Generalizing updates: from models to programs

João Alexandre Leite[*][†]and **Luís Moniz Pereira**[†]
{jleite|lmp}@di.fct.unl.pt
Centro de Inteligência Artificial (CENTRIA)
Departamento de Informática
Universidade Nova de Lisboa
2825 Monte da Caparica
Portugal

Abstract

Recently the field of theory update has seen some improvement, in what concerns model updating, by allowing updates to be specified by so-called revision programs. The updating of theory models is governed by their update rules and also by inertia applied to those literals not directly affected by the update program. Though this is important, it remains necessary to tackle as well the updating of programs specifying theories. Some results have been obtained on the issue of updating a logic program which encodes a set of models, to obtain a new program whose models are the desired updates of the initial models. But here the program only plays the rôle of a means to encode the models.

A logic program encodes much more than a set of models: it encodes knowledge in the form of the relationships between the elements of those models. In this paper we advocate that the principle of inertia is advantageously applied to the rules of the initial program rather than to the individual literals in a model. Indeed, we show how this concept of program update generalizes model or interpretation updates. Furthermore, it allows us to conceive what it is to update one program by another, a crucial notion for opening up a whole new range of applications concerning the evolution of knowledge bases. We will consider the updating of normal programs as well as these extended with explicit negation, under the stable semantics.

Keywords: Updates

[*]Partially supported by PRAXIS XXI scholarship no. BM/437/94.
[†]Partially supported by project MENTAL (PRAXIS XXI 2/2.1/TIT/1593/95).

1 Introduction and Motivation

When dealing with modifications to a knowledge base represented by a propositional theory, two kinds of abstract frameworks have been distinguished by Katsuno and Mendelzon in [KM91]. One, theory revision, deals with incorporating new knowledge about a static world. The other, dealing with changing worlds, is known as theory update. This paper concerns only theory update.

So far, most of the work accomplished in the field of theory update [PT95] [MT94] [KM91]has addressed the modification of models on a one by one basis, by allowing updates to be specified by so-called revision programs. The field of theory update has seen several major achievements, namely the embedding of revision programs into logic programs [MT94], arbitrary rule updates and, the embedding into default logic [PT95].

The update of models is governed by update rules and also by inertia applied to the literals not directly affected by the update program. Though this is important, it remains necessary to tackle as well the updating of programs specifying theories, as opposed to updating its models. Some results have been obtained on the issue of updating a logic program which encodes a set of models, to obtain a new program whose models are the desired justified updates of the initial models [AP97]. But here the program only plays the rôle of a means to encode the models.

A logic program encodes much more than a set of models: it encodes knowledge in the form of the relationships between the elements of those models. In this paper we advocate that the principle of inertia is advantageously applied to the rules of the initial program rather than to the individual literals in a model. Indeed, we show how this concept of program update generalizes model or interpretation updates. Furthermore, it allows us to conceive what it is to update one program by another. A crucial notion for opening up a whole new range of applications concerning the evolution of knowledge bases. We will consider the updating of normal programs as well as these extended with explicit negation, under the stable semantics.

To show that a logic program encodes relationships between the elements of a model, which are lost if we simply envisage updates on a model by model basis, as proposed in [KM91], consider the following situation where an alarm signal is present:

Example 1 *Take the normal program P and its model M:*

$$\begin{array}{ll} P: & sleep \leftarrow not\ alarm \\ & panic \leftarrow alarm \\ & alarm \leftarrow \end{array}$$

$$M = \{alarm, panic\}$$

Now consider an update program stating that the alarm goes off:

$$U: \quad out(alarm) \leftarrow$$

According to [MT94] and model updating we obtain as the single justified update of M the following model:

$$M_U = \{panic\}$$

Stating that, although we know that the alarm is off, we are in a state of panic and not asleep. In [AP97] the authors propose a program transformation that produces a new program whose models are exactly the justified revisions of the models of the initial program, according to the definition proposed in [MT94], and so produces exactly the result above.

But looking at the program and at the update program, we arguably conclude that M_U doesn't represent the intended meaning of the update of P by U for a commonsensical reasoner. Since "panic" was true because the "alarm" was on, the removal of "alarm" should make one expect "panic" to become false. The same kind of reasoner expects "sleep" to become true. The intended update model of the example presumably is:

$$M_U^{'} = \{sleep\}$$

Another symptomatic example, but using explicit negation is this:

Example 2 *Given the statements:*

- *If I've seen something that is unexplainable then I've seen a miracle.*
- *If I've seen a miracle then God exists.*
- *I've seen something.*
- *It is not explainable.*

They can be represented by the following extended logic program:

$$\begin{array}{ll} P: & seen\_miracle \leftarrow seen\_something, not\ explainable \\ & god\_exists \leftarrow seen\_miracle \\ & seen\_something \leftarrow \\ & \neg explainable \leftarrow \end{array}$$

whose answer-set M is:

$$M = \{seen\_something, \neg explainable, seen\_miracle, god\_exists\}$$

Now consider the following update program U stating that we now have an explanation:

$$U: \quad in(explainable) \leftarrow$$

According to model updating we obtain as the single justified update of M the following model M_U:

$$M_U = \{seen\_something, explainable, seen\_miracle, god\_exists\}$$

Once again we arguably conclude that this model doesn't represent the intended meaning and that the correct model should be:

$$M_U = \{seen\_something, explainable\}$$

The purpose of this paper is to generalize model updates to logic program updates. The former are a special case of the latter since they can be coded as factual programs. To do this we must first consider the rôle of inertia in updates.

Newton's first law, also known as the law of inertia, states that: *"every body remains at rest or moves with constant velocity in a straight line, unless it is compelled to change that state by an unbalanced force acting upon it"* (adapted from [Principia]). One often tends to interpret this law in a commonsensical way, as things keeping as they are unless some kind of force is applied to them. This is true but it doesn't exhaust the meaning of the law. It is the result of all applied forces that governs the outcome. Take a body to which several forces are applied, and which is in a state of equilibrium due to those forces canceling out. Later one of those forces is removed and the body starts to move.

The same kind of behaviour presents itself when updating programs. Let us make the parallel between a program rule and a physical body with forces applied to it, the body of the rule being the forces applied to the head. In the same way we have to determine whether the forces are still in a state of equilibrium, before concluding that a physical body is at rest or moves with constant velocity in a straight line due to inertia, when it comes to the updating of a program we have to check if the truth value of a body which determines the truth value of a head hasn't changed before concluding the truth value of the head by inertia. This is so because the truth value of the body may change due to an update rule.

Going back to the previous example, before stating that "*god_exists*" is true by inertia since it wasn't directly affected by the update program, one should verify for instance whether "*explained*" is still not true, for otherwise there would be no longer a way to prove "*god_exists*" and therefore its truth value would no longer be 'true'.

To conclude, we argue that the truth of any element in the updated models should be supported by some rule, i.e. one with a true body, either of the update program or of the given program, in face of new knowledge.

The remainder of this paper is structured as follows: in section 2 we recapitulate some background concepts necessary in the sequel; in section 3 we formalize the normal logic program update process and present a transformation, reminiscent of the one in [AP97], providing the intended results; we conclude the section by showing that the transformation generalizes the one set forth in [PT95]; in section 4 we extend our approach to the case where the program to be updated is a logic program extended with explicit negation, and in section 5 we conclude and elaborate on future developments.

2 Review of Interpretation Updates

In this section we summarize some of the definitions related to the issue of theory update. Some of these definitions will be slightly different, though equivalent to the original ones, with the purpose of making their relationship clearer.

For self containment and to eliminate any confusion between updates and revisions, instead of using the original vocabulary of revision rule, revision program and justified revision, we will speak of update rule, update program and justified update, as in [AP97].

The language used is similar to that of logic programming: update programs are collections of update rules, which in turn are built out of atoms by means of the special operators: $\leftarrow$, *in*, *out*, and ",".

Definition 3 (Update Programs) *[MT94] Let U be a countable set of atoms. An update in-rule or, simply, an in-rule, is any expression of the form:*

$$in(p) \leftarrow in(q_1), ..., in(q_m), out(s_1), ..., out(s_n) \tag{1}$$

where p, q_i, $1 \leq i \leq m$, and s_j, $1 \leq j \leq n$, are all in U, and m, $n \geq 0$.

An update out-rule or, simply, an out-rule, is any expression of the form:

$$out(p) \leftarrow in(q_1), ..., in(q_m), out(s_1), ..., out(s_n) \tag{2}$$

where p, q_i, $1 \leq i \leq m$, and s_j, $1 \leq j \leq n$, are all in U, and m, $n \geq 0$. $\diamondsuit$

A collection of in-rules and out-rules is called an update program *(UP).*

Definition 4 (Necessary Change) *[MT94] Let P be an update program with least model M (treating P as a positive Horn program). The* necessary change *determined by P is the pair (I_P, O_P), where*

$$I_P = \{a : in(a) \in M\} \quad O_P = \{a : out(a) \in M\}$$

Atoms in I_P (resp. O_P) are those that must become true (resp. false). If $I \cap O = \{\}$ then P is said coherent. $\diamondsuit$

Intuitively, the necessary change determined by a program P specifies those atoms that must be added and those that must be deleted as a result of a given update, whatever the initial interpretation.

Definition 5 (P-Justified Update) *[MT94] Let P be an update program and I_i and I_u two total interpretations. The reduct $P_{I_u|I_i}$ with respect to I_i and I_u is obtained by the following operations:*

- Removing from P all rules whose body contains some $in(a)$ and $a \notin I_u$;

- Removing from P all rules whose body contains some $out(a)$ and $a \in I_u$;

- Removing from the body of any remaining rules of P all $in(a)$ such that $a \in I_i$;

- Removing from the body of any remaining rules of P all $out(a)$ such that $a \notin I_i$.

Let (I, O) be the necessary change determined by $P_{I_u|I_i}$. Whenever $P_{I_u|I_i}$ is coherent, I_u is a P-justified update *of I_i with respect to P iff the following stability condition holds:*

$$I_u = (I_i - O) \cup I \quad \diamondsuit$$

The first two operations delete rules which are useless given I_u. The stability condition preserves the initial interpretation in the final one as much as possible.

3 Normal Logic Program Updating

As we've seen in the introduction, updating on the basis of models isn't enough if we want to take advantage of the information encoded by a logic program and not expressed in the set of its models.

When we generalize the notion of p-justified update, from interpretations to the new case where we want to update programs, the resulting update program should be made to depend only on the initial program and on the update program, but not on any specific initial interpretation. An interpretation should be a model of a normal logic program updated by an update program if the truth of each of its literals is either supported by a rule of the update program with true body in the interpretation or, in case there isn't one, by a rule of the initial program whose conclusion is not contravened by the update program.

Another way to view program updating, and in particular the rôle of inertia, is to say that the rules of the initial program carry over to the updated program, due to inertia, instead of the truth of interpretation literals as in [AP97], just in case they are not overruled by the update program. This is to be preferred because the rules encode more information than the literals. Inertia of literals is a special case of rule inertia since literals can be coded as factual rules. Accordingly, program updating generalizes model updating.

To achieve rule inertia we start by defining the sub-program of the initial program which contains the rules that should persist in the updated program due to inertia. We use this program together with the update program to characterize the models of the resulting updated program, i.e. the program-justified updates, whatever the updated program may be. Finally, we present a joint program transformation of the initial and the update programs, which introduces inertia rules, to produce an updated program whose models are the required program-justified updates. Stable model semantics and its generalization to extended logic programs [GL90] will be used to define the models of programs.

We start by defining a translation of an update program over a language that does not contain explicit negation, into a normal logic program extended with explicit negation.

Definition 6 (Interpretation Restriction) *Given a language L that does not contain explicit negation $\neg$. Let $M_\neg$ be an interpretation, over the language $L_\neg$, obtained by augmenting L with the set $E = \{\neg A : A \in L\}$.*

We define the corresponding restricted interpretation M*, over* L*, as:*

$$M = M_\neg \textit{ restricted to } L \ \diamondsuit$$

Definition 7 (Translation of UPs into LPs) *Given an update program* UP*, over a language* L*, its* translation into an extended logic program U *over* $L_\neg$ *is obtained from* UP *by replacing each in-rule (1) with the corresponding rule:*

$$p \leftarrow q_1, ...q_m, not\ s_1, ..., not\ s_n$$

and similarly replacing each out-rule (2) with the corresponding rule:

$$\neg p \leftarrow q_1, ...q_m, not\ s_1, ..., not\ s_n \ \diamondsuit$$

¿From now onwards, and unless otherwise stated, whenever we refer to an update program we mean its reversible translation into an extended logic program according to the previous definition. Notice that such programs do not contain explicitly negated atoms in the body of its rules.

Definition 8 (Inertial Sub-Program) *Let* P *be a normal logic program over* L*,* U *an update program over* $L_\neg$ *and* $M_\neg$ *a model of* U*. Let:*

$$\begin{aligned} Rejected(M_\neg) = \ & \{A \leftarrow body \in P : M_\neg \models body \\ & and\ \exists \neg A \leftarrow body' \in U : M_\neg \models body'\ \} \end{aligned}$$

where A *is an atom. We define* Inertial Sub-Program $P_{inertial}(M_\neg)$ *as:*

$$P_{inertial}(M_\neg) = P - Rejected(M_\neg) \ \diamondsuit$$

Intuitively, the rules for some atom A that belong to $Rejected(M_\neg)$ are those that belong to the initial program but, although their body is still verified by the model, there is an update rule that overrides them, by contravening their conclusion.

Definition 9 (<P,U>-Justified Updates) *Let* P *be a normal logic program,* U *an update program, and* M *an interpretation over the language of* P*.* M *is a* <P,U>-Justified Update *of* P *updated by* U *iff* $M_\neg$ *is an answer-set of* P^**, where*

$$P^* = P_{inertial}(M_\neg) + U \ \diamondsuit$$

Notice that the new definition of program-justified update doesn't depend on any initial model. Once again this is because inertia applies to rules and not model literals. To achieve inertia of model literals it is enough to include them as fact rules, as shown in the sequel.

The following example will show the rôle played by $Rejected(M_\neg)$ when determining the <P,U>-Justified Updates.

Example 10 *Consider program P stating that someone is a pacifist and that a pacifist is a reasonable person. Later on, an update U states that it is not clear whether we're at war or at peace, and that a state of war will make that person no longer a pacifist:*

$$\begin{array}{llcll} P: & pacifist \leftarrow & \quad & U: & \neg pacifist \leftarrow war \\ & reasonable \leftarrow pacifist & & & peace \leftarrow not\ war \\ & & & & war \leftarrow not\ peace \end{array}$$

Intuitively, when performing the update of P by U, we should obtain two models, namely

$$\begin{array}{l} M_1 = \{pacifist, reasonable, peace\} \\ M_2 = \{war\} \end{array}$$

Let's check whether they are <P,U>-justified updates. M_1 is $M_{\neg 1}$ restricted to the language of P:

$$M_{\neg 1} = \quad \{pacifist, reasonable, peace\}$$

Since

$$Rejected(M_{\neg 1}) = \{\}$$

$$P^* = P + U - \{\}$$

$M_{\neg 1}$ is an answer-set of P^*, and so M_1 is a <P,U>-justified update.

M_2 is $M_{\neg 2}$ restricted to the language of P:

$$M_{\neg 2} = \quad \{war, \neg pacifist\}$$

Since

$$Rejected(M_{\neg 2}) = \{pacifist \leftarrow\}$$

$$P^* = P + U - \{pacifist \leftarrow\}$$

$M_{\neg 2}$ is an answer-set of P^* and so M_2 is a <P,U>-justified update.

Let's check if the model

$$M_X = \{reasonable, war\}$$

is a <P,U>-justified update. Intuitively it should not be one because the truth value of reasonable *should be determined by the evaluation of the rule of P,* reasonable←pacifist, *on the strength of the truth of* pacifist *in the updated model, and therefore should be false. Note, however, that this model would be a justified update of the only stable model of P, determined according to interpretation updating.*

Once again M_X is $M_{\neg X}$ restricted to the language of P:

$$M_{\neg X} = \quad \{reasonable, war, \neg pacifist\}$$

Since

$$Rejected(M_{\neg X}) = \{pacifist \leftarrow\}$$

$$P^* = P + U - \{pacifist \leftarrow\}$$

As expected, $M_{\neg X}$ is not an answer-set of P^*, and therefore M_X is not a <P,U>-justified update.

Next we present a program transformation that produces an updated program from an initial program and an update program. The answer-sets of the updated program so obtained will be exactly the <P,U>-justified models, according to Theorem 14 below. The updated program can thus be used to compute them.

Definition 11 (Update transformation of a normal program) *Consider an update program U over a language $L_\neg$. For any normal logic program P over L, its updated program P_U with respect to U is obtained via the operations:*

- *All rules of U and P belong to P_U subject to the changes:*
  - *in the head of every rule of P_U originated in U replace literal X by a new literal X^U;*
  - *in the head of every rule of P_U originated in P replace atom A by a new atom A';*
- *Include in P_U, for every atom A of P or U, the defining rules:*

$$A \leftarrow A', not\ \neg A^U \qquad A \leftarrow A^U \qquad \neg A \leftarrow \neg A^U \ \diamondsuit$$

The above definition assumes that in the language L there are no symbols of the form L' and L^U.This transformation is reminiscent of the one presented in [AP97], where the goal was to update a set of models encoded by a logic program. In [AP97], literals figuring in the head of a rule of U (but it could be for any literal) originate replacement of the corresponding atom in both the head and body of the rules of the initial program, whereas in the above transformation this replacement occurs only in the head (for all rules). This has the effect of exerting inertia on the rules instead of on the model literals because the original rules will be evaluated in the light of the updated model. The defining rules establish that, after the update, a literal is either implied by inertia or forced by an update rule. Note that only update rules are allowed to inhibit the inertia rule, in contrast to the usual inertia rules for model updates. In model updates there are no rule bodies in the coding of the initial interpretation as fact rules, so the conclusion of these rules cannot change, in contradistinction to the case of program updates. Hence the new inertia rule, which applies equally well to model updating (cf. justification in Theorem 17) and so is more general. Their intuitive reading is: A can be true either by inertia or due to the update program.

Example 12 *Consider the normal logic program P with single stable model M:*

$$\begin{array}{rl} P: & a \leftarrow not\ b \\ & d \leftarrow e \\ & e \leftarrow \end{array}$$

$$M = \{a, d, e\}$$

now consider the update program U:

$$\begin{array}{rl} U: & c \leftarrow not\ a \\ & b \leftarrow \\ & \neg e \leftarrow a \end{array}$$

And the updated program P_U is (where the rules for A stand for all their ground instances):

$$\begin{array}{lll} c^U \leftarrow not\ a & a^{'} \leftarrow not\ b & A \leftarrow A', not\ \neg A^U \\ b^U \leftarrow & d^{'} \leftarrow e & A \leftarrow A^U \\ \neg e^U \leftarrow a & e' \leftarrow & \neg A \leftarrow \neg A^U \end{array}$$

whose only answer-set (modulo A' and A^U atoms) is:

$$M_U = \{b, c, d, e\}$$

This corresponds to the intended result: the insertion of **b** *renders* **a** *no longer supported and thus false; since* **a** *is false,* **c** *becomes true due to the first rule of the update program; the last rule of U is ineffective since* **a** *is false;* **e** *is still supported and not updated, so it remains true by inertia; finally* **d** *remains true because still supported by* **e**.

If we consider this same example but performing the updating on a model basis instead, we would get as the only U-justified update of M: $M' = \{a, b, d\}$. The difference, for example in what **a** is concerned, is that in M' **a** is true by inertia because it is true in M and there are no rules for **a** in U. According to our definition, since there aren't any rules (with a true body) in U for **a**, the rule in P for **a** is still valid by inertia and re-evaluated in the final interpretation, where since **b** is true **a** is false.

Example 13 *Consider the P and U of example 10. The updated program P_U of P by U is:*

$$\begin{array}{ll} pacifist' \leftarrow & \neg pacifist^U \leftarrow war \\ reasonable' \leftarrow pacifist & peace^U \leftarrow not\ war \\ A \leftarrow A', not\ \neg A^U & war^U \leftarrow not\ peace \\ A \leftarrow A^U & \neg A \leftarrow \neg A^U \end{array}$$

whose answer-sets (modulo A', A^U and explicitly negated atoms) are:

$$\begin{array}{l} M_1 = \{pacifist, reasonable, peace\} \\ M_2 = \{war\} \end{array}$$

coinciding with the two <P,U>-justified updates determined in example 10.

The following theorem establishes the relationship between the models of the update transformation of a program and its <P,U>-justified updates.

Theorem 14 (Correctness of the update transformation) *Let P be a normal logic program over L and U a coherent update program over $L\neg$. Modulo any primed and X^U literals, the answer-sets of the updated program P_U are exactly the <P,U>-Justified Updates of P updated by U.* $\diamondsuit$

Proof. Let P be a normal logic program consisting of rules of the form:

$$A \leftarrow B_i, not\ C_i$$

and U an update program consisting of rules of the form:

$$\begin{array}{l} A \leftarrow B_j, not\ C_j \\ \neg A \leftarrow B_k, not\ C_k \end{array}$$

where A is an atom and each B and C is some finite set of atoms .

Let P_U^* be the program obtained according to Def. 9:

$$P_U^* = U + P_{inertial}(M_\neg)$$

and note that $P_{inertial}(M_\neg) \subseteq P$.

Let P_U be the program obtained according to Def. 11:

$$\begin{array}{lll} P_U: & A' \leftarrow B_i, not\ C_i & \text{for all rules from } P \\ & \left.\begin{array}{l} A \leftarrow A', not\ \neg A^U \\ A \leftarrow A^U \\ \neg A \leftarrow \neg A^U \end{array}\right\} & \text{for all } A \\ & \left.\begin{array}{l} A^U \leftarrow B_j, not\ C_j \\ \neg A^U \leftarrow B_k, not\ C_k \end{array}\right\} & \text{for all rules from } U \end{array}$$

We will show that P_U is equivalent to P_U^* for our purposes. Performing on P_U a partial evaluation of A^U and $\neg A^U$ on the rules $A \leftarrow A^U$ and $\neg A \leftarrow \neg A^U$ we obtain:

$$\begin{array}{lll} P_U^{'}: & A' \leftarrow B_i, not\ C_i & (1) \\ & A \leftarrow A', not\ \neg A^U & (2) \\ & A \leftarrow B_j, not\ C_j & (3) \\ & \neg A \leftarrow B_k, not\ C_k & (4) \\ & A^U \leftarrow B_j, not\ C_j & (5) \\ & \neg A^U \leftarrow B_k, not\ C_k & (6) \end{array}$$

Note that rules (3) and (4) are exactly the update program.

These rules can be simplified. In particular we don't need the rules for A^U and $\neg A^U$. For some arbitrary A, consider first the case where $\neg A^U$ is false. We can then perform the following simplifications on $P_U^{'}$: replace in (2) A' by the body of (1) and remove *not* $\neg A^U$ to obtain (2*): $A \leftarrow B_i, not\ C_i$; now we no longer need rule (6). Since we don't care about primed nor A^U literals in the updated models we can now remove rule (1), as well as rules (5) and (6)). The so mutilated $P_U^{'}$ preserves the semantics of $P_U^{'}$ when $\neg A^U$ is false, apart primed and U literals, and looks like this:

$$\begin{array}{ll} A \leftarrow B_i, not\ C_i & (2*) \\ A \leftarrow B_j, not\ C_j & (3) \\ \neg A \leftarrow B_k, not\ C_k & (4) \end{array}$$

which corresponds exactly to P_U^* when $P_{inertial}(M_\neg) = P$ when $\neg A^U$ is false, and hence their answer-sets are the same in that case.

For the case where $\neg A^U$ is true, we can delete rule (2); rule (6) is also not needed for we don't care about $\neg A^U$ literals in the updated models. Since we don't care about primed nor A^U literals in the updated models, and A' and A^U don't appear in the body of remaining rules, we can delete rules (1) and (5). The simplified $P_U^{'}$ preserves the semantics of $P_U^{'}$ when $\neg A^U$ is true, apart primed and U literals, and looks like this:

$$\begin{array}{ll} A \leftarrow B_j, not\ C_j & (4) \\ \neg A \leftarrow B_k, not\ C_k & (5) \end{array}$$

which is semantically equal to P_U^*. Indeed, note that when $\neg A^U$ is true, the rules of P for A are rejected if $M_\neg \models B_i, not\ C_i$ and don't belong to P_U^*. So the only possible difference between the simplified $P_U^{'}$ and P_U^* would be the existence of some extra rules in P_U^* such that for any answer-set $M_\neg$ we would have $M_\neg \not\models B_i, not\ C_i$, which does not affect the semantics ■.

The next theorem establishes the relationship between program update and interpretation update. For this we begin by defining a transformation from an interpretation into the arguably simplest normal logic program that encodes it.

Definition 15 (Factual LP) *Let I be an interpretation over a language L. We define the normal logic program associated with I, $P^*(I)$, as:*

$$P^*(I) = \{L \leftarrow : L \in I\} \ \diamondsuit$$

We also need the following closeness relationship:

Definition 16 (Closeness relationship) *Given three total interpretations I, I_u and $I_u^{'}$, we say that $I_u^{'}$ is closer to I than I_u if*

$$(I_u^{'} \setminus I \cup I \setminus I_u^{'}) \subset (I_u \setminus I \cup I \setminus I_u) \ \diamondsuit$$

Theorem 17 (Generalization of Updates) *Let U be an update program and I an interpretation. Then:*

1. *Every U-justified update of I is a $<P^*(I), U>$-justified update.*
2. *A $<P^*(I), U>$-justified update I_u is a U-justified update of I iff*

$$\nexists I_u^{'} \textit{ closer to } I \textit{ than } I_u$$

where $I_u^{'}$ is a $<P^*(I), U>$-justified update. $\Diamond$

Proof.

1. Let U be an update program consisting of rules of the form:

$$\begin{array}{l} A \leftarrow B_j, not\ C_j \\ \neg A \leftarrow B_k, not\ C_k \end{array}$$

where A is an atom and each B and C is some finite set of atoms.

According to [AP97], an interpretation I_u is a U-justified update of I iff it is a total (or two-valued) WFSX model (modulo primed and explicitly negated elements) of the corresponding program P_U:

$$\begin{array}{lll} P_U: & A' \leftarrow & \text{for all } A \in I \\ & A \leftarrow A', not\ \neg A & \multirow{2}{*}{\text{for all } A} \\ & \neg A \leftarrow not\ A', not\ A & \\ & A \leftarrow B_j, \neg C_j & \multirow{2}{*}{\text{for all rules from } U} \\ & \neg A \leftarrow B_k, \neg C_k & \end{array}$$

according to Def. 11, an interpretation $I_u^{'}$ is a $<P^*(I), U>$-justified update iff it is the restriction to the language of I of an answer-set of the program $P_U^{'}$:

$$\begin{array}{lll} P_U^{'}: & A' \leftarrow & \text{for all } A \in I \\ & A \leftarrow A', not\ \neg A^U & \\ & A \leftarrow A^U & \text{for all } A \\ & \neg A \leftarrow \neg A^U & \\ & A^U \leftarrow B_j, not\ C_j & \text{for all rules from } U \\ & \neg A^U \leftarrow B_k, not\ C_k & \end{array}$$

Notice the difference in the translation of update rules in what the kind of negation used in their bodies is concerned. We will show that for every total (or two-valued) WFSX model I_u of the program P_U, there is an answer-set $I_u^{'}$ of $P_U^{'}$ such that $I_u = I_u^{'}$ restricted to the language of I.

Performing a partial evaluation of A^U and $\neg A^U$ on the rules $A \leftarrow A^U$ and $\neg A \leftarrow \neg A^U$ we obtain:

$$\begin{array}{lll} P_U^{'}: & A' \leftarrow & (1) \\ & A \leftarrow A', not\ \neg A^U & (2) \\ & A \leftarrow B_j, not\ C_j & (3) \\ & \neg A \leftarrow B_k, not\ C_k & (4) \\ & A^U \leftarrow B_j, not\ C_j & (5) \\ & \neg A^U \leftarrow B_k, not\ C_k & (6) \end{array}$$

We can safely replace *not* $\neg A^U$ by *not* $\neg A$ in rule (2), for the only rules for $\neg A$ and $\neg A^U$ have the same body. Now, and since we don't care about A^U and $\neg A^U$ in the updated models, we can remove rules (5) and (6) and obtain the following program $P_U^{''}$:

$$\begin{array}{lll|lll}
P_U^{''}: & A' \leftarrow & (1) & P_U: & A' \leftarrow \\
 & A \leftarrow A', not\ \neg A & (2) & & A \leftarrow A', not\ \neg A \\
 & & (3) & & \neg A \leftarrow not\ A', not\ A \\
 & A \leftarrow B_j, not\ C_j & (4) & & A \leftarrow B_j, \neg C_j \\
 & \neg A \leftarrow B_k, not\ C_k & (5) & & \neg A \leftarrow B_k, \neg C_k
\end{array}$$

It is easy to see that the only differences between $P_U^{''}$ and P_U are the kind of negation used in the body of the rules from the update program, and the extra rule (3) in P_U. Suppose that we add rule (3) to $P_U^{''}$: if rule (3) has a true body, rule (2) must have a false body; since we are not concerned about $\neg A$ in the final models, and $\neg A$ doesn't appear in the body of any other rules, adding rule (3) to $P_U^{''}$ wouldn't change the restricted models. Now, the only difference is the kind of negation used, but since in answer-sets we have that if $\neg C$ is true then *not* C is also true, we have that all total WFSX models of P_U are also answer-sets of $P_U^{''}$.

2. There now remains to be proved the closeness part of the theorem, i.e. that the set of interpretations $S = Q - R$, where

$$\begin{array}{l}
Q = \{I_u : I_u \text{ is a } < P^*(I), U > \text{-justified update}\} \\
R = \{I_u : I_u \text{ is a } U\text{-justified update of } I\}
\end{array}$$

is such that for every $I_u^{'}$ in S, there is an I_u in R such that I_u is closer to I than $I_u^{'}$, and thus eliminated by the closeness condition. According to [MT94], I_u is a U-justified update of I iff it satisfies the rules of U (as per Def.3 and where I satisfies $in(a)$ (resp. $out(a)$) if $a \in I$ (resp. $a \notin I$)), and is closest to I among such interpretations. From definition 9, every $<P, U>$-justified update must satisfy the rules of U, of the form:

$$\begin{array}{l}
A \leftarrow B_j, not\ C_j \\
\neg A \leftarrow B_k, not\ C_k
\end{array} \tag{3}$$

Since for any answer-set if $\neg a \in I$ then $a \notin I$, we have that any $<P, U>$-justified update, because it satisfies the rules of (3), must also satisfy the update rules with in's and out's of the form (4)

$$\begin{array}{l}
in(A) \leftarrow in(B_j), out(C_j) \\
out(A) \leftarrow in(B_k), out(C_k)
\end{array} \tag{4}$$

Let X be the set of all interpretations that satisfy the rules of (4). Then the interpretations in $X - R$ are the ones eliminated by the closeness condition, to obtain the U-justified updates, according to [MT94]. Since

$R \subseteq Q$ (first part of the theorem), and every interpretation of Q satisfies the rules of (4), we have that $S \subseteq X$ and thus any interpretation in S is eliminated by the closeness condition of this theorem. ■

Therefore the notion of program update presented here is a generalization of the updates carried out on a model basis. Consequently, the program transformation above is a generalization of the program transformation in [AP97], regarding its 2-valued specialization. Elsewhere [Lei97] the 3-valued case is generalised as well.

Remark 18 (Extending the language of initial programs) *We could allow for initial programs to be of the same form as update programs, i.e. with explicit negated literals in their heads only, as per Def.7. For this, we would have to change Definitions 8 and 11 by replacing atom A there with objective literal L*[1] *(see [Lei97]). However, note that, although both programs have explicit negation in their heads, its use is limited, as explicit negation does not appear in rule bodies. Indeed, all its occurrences can be replaced by allowing* **not** *in heads instead, and then employing a semantics for such generalized programs such as [LW92],[DP96].*

4 Extended Logic Program Updating

When we update a normal logic program the result is an extended logic program. In order to update these in turn we need to extend the results of the previous section to cater for explicit negation in programs. Besides this obvious motivation, there is much work done on representing knowledge using extended logic programs, and we want to be able to update them. We begin by extending the definitions of the previous section to allow for the inclusion of explicit negation anywhere in a normal program.

Definition 19 (Update Rules for Objective Literals) *[AP97]Let K be a countable set of objective literals. Update in-rules or, simply in-rules, and update out-rules or, simply, out-rules, are as (1) and as (2), but with respect to this new set K.* ◊

Also, for extended update programs their transformation into an extended logic programs is now:

[1]An updated program can in turn be updated, once the inertia rule is generalized for objective literals: $L \leftarrow L', not \neg L$. Because the inertia rule contains explicitely negated literals in its body, the language of programs has to be extended, as per the next section. However, the inertia rule itself does not need to be updated, only the program and update rules. These will accumulate dashes in their heads as they are updated. For the inertia rule to recurrently strip away successive dashes one needs to introduce the equivalence $(\neg A)' = \neg(A)'$, and define $\neg$ and $'$ as operators to allow unification to do its work. For the details of such a generalization the reader is referred to [Lei97].

Definition 20 (Translation of extended UPs into ELPs) *[AP97]Given an update program with explicit negation* UP, *its* translation into the extended logic program U *is defined as follows*[2]*:*

1. *Each in-rule*

$$in(L_0) \leftarrow in(L_1), ..., in(L_m), out(L_{m+1}), ..., out(L_n)$$

where $m,\ n \geq 0$, *and* L_i *are objective literals, translates into:*

$$L_0^* \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n$$

where $L_0^* = A^p$ *if* $L_0 = A$, *or* $L_0^* = A^n$ *if* $L_0 = \neg A$*;*

2. *Each out-rule*

$$out(L_0) \leftarrow in(L_1), ..., in(L_m), out(L_{m+1}), ..., out(L_n)$$

where $m,\ n \geq 0$, *and* L_i *are objective literals, translates into:*

$$\neg L_0^* \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n$$

where $L_0^* = A^p$ *if* $L_0 = A$, *or* $L_0^* = A^n$ *if* $L_0 = \neg A$*;*

3. *For every objective literal* L *such that* $in(L)$ *belongs to the head of some in-rule of* UP, U *contains* $\neg L^* \leftarrow L$ *where* $L^* = A^n$ *if* $L = A$, *or* $L^* = A^p$ *if* $L = \neg A$*;*

4. *For every atom* A, U *contains the rules* $A \leftarrow A^p$ *and* $\neg A \leftarrow A^n$. $\diamondsuit$

Intuitively, this transformation converts an atom A into a new atom A^p and an explicitly negated atom $\neg A$ into a new atom A^n and ensures coherence. This way, we no longer have explicitly negated atoms in the heads of the rules of update programs and so we can use explicit negation $\neg L$ to code the $out(L)$ in the heads of rules, as for update programs without explicit negation. Operation 4 maps the A^n and A^p back to their original atoms.

Conversely, any extended logic program (ELP) can be seen as an update program, possibly applied to an empty program. Indeed, translate each ELP rule of the form

$$L_0 \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n$$

where L_i are objective literals, to

$$in(L_0) \leftarrow in(L_1), ..., in(L_m), out(L_{m+1}), ..., out(L_n)$$

[2]This translation employs the results in [DP96], namely the expressive power of WFSX to capture the semantics of extended logic programs with default literals in the heads of rules, via the program transformation P^{not}.

It is easy to see that applying the above translation (Def.20) of such an update program back into an ELP preserves the semantics of the original program because of the read-out rules, $A \leftarrow A^p$ and $\neg A \leftarrow A^n$.

The language of update programs is more expressive than that of ELPs because one may additionally have $out(A_0)$ and $out(\neg A_0)$. The semantics of such ELP$_{out}$ programs can be defined simply by the ELP semantics of the translation into an ELP of their corresponding update programs.

Then we can envisage any ELP (or ELP$_{out}$) program as an update specification for another ELP (or ELP$_{out}$) program, albeit the empty one. Programs can update one another, in succession.

Definition 21 (Extended Interpretation Restriction) *Given a language K with explicit negation. Let M_{np} be the an interpretation over the language K_{np}, obtained by augmenting K with the set $E = \{L^n, L^p : L \in K\}$ (L^n, L^p and L are objective literals).*

We define the corresponding restricted interpretation *M, over K, as:*

$$M = M_{np} \textit{ restricted to } K \ \diamondsuit$$

Definition 22 (Inertial Sub-Program) *Let P be an extended logic program over K, U an update program over K_{np} and M_{np} a model of U. Let:*

$$\begin{array}{rl} Rejected(M_{np}) = & \{A \leftarrow body \in P : M_{np} \models body \\ & and\ \exists \neg A^p \leftarrow body' \in U : M_{np} \models body'\ \} \cup \\ & \cup \{\neg A \leftarrow body \in P : M_{np} \models body \\ & and\ \exists \neg A^n \leftarrow body' \in U : M_{np} \models body'\ \} \end{array}$$

where A is an atom. We define Inertial Sub-Program *$P_{inertial}(M_{np})$ as:*

$$P_{inertial}(M_{np}) = P - Rejected(M_{np}) \ \diamondsuit$$

Again, the rules for some objective literal L that belong to $Rejected(M_{np})$ are those that belong to the initial program but, although their body is still verified by the model, there is an update rule that overrides them, by contravening their conclusion. Note that a rule of P for atom A, with true body, is also countervened by a rule of U with true body for A^n (i.e. one translated from $in(\neg A)$). Since every U also contains the rules $\neg A^p \leftarrow \neg A$ and $\neg A \leftarrow A^n$, then $\neg A$ in $\neg A^p \leftarrow \neg A$ is also true, and so that rule of P is rejected in this case too. Similarly for a rule of P with head $\neg A$, but now with respect to A^p.

Definition 23 (<P,U>-Justified Updates) *Let P be an extended logic program, U an update program, and M an interpretation over the language of P. M is a* <P,U>-Justified Update *of P updated by U iff M_{np} is an answer-set of P^*, where*

$$P^* = P_{inertial}(M_{np}) + U \ \diamondsuit$$

Once again we should point out that the extended <P,U>-Justified Update doesn't depend on any initial interpretation. As for the case of normal logic programs, it is the rules that suffer the effects of inertia and not model literals per se.

Example 24 *Consider a recoding of the alarm example using explicit negation, where P and UP are:*

$$\begin{array}{llll} P: & sleep \leftarrow \neg alarm & UP: & in(\neg alarm) \leftarrow \\ & panic \leftarrow alarm & & \\ & alarm \leftarrow & & \end{array}$$

the update program U obtained from UP is:

$$\begin{array}{l} alarm^n \leftarrow \\ \neg alarm^p \leftarrow \neg alarm \\ alarm \leftarrow alarm^p \\ \neg alarm \leftarrow alarm^n \end{array}$$

Intuitively, when performing the update of P by U, we should obtain a single model, namely

$$M = \{\neg alarm, sleep\}$$

Let's check whether M is an extended <P,U>-justified update. M is M_{np} restricted to the language of P:

$$M_{np} = \quad \{\neg alarm, sleep, alarm^n, \neg alarm^p\}$$

Since

$$Rejected(M_{np}) = \{alarm \leftarrow\}$$

$$P^* = P + U - \{alarm \leftarrow\}$$

M_{np} is an answer-set of P^*, and so M is an extended <P,U>-justified update.

Definition 25 (Update transformation of an extended LP) *Given an update program UP, consider its corresponding extended logic program U. For any extended logic program P, its updated program P_U with respect to U is obtained through the operations:*

- *All rules of U and P belong to P_U subject to the changes, where X is a literal:*
  - *in the head of every rule of P_U originated in U, replace X^p (resp. X^n) by a new literal X^{pU} (resp. X^{nU});*
  - *in the head of every rule of P_U originated in P, replace X by a new literal $X^{'}$;*

- *Include in P_U, for every atom A of P or U, the defining rules:*

$$\begin{array}{ll} A^n \leftarrow \neg A', not\ \neg A^{nU} & A^p \leftarrow A', not\ \neg A^{pU} \\ A^n \leftarrow A^{nU} & A^p \leftarrow A^{pU} \\ \neg A^n \leftarrow \neg A^{nU} & \neg A^p \leftarrow \neg A^{pU} \quad \diamondsuit \end{array}$$

As before, the transformation reflects that we want to preserve, by inertia, the rules for those literals in P not affected by the update program. This is accomplished via the renaming of the literals in the head of rules only, whilst preserving the body, plus the inertia rules.

Theorem 26 (Correctness of the update transformation) *Let P be an extended logic program and U a coherent update program. Modulo any primed, A^U, A^p and A^n elements and their defaults, the answer-sets of the updated program P_U of P with respect to U are exactly the <P,U>-Justified Updates of P updated by U.* $\diamondsuit$

Proof. (sketch): Let P be an extended logic program consisting of rules of the form:

$$\begin{array}{l} A \leftarrow B_i, not\ C_i \\ \neg A \leftarrow B_j, not\ C_j \end{array}$$

and U an update program consisting of rules of the form:

$$\begin{array}{ll} A^p \leftarrow B_k, not\ C_k & A \leftarrow A^p \\ \neg A^p \leftarrow B_l, not\ C_l & \neg A \leftarrow A^n \\ A^n \leftarrow B_m, not\ C_m & \neg A^n \leftarrow A \\ \neg A^n \leftarrow B_n, not\ C_n & \neg A^p \leftarrow \neg A \end{array}$$

where A is an atom and each B and C is some finite set of objective literals.

Let P_U^* be the program obtained according to Def. 9:

$$P_U^* = U + P_{inertial}(M_{np})$$

and note that $P_{inertial}(M_{np}) \subseteq P$.

Let P_U be the program obtained according to Def. 11:

$$P_U: \begin{array}{l} A' \leftarrow B_i, not\ C_i \\ \neg A' \leftarrow B_j, not\ C_j \end{array} \left.\vphantom{\begin{array}{l}a\\b\end{array}}\right\} \text{for all rules from } P$$

$$\begin{array}{l} A^p \leftarrow A', not\ \neg A^{pU} \\ A^n \leftarrow \neg A', not\ \neg A^{nU} \\ A^p \leftarrow A^{pU} \\ \neg A^p \leftarrow \neg A^{pU} \\ A^n \leftarrow A^{nU} \\ \neg A^n \leftarrow \neg A^{nU} \\ A \leftarrow A^p \\ \neg A \leftarrow A^n \end{array} \left.\vphantom{\begin{array}{l}a\\b\\c\\d\\e\\f\\g\\h\end{array}}\right\} \text{for all } A$$

$$\begin{array}{l} A^{pU} \leftarrow B_k, not\ C_k \\ \neg A^{pU} \leftarrow B_l, not\ C_l \\ A^{nU} \leftarrow B_m, not\ C_m \\ \neg A^{nU} \leftarrow B_n, not\ C_n \\ \neg A^{nU} \leftarrow A \\ \neg A^{pU} \leftarrow \neg A \end{array} \left.\vphantom{\begin{array}{l}a\\b\\c\\d\\e\\f\end{array}}\right\} \text{rules from } U$$

We will show that P_U is equivalent to P_U^* for our purposes. Performing on P_U a partial evaluation of A^{pU}, $\neg A^{pU}$, A^{nU} and $\neg A^{nU}$ on the rules $A^p \leftarrow A^{pU}$, $\neg A^p \leftarrow \neg A^{pU}$, $A^n \leftarrow A^{nU}$ and $\neg A^n \leftarrow \neg A^{nU}$ we obtain:

$$P_U^{'}: \begin{array}{lrlr} A' \leftarrow B_i, not\ C_i & (1) & \neg A^p \leftarrow \neg A & (10) \\ \neg A' \leftarrow B_j, not\ C_j & (2) & A \leftarrow A^p & (11) \\ A^p \leftarrow A', not\ \neg A^{pU} & (3) & \neg A \leftarrow A^n & (12) \\ A^n \leftarrow \neg A', not\ \neg A^{nU} & (4) & A^{pU} \leftarrow B_k, not\ C_k & (13) \\ A^p \leftarrow B_k, not\ C_k & (5) & \neg A^{pU} \leftarrow B_l, not\ C_l & (14) \\ \neg A^p \leftarrow B_l, not\ C_l & (6) & A^{nU} \leftarrow B_m, not\ C_m & (15) \\ A^n \leftarrow B_m, not\ C_m & (7) & \neg A^{nU} \leftarrow B_n, not\ C_n & (16) \\ \neg A^n \leftarrow B_n, not\ C_n & (8) & \neg A^{nU} \leftarrow A & (17) \\ \neg A^n \leftarrow A & (9) & \neg A^{pU} \leftarrow \neg A & (18) \end{array}$$

Note that rules (5)-(12) are exactly equal to the rules of the update program.

The structure of the remaining part of the proof is quite similar to the one set forth in theorem 14. Its details are slightly more extensive for we now have to simplify $P_U^{'}$ eliminating A^{pU}, $\neg A^{pU}$, A^{nU} and $\neg A^{nU}$ whilst in theorem 14 we only had to consider A^U and $\neg A^U$. ■

Example 27 *Applying this transformation to the alarm example (Ex. 24)*

$$P: \begin{array}{l} sleep \leftarrow \neg alarm \\ panic \leftarrow alarm \\ alarm \leftarrow \end{array} \qquad U: \ in(\neg alarm) \leftarrow$$

we obtain (where the rules for A and $\neg A$ stand for their ground instances):

$$\begin{array}{lll}
P_U: & sleep' \leftarrow \neg alarm & A^p \leftarrow A', not\ \neg A^{pU} \\
& panic' \leftarrow alarm & A^n \leftarrow \neg A', not\ \neg A^{nU} \\
& alarm' \leftarrow & A^p \leftarrow A^{pU} \\
& alarm^{nU} \leftarrow & \neg A^p \leftarrow \neg A^{pU} \\
& \neg alarm^{pU} \leftarrow \neg alarm & A^n \leftarrow A^{nU} \\
& A \leftarrow A^p & \neg A^n \leftarrow \neg A^{nU} \\
& \neg A \leftarrow A^n &
\end{array}$$

with model (modulo L', L^n, L^p, L^U):

$$M_U = \{sleep, \neg alarm\}$$

Definition 15 and Theorem 17 both now carry over to a language K with explicit negation.

Definition 28 (Extended factual LP) *Let I be an interpretation over a language K with explicit negation. We define the extended logic program associated with I, $P^*(I)$, as:*

$$P^*(I) = \{L \leftarrow : L \in I\}$$

where the Ls are objective literals. $\diamondsuit$

It is worth pointing out that the translation of update programs into extended logic programs, making use of explicit negation $\neg$ to code the out's in the heads of update rules and default negation *not* to code the out's in the bodies of the same rules, allows for some pairs of answer-sets, one of which will always be closer than the other to the initial interpretation. This is best illustrated by the following example:

Example 29 *Let $I = \{a\}$ and $U = \{\neg a \leftarrow not\ a\}$ where U is the translation of $U' = \{out(a) \leftarrow out(a)\}$ according to Def.7. The updated program is:*

$$\begin{array}{ll}
P_U: & a' \leftarrow \\
& a \leftarrow a', not\ \neg a^U \\
& \neg a^U \leftarrow not\ a
\end{array}$$

with two answer-sets whose restrictions are $M_1 = \{a\}$ and $M_2 = \{\}$. Note that M_1 is closer to I than M_2.

The closeness condition in theorems 17 and 30 exists to eliminate such farther models in order to obtain the U-justified updates only. As mentioned, this phenomena is due to the translation of the update programs. This is also shared by [AP97] for the case of updates extended with explicit negation, and so their soundness and completeness theorem should also make use of the closeness relationship.

This translation has the virtue of not excluding such models, just in case they are seen as desired. Another approach exists, mentioned in the conclusions, that avoids the need for the closeness relation by excluding the non-closest updates by construction.

Theorem 30 (Generalization of Updates) *Let U be an update program with explicit negation and I an interpretation. Then:*

1. *Every U-justified update of I is a $<P^*(I), U>$-justified update.*
2. *A $<P^*(I), U>$-justified update I_u is a U-justified update of I iff*

$$\nexists I_u^{'} \textit{ closer to } I \textit{ than } I_u$$

where $I_u^{'}$ is a $<P^*(I), U>$-justified update. $\diamondsuit$

Proof. (sketch): Let U be an update program consisting of rules of the form:

$$\begin{array}{ll}
A^p \leftarrow B_k, not\ C_k & A \leftarrow A^p \\
\neg A^p \leftarrow B_l, not\ C_l & \neg A \leftarrow A^n \\
A^n \leftarrow B_m, not\ C_m & \neg A^n \leftarrow A \\
\neg A^n \leftarrow B_n, not\ C_n & \neg A^p \leftarrow \neg A
\end{array}$$

where A is an atom and each B and C is some finite set of objective literals.

According to [AP97], a total (or two-valued) WFSX model (modulo primed and explicitly negated elements) of the program P_U is a U-justified update iff it is closest to I, among all such models, where P_U is:

$$\begin{array}{lll}
P_U: & A' \leftarrow & \text{for all } A \in I \\
 & \neg A' \leftarrow & \text{for all } \neg A \in I \\
 & \left.\begin{array}{l}
A^p \leftarrow A', not\ \neg A^p \\
\neg A^p \leftarrow not\ A', not\ A^p \\
A^n \leftarrow \neg A', not\ \neg A^n \\
\neg A^n \leftarrow not\ \neg A', not\ A^n \\
A \leftarrow A^p \\
\neg A \leftarrow A^n \\
\neg A^n \leftarrow A \\
\neg A^p \leftarrow \neg A
\end{array}\right\} & \text{for all } A \\
 & \left.\begin{array}{l}
A^p \leftarrow B_k, not\ C_k \\
\neg A^p \leftarrow B_l, not\ C_l \\
A^n \leftarrow B_m, not\ C_m \\
\neg A^n \leftarrow B_n, not\ C_n
\end{array}\right\} & \text{rules from } U
\end{array}$$

according to Def. 11, an interpretation $I_u^{'}$ is a $<P^*(I), U>$-justified update iff it is the restriction of an answer-set of the program $P_U^{'}$ (after the same partial evaluation as done in the proof of theorem 26):

$$\begin{array}{llrlr}
P'_U: & A' \leftarrow B_i, not\ C_i & (1) & \neg A^p \leftarrow \neg A & (10) \\
& \neg A' \leftarrow B_j, not\ C_j & (2) & A \leftarrow A^p & (11) \\
& A^p \leftarrow A', not\ \neg A^{pU} & (3) & \neg A \leftarrow A^n & (12) \\
& A^n \leftarrow \neg A', not\ \neg A^{nU} & (4) & A^{pU} \leftarrow B_k, not\ C_k & (13) \\
& A^p \leftarrow B_k, not\ C_k & (5) & \neg A^{pU} \leftarrow B_l, not\ C_l & (14) \\
& \neg A^p \leftarrow B_l, not\ C_l & (6) & A^{nU} \leftarrow B_m, not\ C_m & (15) \\
& A^n \leftarrow B_m, not\ C_m & (7) & \neg A^{nU} \leftarrow B_n, not\ C_n & (16) \\
& \neg A^n \leftarrow B_n, not\ C_n & (8) & \neg A^{nU} \leftarrow A & (17) \\
& \neg A^n \leftarrow A & (9) & \neg A^{pU} \leftarrow \neg A & (18)
\end{array}$$

We will have to show that these two transformed programs have the same models, apart from irrelevant elements.

Following similar, though slightly more complex, arguments as in the proof of theorem 17, we can replace A^{pU}, $\neg A^{pU}$, A^{nU} and $\neg A^{nU}$ by A^p, $\neg A^p$, A^n and $\neg A^n$ in rules (3)-(6), and deleting rules (15)-(20). Also rules $\neg A^p \leftarrow not\ A', not\ A^p$ and $\neg A^n \leftarrow not\ \neg A', not\ A^n$ of P_U are irrelevant for the only rules with $\neg A^p$ and $\neg A^n$ in their body also have A' and $\neg A'$ in their body, respectively, which could never be true. Removing those rules from P_U, it would be exactly equal to P'_U, after the simplifications mentioned, thus proving the theorem. ■

5 Conclusions

In this paper we have generalized the notion of updates to the case where we want to update programs instead of just their models. We have shown that since a program encodes more information than a set of models, the law of inertia should be applied to rules instead of to model literals, as had been done so far. We presented a transformation which, given an initial program and an update program, generates the desired updated program. Our results have been further extended to allow for both programs and update programs extended with explicit negation. This is important inasmuch as it permits our updated programs to be updated in turn, and allows us to conceive what it is to successively update one program by another, and so to define the evolution of knowledge bases by means of updates[3].

Future foundational work involves dealing with partial interpretations and

non-coherent update programs and their contradiction removal requirements, among other developments. Indeed, as the world changes, so must logic programs that represent it. Program updating is a crucial notion opening up a

[3]Iterated updates are made easier by a similar approach to that of Footnote 1, where instead the equivalences $(A_n)' = (A')_n$, $(A_p)' = (A')_p$, $(A_n^U)' = (A')_n^U$ and $(A_n^U)' = (A')_n^U$ are introduced. Lack of space prevents us to elaborate further on iterated updates, and garbage collection techniques to do away with rules rendered useless. For the details on these topics the reader is referred to [Lei97].

whole new range of applications, from specification of software updates to temporal databases, from reasoning about actions to active databases, and in general as a means for better representing reasoning, including belief revision.

Acknowledgments We thank José Júlio Alferes, Halina Przymusinska and Teodor Przymusinski for their insightful discussions and suggestions, and the anonymous referees for their comments. A joint paper together with them is well under way, improving on and generalizing the results presented here, as well as exploring some of the application areas mentioned above. (A Prolog implementation of this more general theory is already available.)

References

[AP96] J. J. Alferes, L. M. Pereira. *Reasoning with logic programming*, LNAI 1111, Berlin, Springer-Verlag, 1996.

[AP97] J. J. Alferes, L. M. Pereira. *Update-programs can update programs.* In J. Dix, L. M. Pereira and T. Przymusinski, editors, Selected papers from the ICLP'96 ws NMELP'96, vol. 1216 of LNAI, pages 110-131. Springer-Verlag, 1997.

[APP96] J. J. Alferes, L. M. Pereira and T. Przymusinski. *Strong and Explicit Negation in Nonmonotonic Reasoning and Logic Programming.* In J. J. Alferes, L. M. Pereira and E. Orlowska, editors, JELIA '96, volume 1126 of LNAI, pages 143-163. Springer-Verlag, 1996.

[BD95] S. Brass and J. Dix. *Disjunctive Semantics based upon Partial and Bottom-Up Evaluation.* In Leon Sterling, editor, Procs. of the 12th Int. Conf. on Logic Programming, Tokyo, pag. 85-98, Berlin, June 1995. Springer-Verlag.

[DP96] C. V. Damásio and L. M. Pereira. *Default negated conclusions: why not?* In R. Dyckhoff, H. Herre and P. Schroeder-Heister, editors, Procs. of ELP'96, volume 1050 of LNAI, pages 103-118. Springer-Verlag, 1996.

[GL90] M. Gelfond and V. Lifschitz. *Logic Programs with classical negation.* In Warren and Szeredi, editors, 7th Int. Conf. on LP, pages 579-597. MIT Press, 1990.

[KM91] H. Katsuno and A. Mendelzon. *On the difference between updating a knowledge base and revising it.* In James Allen, Richard Fikes and Erik Sandewall, editors, Principles of Knowledge Representation and Reasoning: Proc. of the Second Int'l Conf. (KR91), pages 230-237, Morgan Kaufmann 1991.

[Lei97] João A. Leite. *Logic Program Updates.* MSc dissertation, Universidade Nova de Lisboa, 1997.

[LW92] V. Lifschitz and T. Woo. *Answer sets in general nonmonotonic reasoning (preliminary report).* In B. Nebel, C. Rich and W. Swartout, editors, Principles of Knowledge Representation and Reasoning, Proc. of the Third Int'l Conf (KR92), pages 603-614. Morgan-Kaufmann, 1992

[MT94] V.Marek and M. Truszczynski. *Revision specifications by means of programs.* In C. MacNish, D. Pearce and L. M. Pereira, editors, JELIA '94, volume 838 of LNAI, pages 122-136. Springer-Verlag, 1994.

[Principia] Isaaco Newtono. *Philosophiæ Naturalis Principia Mathematica.* Editio tertia aucta & emendata. Apud Guil & Joh. Innys, Regiæ Societatis typographos. Londini, MDCCXXVI. Original quotation:*"Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus illud a viribus impressis cogitur statum suum mutare."*.

[PA92] L. M. Pereira and J. J. Alferes. *Well founded semantics for logic programs with explicit negation.* In B. Neumann, editor, European Conf. on AI, pages 102-106. John Wiley & Sons, 1992.

[PT95] T. Przymusinski and H. Turner. *Update by means of inference rules.* In V. Marek, A. Nerode, and M. Truszczynski, editors, LPNMR'95, volume 928 of LNAI, pages 156-174. Springer-Verlag, 1995.