer sets semantics [Gelfond and Lifschitz, 1988] [Gelfond and Lifschitz, 1991] is a view of logic programs as sets of inference rules (more precisely, default inference rules). Alternatively, one can see a program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. Consider for instance the simple program $\{q \leftarrow \textit{not } p.\ \ p \leftarrow \textit{not } q.\}$: the first rule is read as "assuming that p is false, we can *conclude* that q is true." This program has two answer sets. In the first one, q is true while p is false; in the second one, p is true while q is false.

A subset M of the Herbrand base B_Π of a $DATALOG^{\neg}$ program Π is an answer set of Π, if M coincides with the least model of the reduct Π^M of Π with respect to M. This reduct is obtained by deleting from Π all rules containing a condition *not* a, for some a in M, and by deleting all negative conditions form the other rules. Answer sets are minimal supported models, and form an anti-chain. Whenever a program has no answer sets, we will say that the program is *inconsistent*. Correspondingly, checking for consistency means checking for the existence of answer sets.

The *Well-founded semantics* [Van Gelder et al., 1990] defines a three-valued Well-Founded model $WFS(\Pi) = \langle T, F\rangle$ that always exists, giving the sets of atoms which are (resp.) true and false. All the other atoms have, implicitly, truth value "undefined", and are crucial for finding the answer sets: in fact, every answer set extends set T, with no intersection with set F.

Definition 1 *A program* Π *is WFS-irreducible if and only if* $WFS(\Pi) = \langle \emptyset, \emptyset\rangle$.

It is easy to see that in WFS-irreducible programs there are no facts, that would belong to set T. Given an arbitrary logic program, it is possible to obtain a WFS-irreducible reduct by getting rid of atoms that are true/false w.r.t. the WFS [Costantini, 1992]. Several ASP solvers [ASP solvers] apply this kind of simplification, since the answer sets of the original program can be easily obtained from the answer sets of the reduct.

In the rest of the paper, an *update* is understood as a modification to the program performed by addition/deletion of facts/rules.

Definition 2 *Let* Π *be a logic program. A positive update (or just update)* Π_{up} *is a set of facts and/or rules to be added to* Π. *A negative update* Π^{-}_{up} *is a set of facts and/or rules to be deleted from* Π.

According to the above definition, for modifying a rule one has first to perform a negative update for deleting the rule, and then a positive update for adding the modified rule. Notice that, since the answer set semantics is non-monotonic, even positive updates may drop some previous consequences of a program. Take for instance simple program $p \leftarrow \textit{not } q$: update q makes p false.

If after an update there are two rules with the same conclusion and the bodies in subset relation, we assume that they are merged together, keeping the "shortest" one.

3 Toward a Theory of Updates: Coherence

In the early stages of program development, after modifying a program Π by performing update Π_{up}, the answer sets of the resulting program $\Pi' = \Pi \cup \Pi_{up}$ may well be very different in number and content from those of Π. In subsequent stages, however, one may want assume that the modification is *conservative*, i.e. that well-established, intended contents of some or all of the answer sets of Π are in some sense *preserved.* To illustrate the point, consider the development of the following simple program.

Example 1 *Let π_1 be*

$at\_work \leftarrow$ *not* at_home
$at\_home \leftarrow$ *not* at_work

with answer sets $\{at\_work\}$, $\{at\_home\}$. Suppose that we would like to obtain that one is happy at home, and bored at work. As a first attempt (made by a not-very-good programmer), update π_{up1} could be:

$happy \leftarrow$ *not* $happy$, *not* at_home

The resulting program has the unique answer set $\{at\_home\}$: one answer set is lost, the intended effect is not achieved.

As a second attempt, trying to fix the problem, update π_{up2} could be:

$happy \leftarrow$ *not* $happy$, *not* at_home
$bored \leftarrow$ *not* $bored$, *not* at_work

Unfortunately, the resulting program is inconsistent. Another update π_{up3} could be:

$happy \leftarrow$ *not* $bored$, *not* at_work
$bored \leftarrow$ *not* $happy$

Thence, the answer sets are: $\{at\_work, bored\}$, $\{at\_home, happy\}$ and $\{at\_home, bored\}$. There is an unwanted answer set!

Finally, update π_{up4}

$happy \leftarrow$ *not* $bored$, *not* at_work
$bored \leftarrow$ *not* $happy$, *not* at_home

gets the correct effect, with answer sets $\{at\_work, bored\}$, $\{at\_home, happy\}$.

The example above shows that a formal theory of updates is really needed. The aim of this paper is to start lying the foundations of such theory and practice.

Although one cannot in general look into the intended meaning of programs, we believe that in many practical cases it is possible to identify, for some or all of the answer sets of Π, an interesting subset corresponding to intended conclusions that we want to preserve. We will call these subsets *core answer sets.*

The property of *coherence* of update Π_{up} is defined as the preservation of core answer sets as proper subsets of the answer sets of the updated program. Since an answer set can coincide with its core, For the sake of simplicity, in the examples we will assume that this is the case. Also, in what follows let $\mathcal{A}(\Pi)$ be the set of the answer sets of given program Π.

Definition 3 *Let $\mathcal{A}(\Pi) = \{S_1, \ldots, S_r\}$. $\mathcal{S}(\Pi) = \{M_1, \ldots, M_n\}$ is a set of core answer sets for Π if for each $M_i \in \mathcal{S}(\Pi)$ there exists $S_j \in \mathcal{A}(\Pi)$ such that $M_i \subseteq S_j$. We say that M_i is a core answer set of S_j.*

Below we introduce the novel notion of Coherence of updates. Firstly, we consider the effect of an update on a single answer set. In particular, Local Coherence requires that the core M of answer set S of program Π be preserved (though possibly extended) after update Π_{up}.

Definition 4 (Local Coherence) *Given program Π and core answer set $M \in \mathcal{S}(\Pi)$, an update Π_{up} of Π is* locally coherent *w.r.t. Π and M if there exists $M' \in \mathcal{A}(\Pi \cup \Pi_{up})$ such that $M \subseteq M'$.*

Example 2 (Local Coherence) *Let π_2 be*

$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

with (core) answer sets $\{a\}$ and $\{b\}$. Given update π_{up}

$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

the resulting program has answer sets $\{a, e\}$, $\{a, f\}$, $\{b, e\}$, $\{b, f\}$, where each answer set of π_2 is a subset of some answer set of the updated program. Then, update π_{up} is locally coherent w.r.t. both answer sets of π_2.

Notice that local coherence of Π_{up} requires that $\Pi \cup \Pi_{up}$ is consistent, which is not always the case, and may be very expensive (computationally) to check. Take for instance program $\{p \leftarrow$ *not* $p, p \leftarrow$ *not* $a\}$ which is consistent, with unique answer set $\{p\}$: update consisting simply in the addition of fact $a \leftarrow$ produces an inconsistent program, while update $\{a \leftarrow, p \leftarrow$ *not* $b\}$ produces a consistent program with unique answer set $\{p, a\}$.

A more general notion of coherence concerns the whole set of the answer sets of the given program Π, and requires that some or all the core answer sets Π be preserved (though possibly extended) after update Π_{up}. Let us consider an intermediate formulation where we require that all the answer sets of the updated program extend some of the answer sets of the original program. That is, the updated program is allowed to have any number of answer set, even fewer than the original program. This definition is useful in those cases where some unwanted models have to be dropped (typically by adding constraints to the program).

Definition 5 (Weak Coherence) *An update Π_{up} of program Π is* weakly coherent *w.r.t. Π if for each $M' \in \mathcal{A}(\Pi \cup \Pi_{up})$ there exists $M \in \mathcal{S}(\Pi)$ such that $M \subseteq M'$.*

Example 3 (Weak Coherence) *Let π_3 be*

$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

Let update π_{up} consist of the rule

$q \leftarrow$ *not* q, *not* b

Update π_{up} is weakly coherent w.r.t. π_3: while π_3 has (core) answer sets $\{a\}$ and $\{b\}$, $\pi_3 \cup \pi_{up}$ has the unique answer set $\{b\}$. This update in fact consists in a constraint that states that b must be true (otherwise, the program is inconsistent) and thus rules out every answer set not including b. Constraints are widely used in Answer Set Programming and in fact all the answer set solvers allow to express them in the compact form $\leftarrow$ not b.

We now define the stronger notion of Global coherence, that requires the answer sets of the updated program to extend all the core answer sets of the original program. This definition works in those cases where all the answer sets of the original program contain relevant consequences that have to be preserved.

Definition 6 (Global Coherence) *An update Π_{up} to program Π is* globally coherent *w.r.t. Π if for each $M \in \mathcal{S}(\Pi)$ there exists $M' \in \mathcal{A}(\Pi \cup \Pi_{up})$ such that $M \subseteq M'$.*

Example 4 (Global Coherence) *Let π_4 be*

$a \leftarrow$ *not* b
$b \leftarrow$ *not* a

with (core) answer sets $\{a\}$ and $\{b\}$. Given update π_{up}

$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

the resulting program has answer sets $\{a, e\}$, $\{a, f\}$, $\{b, e\}$, $\{b, f\}$, where each answer set of π_4 is a subset of some answer set of the updated program.

Global coherence as defined above allows $\Pi \cup \Pi_{up}$ to have more answer sets than Π. A rigid position where the number of the answer sets of $\Pi \cup \Pi_{up}$ is required to be the same as the number of the answer sets of Π is formalized by the following notion.

Definition 7 (Strong Coherence) *An update Π_{up} of program Π is* strongly coherent *w.r.t. Π if it is globally coherent, and the cardinality of $\mathcal{A}(\Pi)$ and $\mathcal{A}(\Pi \cup \Pi_{up})$ is the same.*

Checking coherence of updates is a challenging problem, related to the more general one of existence and number of answer sets. As a first step, in what follows we will show that by referring to a limited though in our opinion significant class of logic programs, relevant sufficient conditions for coherence can be defined.

4 Preliminary definitions: Kernel Programs

Kernel logic programs (or simply *kernels*) have a simple uniform structure, and are actually a normal form any logic program can be reduced to [Costantini, 1995] [Costantini and Provetti, 2002]. In this context however, we take kernels just as a relevant class of programs for which sufficient conditions for coherence can be specified in an elegant and concise way, although our future objective is to extend these conditions to more general classes of programs.

Kernel programs are WFS-irreducible, and the body of their rules is composed of negative literals only.

Example 5 *To informally illustrate which are the elements a kernel program is composed of, we propose the following sample program.*

1. $a \leftarrow$ *not* b
2. $b \leftarrow$ *not* a
3. $c \leftarrow$ *not* f
4. $f \leftarrow$ *not* g, *not* a
5. $g \leftarrow$ *not* e
6. $p \leftarrow$ *not* p, *not* g
7. $f \leftarrow$ *not* b

Rules 1 and 2 form a negative even cycle, *since atom a depends (negatively) upon not b, and vice versa. In the rest of this paper we consider only negative dependencies, and thus only negative cycles, that for short we call just* cycles. *This cycle is* even, *since it involves an even number of atoms. Rules 3, 4, and 5 form an* odd *cycle, that involves the three atoms e, f and g. Literal not a which occurs in a rule of the cycle, where however atom a is not involved in the cycle itself, is called an AND handle of the cycle. Rule 6 is a particular case of an odd cycle with an AND handle, namely not g, since it involves just one rule: it is called a* self-loop. *Rule 7 does not belong to any cycle, although its head f is involved in the (odd) cycle consisting of rules 3, 4 and 5. This kind of rule is called an* auxiliary rule *of that cycle and the literal not b which occurs in its body is called an* AND handle *of the cycle.*

Handles constitute the *connections* between cycles. As we will see later, looking at these connections will allow us to understand (at least in some cases) what is the effect of an update. Formally:

Definition 8 *A set of rules C is called a cycle if it has the following form:*

$\lambda_1 \leftarrow$ *not* $\lambda_2, \Delta_1, \ldots, \lambda_n \leftarrow$ *not* λ_1, Δ_n

where the λ_i's are distinct atoms, $n \geq 1$. Each Δ_i is a (possibly empty) conjunction $\delta_{i1}, \ldots, \delta_{ih}$ of literals (either positive or negative) and for each δ_{ij}, $\delta_{ij} \neq \lambda_i$ and $\delta_{ij} \neq$ not λ_i. The Δ_i's are called the AND handles of the cycle. We say that Δ_i is an AND handle for (or referring to) atom λ_i.

For $n = 1$ we have the self-loop $\lambda_1 \leftarrow$ *not* λ_1, Δ_1. We say that C has size n and it is even (respectively odd) if $n = 2k$, $k \geq 1$ (respectively, $n = 2k + 1$, $k \geq 0$). We call $Composing\_atoms(C)$ the set containing all the atoms *involved* in cycle C. We say that the rules listed above *belong*to the cycle, or *form* the cycle.

Definition 9 *A rule is called an* auxiliary rule of cycle C *(or, equivalently,* to *cycle C) if it is of this form:*

$\lambda_i \leftarrow \Delta$

where $\lambda_i \in$ Composing_Atoms(C), and Δ is a non-empty conjunction $\delta_{i1}, \ldots, \delta_{ih}$ of literals (either positive or negative) and for each δ_{ij}, $\delta_{ij} \neq \lambda_i$ and $\delta_{ij} \neq$ not λ_i. Δ is called an OR handle of cycle C (more specifically, an OR handle for, or referring to, λ_i). A cycle may possibly have several auxiliary rules, corresponding to different OR handles. It may

be the case that different auxiliary rules have the same body, although referring to different atoms among those involved in the cycle.

Definition 10 *A logic program* Π *is in* kernel normal form *(or, equivalently,* Π *is a* kernel program*, or for short just a* kernel*) if and only if:*

1. Π *is WFS-irreducible (and thus there are no facts);*
2. *every rule has its body composed of negative literals only;*
3. *every atom in* Π *occurs both in the head and in the body of some rule;*
4. *every rule either belongs to a cycle, or is auxiliary to a cycle.*

5 Technical Results about Coherent Updates

An update Π_{up} is not required to be a kernel. Even if Π is a kernel, the result $\Pi' = \Pi \cup \Pi_{up}$ of the update is in general not in kernel form. In the rest of this section however, for the sake of simplicity we assume that $\Pi \cup \Pi_{up}$ is still a kernel.

We make the following further simplifying assumptions, that we mean to remove in the future developments of this research: we assume that an answer set coincides with its core, we consider positive updates only, and we suppose that Π', is consistent.

As it is well-known (see for instance [Costantini, 1995]), in an answer set M every true atom a is supported, in the sense that there exists a rule in the program with head a, and body true w.r.t. M. In kernel programs, since the body is composed of negative literals only, this is equivalent to say that all the atoms occurring in the body must be false w.r.t. M. In the above definition, we build the set of all the atoms that potentially support (by being false) a given atom a, by collecting them in the bodies of rules with head a.

Definition 11 *Given (core) answer set* M *of* Π*, and given atom* $a \in M$*, let* $\{a \leftarrow Body_1, \ldots, a \leftarrow Body_n\}$ *be the rules with head* a *and body which is true in* M*. The set* $Support(a)$ *is composed of all the atoms occurring in* $Body_1, \ldots, Body_n$*.*

The set $Support(a)$ is of interest for coherence, since a potential source of problems is that one atom belonging to $Support(a)$, that is false w.r.t. M, becomes true due to the rules of Π_{up}, thus potentially leaving a unsupported. This would imply the risk that a cannot be true in the answer sets of Π', which means that Π_{up} is not locally coherent for M.

For any given kernel program Π, let:

(a) $heads(\Pi)$ be the set of atoms occurring as the heads of the rules of Π;

(b) $Ands(\Pi)$ be the set of atoms occurring in the AND handles of the rules of Π;

(c) $Ors(\Pi)$ be the set of atoms occurring in the OR handles of the rules of Π.

Since Π_{up} in general is not a kernel, we cannot properly define $Ands(\Pi_{up})$ and $Ors(\Pi_{up})$. However, given $\Pi' = \Pi \cup \Pi_{up}$ and thus given $Ands(\Pi')$ and $Ors(\Pi')$, by abuse of notation we indicate with $Ands(\Pi_{up})$ and $Ors(\Pi_{up})$ their subsets, corresponding to the AND and OR handles of Π' occurring in its subprogram Π_{up}.

The handles of Π_{up} are of interest for coherence, since, as illustrated in the example below, if an atom occurs in an handle of Π_{up}, it may *have to* change its truth value in the answer sets of Π' (the reader may refer to [Costantini, 2003] for a discussion). Below we give a sufficient condition to ensure that Π_{up} is a locally coherent update w.r.t. (core) answer set M of Π. This condition can then be used for checking either weak or global coherence.

Proposition 1 *Let* Π *be a kernel and* Π_{up} *an update such that* $\Pi \cup \Pi_{up}$ *is still a kernel.* Π_{up} *is a locally coherent update for* Π *with respect to core answer set* M *of* Π*, if the following conditions hold:*

(i) for every $a \in M$, $Head(\Pi_{up}) \cap Support(a) = \emptyset$*;*

(ii) $M \cap Ors(\Pi_{up}) = \emptyset$*;*

(iii) for every $a \in M$, $Ands(\Pi_{up}) \cap Support(a) = \emptyset$*.*

Proof 1 *(sketch) If atom* a *belonging to answer set* M *is supported in* M *for instance by rule* $a \leftarrow$ *not* b*,not* c*, condition (i) forbids* Π_{up} *from containing new clauses for* b *and* c*, otherwise they might become true in the answer sets of* $\Pi' = \Pi \cup \Pi_{up}$*, thus leaving* a *without support.*

Condition (ii) instead, ensures that a *which is true in* M *is not forced to become false in the answer sets of* Π' *for keeping* Π' *consistent.*

Finally, condition (iii) ensures that no atom that supports a *in* M *(by being false) is forced to be assumed true for keeping* Π' *consistent.*

The conditions are illustrated by the following example, where again we assume that the answer sets of given program coincide with their core.

Example 6 *Let* π_5 *be*

$a \leftarrow$ *not* b
$b \leftarrow$ *not* a
$p \leftarrow$ *not* p*,not* b*.*
$e \leftarrow$ *not* f
$f \leftarrow$ *not* e

with (core) answer sets $\{b, e\}$ *and* $\{b, f\}$*. We consider below three different updates, each one not satisfying one of the conditions (i)-(iii) for answer set* $\{b, e\}$*, while satisfying all the conditions for answer set* $\{b, f\}$*. We will show that in fact each of the three updates is not locally coherent w.r.t.* $\{b, e\}$*, while it is locally coherent w.r.t.* $\{b, f\}$*.*

The update π_{up1}

$f \leftarrow$ *not* a

does not satisfy condition (i) for answer set $\{b, e\}$ *of* π_5*, since* $Support(e) = \{f\}$ *and there are clauses for* f *in this update. In fact, the updated program* $\pi_5^1 = \pi_5 \cup \pi_{up1}$ *has unique*

answer set $\{b, f\}$, and thus π_{up1} is not locally coherent w.r.t. $\{b, e\}$ while it is locally coherent w.r.t. $\{b, f\}$.

The update π_{up2}

$q \leftarrow$ *not* q
$q \leftarrow$ *not* e.

does not satisfy condition (ii) for answer set $\{b, e\}$ of π_5, because e occurs in $Ors(\pi_{up2})$. Here, assuming e true would mean to have an inconsistent self-loop on q. Then, the updated program $\pi_5^2 = \pi_5 \cup \pi_{up2}$ cannot have answer sets where e is true. In fact, π_{up2} is not locally coherent w.r.t. $\{b, e\}$, since π_5^2 it has the unique answer set $\{b, f, q\}$.

The update π_{up3}

$q \leftarrow$ *not* q, *not* f

does not satisfy condition (iii) for answer set $\{b, e\}$ of π_5, because $Support(e) = \{f\}$ and f occurs in $Ands(\pi_{up3})$. Here, f must be true for avoiding an inconsistent self-loop on q. Then, the updated program π_5^3 cannot have answer sets where f is false. On the other hand however, f should be false for supporting e. In fact, π_{up3} is not locally consistent for $\{b, e\}$, since π_5^3 has unique answer set $\{b, f\}$.

Finally, the update π_{up4}

$p \leftarrow$ *not* a

satisfies all the three conditions, and consequently is locally coherent, for both the answer sets of π_5. In fact, the updated program $\pi_5^4 = \pi_5 \cup \pi_{up4}$ has answer sets $\{b, e, p\}$ and $\{b, f, p\}$. Notice that this update is strongly coherent, since the number of answer sets remains the same.

It is possible in perspective to define finer conditions, going deeper into the syntactic form of Π and Π_{up}. It is however important to notice that the conditions stated in the above proposition work in many practical cases, and are easy to check.

6 Concluding Remarks: beyond kernels

In this paper we have proposed the new notion of coherence, for relating the programs produced in subsequent steps of a software development process in Answer Set Programming. We have defined various notions of coherence, that establish what relation there should be between the answer sets of the original programs, and the answer sets of the programs resulting from update/modification. We have introduced significant sufficient conditions for coherence on the class of *kernel* programs, that are a normal form of logic programs that allow one to distinguish the elements the program is composed of, namely cycles and handles, and thus to reason more easily about what happens when the program is modified. A more difficult issue is that of investigating conditions for coherence in wider classes of programs. This would require a careful analysis, in order to identify for more general cases the elements a program is composed of, and how they are related.

Essentially, the elements are still the same: the answer sets of any program can be easily obtained from the answer sets of its kernel [Costantini and Provetti, 2002], and the kernel always exists and is unique, which means that the kernel contains all the relevant information. Nevertheless, one has to cope with the indirectness introduced by positive atoms in the bodies of rules, by "bridges" between cycles, and by atoms which occur in the heads but not in the bodies of rules. Such a deeper analysis of the inherent structure of arbitrary programs is a topic for future research.

References

[Alferes and Pereira, 2002] J. J. Alferes, L. M. Pereira. *Logic Programming Updating - A Guided Approach*, Computational Logic: Logic Programming and Beyond, Essays in Honor of Robert A. Kowalski, Part II, LNAI 2408, Springer-Verlag, Berlin: 382–412, 2002.

[Brignoli et al., 1999] G. Brignoli, S. Costantini, O. D'Antona, and A. Provetti. *Characterizing and Computing Stable Models of Logic Programs: the Non–stratified Case*. Proc. of 1999 Conference on Information Technology, held in Bhubaneswar, India, December 1999.

[Cabalar, 2001] P. Cabalar. *Well-founded Semantics as Two-dimensional Here and There*. Proc. of AAAI Spring Symposium ASP2001, AAAI Press, Tech. report SSS01, 2001.

[Costantini, 1992] S. Costantini. *Contributions to the Stable Model semantics of Logic Programs with Negation*. in: A. Nerode and V.S. Subrahmanian (eds.), "Logic Programming and Non-Monotonic Reasoning", Proceedings of the 2nd International Workshop. The MIT Press, USA, 1993.

[Costantini, 1995] S. Costantini. *Contributions to the stable model semantics of logic programs with negation*. Theoretical Computer Science, 149: 231-255, 1995.

[Costantini et al., 1996] S. Costantini, G. A. Lanzarone, G. Magliocco. *Asserting Lemmas in the Stable Model Semantics*. Logic Programming: Proc. of the 1996 Joint International Conference and Symposium (held in Bonn, Germany, September 1996). The MIT Press, USA, 1996.

[Costantini, 2003] S. Costantini. *On the Existence of Stable Models of Non-stratified Logic Programs*. Technical Report Univ. of L'Aquila, submitted.

[Costantini and Provetti, 2002] S. Costantini, and A. Provetti. *Normal Forms for Answer Set Programming*. Technical Report Univ. of L'Aquila, submitted.

[Dix, 1995a] J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties*. Fundamenta Informaticae XXII(3): 227–255, 1995.

[Dix, 1995b] J. Dix. *A Classification Theory of Semantics of Normal Logic Programs: I. Weak Properties*. Fundamenta Informaticae XXII(3) : 257–288, 1995.

[Gelfond and Lifschitz, 1988] M. Gelfond, and V. Lifschitz. *The stable model semantics for logic programming*. Proc. of 5th ILPS conference : 1070–1080.

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

[Lifschitz et al., 2001] V. Lifschitz, D. Pearce, and A. Valverde. *Strongly Equivalent logic programs*. ACM Transactions on Computational Logic, 2:526–541.

[Marek and Truszczyński, 1999] W. Marek, and M. Truszczyński. *Stable models and an alternative logic programming paradigm*, The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag: 375–398.

[Otero, 2001] R. Otero. *Pertinence Logic Characterization of Stable Models*. Proc. AAAI Spring Symposium ASP2001, AAAI Press, Tech. report SSS01, 2001.

[Pearce, 1997] D. Pearce. *A New Logical Characterization of Stable Models and Answer Sets*. Non-Monotonic Extensions of Logic Programming, Springer-Verlag, Berlin, LNAI 1216: 55–70.

[ASP solvers] Web location of some ASP solvers:
CCALC: *http://www.cs.utexas.edu/users/mcain/cc*
Cmodels: *http://www.cs.utexas.edu/users/tag/cmodels.html*
DeReS: *http://www.cs.engr.uky.edu/~lpnmr/DeReS.html*
DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*
NoMoRe: *http://www.cs.uni-potsdam.de/~linke/nomore/*
SMODELS: *http://www.tcs.hut.fi/Software/smodels/*

[Turner, 2001] H. Turner. *Strong Equivalence for Logic Programs and Default Theories (Made Easy)*. *Logic Programming and NonMonotonic Reasoning*, Springer-Verlag, Berlin, LNAI 2173: 81–92.

[Van Gelder et al., 1990] A. Van Gelder A., K. A. Ross, and J. Schlipf J. *The Well-Founded Semantics for General Logic Programs*. Journal of the ACM 38(3).