Iterated Logic Program Updates

João Alexandre Leite
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

The field of theory update has seen some improvement, in what regards updating, by allowing desired updates to be specified by update programs. The updating of models is governed by update rules, as well as by inertia applied to those literals not directly affected by the update program. Though this is important, it remains necessary to tackle too the updating of theories specified by programs. In this paper we explore logic program updates, namely how to exert inertia on program rules rather than on the literals of their models. We define program updates and characterize them semantically, both for the 2-valued and the 3-valued cases. We also define iterated logic program updating, a crucial notion inasmuch as it allows us to conceive what it is to successively update one program by another, and so to prescribe the evolution of knowledge bases by means of updates.

Keywords: Nonmonotonic reasoning, Updates

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 both by Keller and Winslett in [8] and by Katsuno and Mendelzon in [7]. One, theory revision, deals with incorporating new knowledge about a static world. The other deals with changing worlds, and is known as theory update. In this work, we are concerned with theory update only, and, in particular, within the framework of logic programming.

A key insight into the issue of updating theories is due to Winslett [14], who showed that one must consider the effect of an update in each of the states of the world that are consistent with our current knowledge of its state. Following this approach, most of the work to date concerns the updating of models on a one by one basis [11][13][4]. The common intuition behind the update of a model has been based on what is referred to as the commonsense law of inertia, i.e. things do not change unless they are expressly made to, in this case that the truth value of each element of the model to be updated should remain the same unless explicitly changed by the update. Suppose, for example, that we have a model in which "sunshine" is true and "rain" is false; if later we receive the information that the sun is no longer shining, we conclude that "sunshine" is false, due to the update, and that "rain" is still false by inertia.

Suppose now that our vision of the world is described instead by a logic program and we want to update it. Is updating each of its models enough? Is all the information borne by a logic program contained within its set of models? The answer to both these questions is negative. A logic program encodes more than the set of its individual models. It encodes the relationships between the elements of a model, which are lost if we envisage updates simply on a model by model basis, as proposed in [7].

To show that a logic program encodes more than its set of models, consider the following situation:

Example 1 *Consider program P stating that someone is a pacifist and that a pacifist is a reasonable person:*

$$P : reasonable \leftarrow pacifist; \quad pacifist \leftarrow .$$

whose only stable model is $M = \{pacifist, reasonable\}$. Later on, an update program (originally introduced in [11] under the name of revision program*) states that we're at war, and that a state of war will make that person no longer a pacifist:*

$$U : out(pacifist) \leftarrow in(war); \quad in(war) \leftarrow .$$

According to [11] and model updating we obtain as the single update of M the model $M_U = \{war, reasonable\}$, to the effect that, although we know that the person is not a pacifist, he/she is still considered a reasonable person. 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 "$reasonable$" was true because "$pacifist$" was true, the removal of "$pacifist$" should make one expect "$reasonable$" to become false. The intended update model of the example presumably is $M_U^{'} = \{war\}$. $\diamondsuit$

Why are we obtaining this somewhat unintuitive result? To answer this question we must first consider the role 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 [12]). 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. Before obtaining the truth value, by inertia, of those elements not directly affected by the update program, one should verify whether the truth of such elements is not indirectly affected by the updating of other elements.

Going back to the example, before stating that "$reasonable$" is true by inertia, since it wasn't directly affected by the update program, we should verify that since the person is no longer a pacifist, there is no way to prove "$reasonable$" and therefore its truth value should no longer be 'true'.

Another way to view program updating, and in particular the role 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 literals, just in case they aren't overruled by the update program. Once again this should be so because the rules encode more information than their models.

This approach was first adopted in [10], under a two-valued semantics, where the authors present a program transformation which, given an initial program and an update program, produces an updated program obeying rule inertia.

In this paper we start by generalizing the transformation and semantical characterization of [10] to the case where the initial program is of the same kind as the update program i.e. capable of expressing negative information in rule conclusion. Then, in section 4, we introduce the notion of iterated update programming: Suppose knowledge is represented by an initial program (possibly empty) and a sequence of update programs. Iterated update programming allows us to represent the evolution of that knowledge, characterizing it after each update. In section 5, we extend logic program updating to the case where a partial stable semantics is used. Finally, in section 6, we conclude and elaborate on future developments.

2 Preliminary Definitions

The following definitions will be needed in the subsequent sections.

Definition 1 (Three-valued interpretation) *A three-valued interpretation M of a language $\mathcal{L}$ is any set*

$$T \cup not\ F$$

where T and F are disjoint subsets of $\mathcal{L}$.

The set T contains all ground atoms which are true in M, the set F contains all ground atoms which are false in M. The truth value of the remaining atoms is unknown (or undefined). Two-valued interpretations are a special case of 3-valued ones, for which $T \cup F = \mathcal{L}$ is further imposed. $\diamondsuit$

Proposition 1 *Any interpretation $M = T \cup not\ F$ can equivalently be viewed as a function $M : \mathcal{L} \rightarrow V$ where $V = \{0, \frac{1}{2}, 1\}$, defined by:*

$$M(A) = \begin{cases} 0 & \textit{if not } A \in M \\ 1 & \textit{if } A \in M \\ \frac{1}{2} & \textit{otherwise} \quad \diamondsuit \end{cases}$$

Based on this function we can define a truth valuation of formulae.

Definition 2 (Truth valuation) *If M is an interpretation, the truth valuation $\hat{M}$ corresponding to M is a function $\hat{M} : C \rightarrow V$ where C is the set of all formulae of the language recursively defined as follows:*

-if A is a ground atom then $\hat{M}(A) = M(A)$.

-if S is a formula then $\hat{M}(not\ S) = 1 - M(S)$.

-if S and V are formulae then:

- $\hat{M}((S, V)) = \min(\hat{M}(S), \hat{M}(V))$.
- $\hat{M}(L \leftarrow S) = 1$ *if* $\hat{M}(S) \leq \hat{M}(L)$, *and* 0 *otherwise.* $\diamondsuit$

As in [10], in order to translate update programs ([11]) written in a language $\mathcal{L}$ into logic programs, we extend $\mathcal{L}$ with explicit negation $\neg$. The next definition relates interpretations of this extended language with interpretations of $\mathcal{L}$.

Definition 3 (Interpretation Restriction) *Given a language $\mathcal{L}$ that does not contain explicit negation $\neg$, let $M_\neg = T_{M_\neg} \cup not\ F_{M_\neg}$*[1] *be an interpretation, of the*

[1] When dealing with total interpretations, M can be more simply represented by just the literals that are true in it.

language $\mathcal{L}_\neg$ *obtained by augmenting* $\mathcal{L}$ *with the set* $E = \{\neg A : A \in \mathcal{L}\}$. *We define the corresponding* restricted interpretation *as* $M = T_M \cup not\ F_M$, *of* $\mathcal{L}$, *where* $T_M = \{A : A \in T_{M_\neg}$ *and* $A \in \mathcal{L}\}$ *and* $F_M = \{A : A \in F_{M_\neg}$ *and* $A \in \mathcal{L}\}$. *Additionally we define an* objective literal *of* $\mathcal{L}_\neg$ *as being an atom* A *or its explicit negation* $\neg A$. $\diamondsuit$

The translation of update programs into extended logic programs follows:

Definition 4 (Translation of UPs into LPs) *Let* UP *be an update program (or revision program as designated in [11]) in the language* $\mathcal{L}$. *Its* translation into an extended logic program U *in the language* $\mathcal{L}_\neg$ *is obtained from* UP *by replacing each update rule (r) by the corresponding rule (r') where:*

- *If r is of the form* $in(p) \leftarrow in(q_1), ..., in(q_m), out(s_1), ..., out(s_n)$
  then r' is of the form $p \leftarrow q_1, ...q_m, not\ s_1, ..., not\ s_n$
- *If r is of the form* $out(p) \leftarrow in(q_1), ..., in(q_m), out(s_1), ..., out(s_n)$
  then r' is of the form $\neg p \leftarrow q_1, ...q_m, not\ s_1, ..., not\ s_n$ $\diamondsuit$

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. In fact, although defined in the language $\mathcal{L}_\neg$, the use of explicit negation in these programs is limited for we will be concerned only with the models of the restricted language $\mathcal{L}$. For this reason, we will dub these programs normal logic programs. Also, and to prepare for further generalization, we allow for the initial program to be of the same form of the translated update program. In [1] we defined iterative updates for programs under stable model semantics and allowing for default negation in heads of rules.

3 Updating Logic Programs under Answer-Set Semantics

To achieve the rule inertia mentioned in the Introduction, we start by defining the subprogram of the initial program containing the rules that persist in the updated program. We use this program together with the update program to characterize the desired models resulting from the update, i.e. the program-justified updates. Afterwards, we present a joint program transformation of the initial and update programs, which includes also inertia rules, to produce an updated program whose models are exactly the required program-justified updates. Answer-sets semantics of extended logic programs [6] will be used to define the models of programs.

3.1 <P,U>-Justified Updates

The <P,U>-justified updates are the set of interpretations that characterize the update of a normal logic program P by an update program U. To obtain them we start by defining a sub-program of the initial program.

Definition 5 (Inertial Sub-Program) *Let* P *be a normal logic program in the language* $\mathcal{L}_\neg$, U *an update program in the language* $\mathcal{L}_\neg$ *and* $M_\neg$ *an interpretation of* $\mathcal{L}_\neg$. *Let:*

$$\begin{aligned} Rejected(M_\neg, P, U) = \{ & L \leftarrow body \in P : \hat{M}_\neg(body) = 1 \\ & and\ \exists \neg L \leftarrow body' \in U : \hat{M}_\neg(body') = 1 \ \} \end{aligned} \quad (1)$$

where L *is an objective literal. The* Inertial Sub-Program $P_{inertial}(M_\neg, P, U)$ *is:*

$$P_{inertial}(M_\neg, P, U) = P - Rejected(M_\neg, P, U) \ \diamondsuit \quad (2)$$

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

Definition 6 (<P,U>-Justified Updates) *Let P be a normal logic program and U an update program, both in the language $\mathcal{L}_\neg$. Let M be an interpretation of the language $\mathcal{L}$. M is a* <P,U>-Justified Update *of P updated by U, iff there is an interpretation $M_\neg$ of $\mathcal{L}_\neg$ such that $M_\neg$ is an answer-set of $\uplus(M_\neg, P, U)$, where*

$$\uplus(M_\neg, P, U) = P_{inertial}(M_\neg, P, U) + U \ \Diamond \tag{3}$$

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.

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

Example 2 *Consider P and U from Example 1. Let's check whether $M'_U = \{war\}$ is a <P,U>-justified update. M'_U is the restriction of $M'_{\neg U} = \{war, \neg pacifist\}$. Since*

$$Rejected(M'_{\neg U}, P, U) = \{pacifist \leftarrow\}\,; \quad \uplus(M'_{\neg U}, P, U) = P + U - \{pacifist \leftarrow\}$$

$M'_{\neg U}$ is an answer-set of $\uplus(M'_{\neg U}, P, U)$ and so M'_U is a <P,U>-justified update.

The model $M_U = \{reasonable, war\}$, being an update of the only answer-set of P, determined according to interpretation updating, is not a <P,U>-justified update. Intuitively, it should not be one because the truth value of "reasonable" should be determined by the evaluation of rule $reasonable \leftarrow pacifist$, of P, on the strength of the truth of "$pacifist$" in the updated model, and therefore false. M_U is the restriction of $M_{\neg U} = \{reasonable, war, \neg pacifist\}$. We have that

$$Rejected(M_{\neg U}, P, U) = \{pacifist \leftarrow\}\,; \quad \uplus(M_{\neg U}, P, U) = P + U - \{pacifist \leftarrow\}$$

and $M_{\neg U}$ since not an answer-set of $\uplus(M_{\neg U}, P, U)$, M_U is not a <P,U>-justified update. $\Diamond$

3.1.1 Properties of <P,U>-Justified Updates

Given the definition of <P,U>-justified updates we can demonstrate the following properties:

Theorem 1 *Let Q be a normal logic program in the language $\mathcal{L}_\neg$, and $\emptyset$ an empty program. Then the answer-sets of Q are exactly any of: the <Q,$\emptyset$>-justified updates; the <$\emptyset$,Q>-justified updates; the <Q,Q>-justified updates.* $\Diamond$

Theorem 2 (Associativity of Updates) *Let P, Q and R be three normal logic programs in the language $\mathcal{L}_\neg$. Let $\uplus(M_\neg, P, U)$ be the program obtained as per Def.6, corresponding to the update of P by U given $M_\neg$, where $M_\neg$ is an interpretation of $\mathcal{L}_\neg$. If P updated by Q (resp. Q updated by R) has at least one <P,Q>-justified update (resp. <Q,R>-justified update) then $M_\neg$ is an answer-set of the program $\uplus(M_\neg, \uplus(M_\neg, P, Q), R)$ iff $M_\neg$ is an answer-set of the program $\uplus(M_\neg, P, \uplus(M_\neg, Q, R))$.* $\Diamond$

3.2 Update Transformation of a Normal Program

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 updates, according to Theorem 3 below. The updated program can then be used to compute them.

Definition 7 (Update transformation of a normal program) *Given an update program U in the language $\mathcal{L}_\neg$. For a logic program P in the language $\mathcal{L}_\neg$, its updated program P_U with respect to U , written in the extended language $\mathcal{L}_\neg + \{A', \neg\_A', A^U, \neg A^U : A \in \mathcal{L}\}$ 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 objective literal L by a new objective literal L^U;*
- *in the head of every rule of P_U originated in P replace atom A (resp. $\neg A$) by a new atom A' (resp $\neg\_A'$);*

-Include in P_U, for every atom A in $\mathcal{L}$, the defining rules:

$$\begin{array}{ll} A \leftarrow A', not\ \neg A^U; & A \leftarrow A^U; \\ \neg A \leftarrow \neg\_A', not\ A^U; & \neg A \leftarrow \neg A^U.\ \diamondsuit \end{array} \tag{4}$$

The above assumes that in the language $\mathcal{L}_\neg$ there are no symbols of the form A', $\neg\_A'$, A^U and $\neg A^U$. The defining rules establish that, after the update, a literal is either implied by inertia or forced by an update rule.

Example 3 *Consider the P and U of Example 2. The updated program P_U of P by U is (where the rules for A stand for all their ground instances):*

$$\begin{array}{lll} pacifist' \leftarrow; & war^U \leftarrow; & A \leftarrow A', not\ \neg A^U; \\ reasonable' \leftarrow pacifist; & A \leftarrow A^U; & \neg A \leftarrow \neg\_A', not\ A^U; \\ \neg pacifist^U \leftarrow war; & \neg A \leftarrow \neg A^U. & \end{array}$$

whose only answer-set (modulo A', A^U and explicitly negated atoms) is $M_1 = \{war\}$, coinciding with the only <P,U>-justified update determined in Example 2. $\diamondsuit$

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

Theorem 3 (Correctness of the update transformation) *Let P be a normal logic program in the language $\mathcal{L}_\neg$ and U an update program in the language $\mathcal{L}_\neg$. Modulo any primed, L^U and explicitly negated literals, the answer-sets of the updated program P_U are exactly the <P,U>-Justified Updates of P updated by U.* $\diamondsuit$

4 Iterated Updates

What happens if we want to update a program more than once? Since the definitions in the previous section apply only to programs without explicit negation in the body of their rules, and updated programs obtained via Def.7 include such negations in the body of inertia rules, those definitions cannot be used to update previously updated programs.

But if we take a closer look at the rôle of inertia rules, as well as the intuitive characterization of justified updates, we realize that inertia rules don't need to be updated. Indeed, given a sequence of update programs to be applied to an initial program (which, as we've seen in theorem 1, can itself be envisaged as an update program applied to the empty program), the justified updates should be answer-sets of a program composed of the rules of the last update program, together with some rules from each of the previous update programs, that haven't been overriden by a subsequent update. Equally, to construct a program whose answer-sets are the justified updates, we might just use the rules of each update program, together with some rules enacting inertia from one stage to the next.

In this section we will generalize the notions presented in the previous one to the case where we have a sequence of update programs. For this we start with a definition, allowing us to "time-stamp" updates.

Definition 8 (State Set) *Let the* State Set T *be an initial sequence of natural numbers* $T = \{0, 1, 2, ..., n\}$. *We call the members of* T *states, and the first element 0 the* initial state. $\diamondsuit$

4.1 S-justified Updates

We now characterize the set of interpretations that should be accepted as justified updates, at any given state, in a sequence of updates. They are called the S-justified updates. Like for <P,U>-justified updates, we start by defining the set of rules, of each update program, that should persist up to some state under consideration.

Definition 9 (Inertial Sub-Program of state t_1 at state t) *Consider an ordered sequence* $S = \{U_t : t \in T\}$, *indexed by* T, *of update logic programs* U_t *in the language* $\mathcal{L}_\neg$, *and* $M_\neg$ *an interpretation of* $\mathcal{L}_\neg$. *Let:*

$$Rejected(M_\neg, t_1, t) = \bigcup_{t_1 < t_2 \leq t} Rejected_i(M_\neg, t_1, t_2) \tag{5}$$

$$\begin{aligned} Rejected_i(M_\neg, t_1, t_2) = \{L \leftarrow body \in U_{t_1} : \hat{M}_\neg(body) = 1 \\ and\ \exists \neg L \leftarrow body' \in U_{t_2} : \hat{M}_\neg(body') = 1\} \end{aligned} \tag{6}$$

where L *is an objective literal. We then define the* Inertial Sub-Program of state t_1 at state t, $U^{inertial}(M_\neg, t_1, t)$ *as:*

$$U^{inertial}(M_\neg, t_1, t) = U_{t_1} - Rejected(M_\neg, t_1, t)\ \diamondsuit \tag{7}$$

Note that $Rejected(M_\neg, t, t) = \{\}$ and so $U^{inertial}(M_\neg, t, t) = U_t$.

Intuitively, the rules for some literal L that belong to $Rejected(M_\neg, t_1, t)$ are those that belong to U_{t_1} and, although their body is still verified by the model, there is a rule of a subsequent update program that overrides them, by contravening their conclusion.

Definition 10 (S-Justified Updates at a given state t) *Consider an ordered sequence* $S = \{U_t : t \in T\}$, *indexed by* T, *of update logic programs* U_t *in the language* $\mathcal{L}_\neg$, *with smallest element* U_{t_0}, *and* M *an interpretation of* $\mathcal{L}$. M *is a* S-Justified Update at a given state t *iff there is an interpretation* $M_\neg$ *of* $\mathcal{L}_\neg$ *such that* $M_\neg$ *is an answer-set of* $\uplus_S(t)$, *where*

$$\uplus_S(t) = \bigcup_{t_i \leq t} (U^{inertial}(M_\neg, t_i, t)) \tag{8}$$

and $t_i \in T$, *if there is a non-contradictory S-justified update at every state* t_i, $t_0 \leq t_i \leq t$. *Otherwise there is no S-justified update at state* t. $\diamondsuit$

Intuitively, if $M_\neg$ is an answer-set of the program consisting of the rules from the update program at state t, together with every rule belonging to a program of a previous state whose conclusion has not been contravened by a rule of a subsequent update program (up to t). The non-contradictoriness condition for every previous state ensures that we reached state t without passing a contradictory state. Addressing contradiction is beyond the scope of this work.

Note that if a rule is rejected by some rule, which itself is subsequently rejected, there is no need to reinstate the original rule because the rule rejecting the rejector rule takes it place.

Theorem 4 *Let $S = \{U_0, ..., U_n\}$ be an ordered sequence of update logic programs U_n in the language $\mathcal{L}_\neg$, with smallest element U_0. If there is at least one S-justified update at state $n-1$ then the S-justified updates at state n are exactly the* $<\uplus_S(n-1), U_n>$*-justified updates.* $\diamondsuit$

Corollary 1 (Generalization of <P,U>-justified updates) *Let P and U be update programs in the language $\mathcal{L}_\neg$. Let $T = \{0,1\}$ and $S = \{U_0, U_1\}$ such that $U_0 = P$ and $U_1 = U$. Then an interpretation M is an S-justified update at state 1 iff it is a* $<P,U>$*-justified update.* $\diamondsuit$

4.2 Iterated Update Transformation

Next we define a transformation which, given a sequence of update logic programs, produces a final logic program. It does so by introducing inertia rules to 'link' the intermediate programs in the given sequence. The final program, together with defining rules as will be seen below, has exactly as its models the S-justified Updates of the update sequence.

Definition 11 (Iterated Program Updates) *Let $S = \{U_t : t \in T\}$ be an ordered sequence of update logic programs U_t, indexed by T, in the language $\mathcal{L}_\neg$, with smallest element U_{t_0}. The* Iterated Update Program *P_S defined by S, in the extended language $\mathcal{L}^*_\neg = \mathcal{L}_\neg + \left\{A_t, \neg\_A_t, A^U_t, \neg\_A^U_t : A \in \mathcal{L}, t \in T\right\}$ is obtained via the operations:*

- All rules of $U_t \in S$ belong to P_S except that in the head of every rule one replaces atom A (resp. $\neg A$) by a new atom A^U_t (resp $\neg\_A^U_t$);

- Include in P_S, for every atom A of $\mathcal{L}$, and for any $t > t_0$, the defining rules:

$$\begin{array}{ll} A_t \leftarrow A_{t-1}, not\ \neg\_A^U_t; & A_t \leftarrow A^U_t; \\ \neg\_A_t \leftarrow \neg\_A_{t-1}, not\ A^U_t; & \neg\_A_t \leftarrow \neg\_A^U_t. \end{array} \tag{9}$$

- Include in P_S, for every atom A of $\mathcal{L}$, the rules:

$$A_{t_0} \leftarrow A^U_{t_0} \quad and \quad \neg\_A_{t_0} \leftarrow \neg\_A^U_{t_0}\ \diamondsuit \tag{10}$$

The above definition assumes that in the language $\mathcal{L}_\neg$ there are no symbols of the form A_t, $\neg\_A_t$, A^U_t and $\neg\_A^U_t$. Note that P_S does not contain any rules for atoms of $\mathcal{L}_\neg$, i.e. neither A nor $\neg A$ appear in the heads of the rules of P_S.

Now, to find out what is the result of the updates up to a given state t, we add the "current state" defining rules, to obtain the program at a given state t:

Definition 12 (Iterated Update Program at a current state t) *Consider a non-contradictory iterated update program P_S. The* Iterated Update Program at a given current state t $\in T$, $P_S(t)$, *is the program* P_S *augmented with the* current state *defining rules:*

$$A \leftarrow A_t \quad \text{and} \quad \neg A \leftarrow \neg\_A_t \tag{11}$$

for all $A \in \mathcal{L}$, *if* $P_S(r)$ *is non-contradictory for all* $r \in T, r < t$. *Otherwise* $P_S(t)$ *is not defined.* ♢

Example 4 *Let* $S = \{U_1, U_2, U_3\}$ *be an ordered sequence of programs such that:*

$$\begin{array}{lll} U_1 : pacifist \leftarrow; & U_2 : \neg pacifist \leftarrow war; & U_3 : \neg war \leftarrow . \\ \quad reasonable \leftarrow pacifist. & \quad war \leftarrow . & \end{array}$$

The iterated update program P_S *is the logic program (where the rules for* A *stand for all their ground instances and t=2,3):*

$$\begin{array}{lll} pacifist_1^U \leftarrow; & \neg\_war_3^U \leftarrow; & A_t \leftarrow A_{t-1}, not\ \neg\_A_t^U; \\ reasonable_1^U \leftarrow pacifist; & A_1 \leftarrow A_1^U; & \neg\_A_t \leftarrow \neg\_A_{t-1}, not\ A_t^U; \\ \neg\_pacifist_2^U \leftarrow war; & \neg\_A_1 \leftarrow \neg\_A_1^U; & A_t \leftarrow A_t^U; \\ war_2^U \leftarrow; & & \neg\_A_t \leftarrow \neg\_A_t^U. \end{array}$$

The iterated update program at a given current state t is P_S *together with the current state defining rules (11), so that* $P_S(1)$ *has the single answer-set* $M_1 = \{pacifist, reasonable\}$, $P_S(2)$ *has the single answer-set* $M_2 = \{war\}$, *and* $P_S(3)$ *has the single answer-set* $M_3 = \{pacifist, reasonable\}$ *(all answer-sets are modulo irrelevant elements). Note that since after* U_3 *there is no longer a state of war, the rule "*$\neg pacifist \leftarrow war$*" of* U_2 *has no effect and the truth value of "*$pacifist$*" is determined solely by the rule "*$pacifist \leftarrow$ *" of* U_1. ♢

The following theorem establishes that iterated program updates generalize program updates.

Theorem 5 (Generalization of Program Updates) *Let* P *and* U *be update programs in the language* $\mathcal{L}_\neg$. *Let* $T = \{0, 1\}$ *and let* $U_0 = P$ *and* $U_1 = U$. *Then the iterated update program* $P_S(1)$ *at state* $t = 1$ *is semantically equivalent to the updated program* P_U *of* P *by* U, *i.e. for every answer-set* $M_\neg$ *of* $P_S(1)$ *there is exactly an answer-set* $M'_\neg$ *of* P_U *such that* $M = M'$, *conversely, for every answer-set* $M'_\neg$ *of* P_U *there is exactly an answer-set* $M_\neg$ *of* $P_S(1)$ *such that* $M' = M$. ♢

The next theorem establishes the relationship between the iterated program and the S-justified updates at a given state t.

Theorem 6 (Correctness of the iterated UP at a state t) *Given an ordered sequence* $S = \{U_t : t \in T\}$ *of update logic programs* U_t *in the language* $\mathcal{L}_\neg$, *with smallest element* U_{t_0}. *Restricted to* $\mathcal{L}$, *the answer-sets of the iterated update program at a given current state* t, *are exactly the S-justified updates at state* t. ♢

5 Updating Normal Logic Programs under Partial Stable Semantics

In this section we extend the notions of logic program updating to the case where a partial stable semantics is used. To bring out the implications of such an extension in a simple way, it will be just presented for the case of normal logic programs, and of a single update. The extension to the iterated case, is similar to the one presented in the previous section.

As will be seen, we need only extend the characterization of <P,U>-justified updates. The update transformation used before for normal logic programs is valid for the three-valued case simply by using the partial stable semantics to compute models. For simplicity, we represent 3-valued interpretations as sets of objective and default literals.

5.1 Three-valued <P,U>-Justified Updates

The characterization of <P,U>-justified updates will have to suffer a modification to allow for partial iterpretations and models. To better understand the need for such modifications, and the intuition behind them, consider a very simple example:

Example 5 *Initial program P and update program U (next to each rule of U we exhibit the original update rule):*

$$P : a \leftarrow . \qquad U : \begin{array}{ll} \neg a \leftarrow c; & [out(a) \leftarrow in(c)] \\ c \leftarrow not\ c. & [in(c) \leftarrow out(c)] \end{array}$$

Any 3-valued U-justified update (as in [4]) of $M = \{a\}$ should be a 3-valued <P,U>-justified update as well because P contains just facts. According to [4], $M = \{\}$ is the only 3-valued U-justified update i.e., both **a** *and* **c** *are undefined. Looking at the rules of both P and U we note the impossibility of having a partial stable model (PSM) where* **a** *is undefined, of a program composed of those rules. This is so because having* $\neg$**a** *undefined doesn't affect the truth value of* **a**. $\diamondsuit$

The previous example suggests that the characterization of 3-valued <P,U>-justified updates can no longer be made to depend on a composition of the rules of P and U alone. We will have, under certain conditions, to explicitly undefine some objective literals. Informally, this should be done whenever a literal L is true due to a rule of P but $\neg L$ is undefined due to a rule of U. Formally:

Definition 13 (Three-valued Inertial Sub-Program) *Let P be a normal logic program in the language $\mathcal{L}_\neg$, U an update program in the language $\mathcal{L}_\neg$ and $M_\neg = \langle T_{M_\neg}, F_{M_\neg} \rangle$ a 3-valued interpretation of $\mathcal{L}_\neg$. Let:*

$$\begin{aligned} Rejected(M_\neg, P, U) = \{ & L \leftarrow body \in P : \hat{M}_\neg(body) = 1 \\ & and\ \exists \neg L \leftarrow body' \in U : \hat{M}_\neg(body') \neq 0\ \} \end{aligned} \tag{12}$$

where L is an objective literal. We define Three-valued Inertial Sub-Program *$P_{inertial}(M_\neg, P, U)$ as:*

$$P_{inertial}(M_\neg, P, U) = P - Rejected(M_\neg, P, U)\ \diamondsuit \tag{13}$$

Intuitively, the rules for some L that belong to $Rejected(M_\neg, P, U)$ are those that belong to the initial program but, although their body is still verified by the interpretation, there is an update rule that overrides them, by either truthifying or undefining its complementary literal.

Now we define the set of literals that are to be explicitly undefined:

Definition 14 (Undefining Rules) *Let P be a normal logic program in the language $\mathcal{L}_\neg$, U an update program in the language $\mathcal{L}_\neg$ and $M_\neg = \langle T_{M_\neg}, F_{M_\neg} \rangle$ a 3-valued interpretation of $\mathcal{L}_\neg$. The set of* Undefining Rules*, $Undefining(M_\neg, P, U)$ is:*

$$\begin{aligned} Undefining(M_\neg, P, U) = \{ & L \leftarrow \mathbf{u} : L \leftarrow body \in Rejected(M_\neg, P, U) \\ & and\ \nexists \neg L \leftarrow body' \in U : \hat{M}_\neg(body') = 1\ \}\ \diamondsuit \end{aligned} \tag{14}$$

$Undefining(M_\neg, P, U)$ contains rules to undefine precisely those literals L such that there is an update rule undefining its complementary literal $\neg L$, and none truthifying it, which are required to reject rules for L. Recall that $\mathbf{u}$ stands for the propositional symbol with the property of being undefined in every interpretation. Finally, the three-valued <P,U>-justified updates are defined as:

Definition 15 (3-valued <P,U>-Justified Updates) *Let P be a normal logic program and U an update program, both in the language $\mathcal{L}_\neg$. Let M be a 3-valued interpretation of the language $\mathcal{L}$. M is a* 3-valued <P,U>-Justified Update *of P updated by U, iff there is an interpretation $M_\neg$ of $\mathcal{L}_\neg$ such that $M_\neg$ is a partial stable model of $\uplus(M_\neg, P, U)$, where*

$$\uplus(M_\neg, P, U) = U + P_{inertial}(M_\neg, P, U) + Undefining(M_\neg, P, U) \ \Diamond \tag{15}$$

The next example shows the rôle played by $Undefining(M_\neg, P, U)$ when determining the 3-valued <P,U>-Justified Updates.

Example 6 *Consider the program P of the pacifist situation of Example 1 but where now the 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:*

$$\begin{array}{llll} P: & pacifist \leftarrow; & U: & \neg pacifist \leftarrow war; \\ & reasonable \leftarrow pacifist. & & peace \leftarrow not\ war; \\ & & & war \leftarrow not\ peace. \end{array}$$

Intuitively, examining the two 2-valued <P,U>-justified updates:

$$\begin{array}{l} M_1 = \{pacifist, reasonable, peace, not\ war\} \\ M_2 = \{not\ pacifist, not\ reasonable, not\ peace, war\} \end{array}$$

one should expect the interpretation $M_3 = \{\}$ to be a 3-valued <P,U>-justified update. Let's check if it is so. M_3 is the restriction of $M_{\neg 3} = \{\}$. Since the truth value of war in the rule $\neg pacifist \leftarrow war$ is not false, we have that the rule $pacifist \leftarrow$ should be rejected, i.e.

$$Rejected(M_{\neg 3}, P, U) = \{pacifist \leftarrow\}$$

Now, we have to include an undefining rule for pacifist just in case there aren't any rules in U for $\neg pacifist$ with a true body. Since that is the case we have

$$Undefining(M_{\neg 3}, P, U) = \{pacifist \leftarrow \mathbf{u}\}$$

and we obtain the program:

$$\begin{array}{lll} \uplus(M_{\neg 3}, P, U): & pacifist \leftarrow \mathbf{u}; & \neg pacifist \leftarrow war; \\ & reasonable \leftarrow pacifist; & peace \leftarrow not\ war; \\ & & war \leftarrow not\ peace. \end{array}$$

Since $M_{\neg 3}$ is a PSM of $\uplus(M_{\neg 3}, P, U)$, M_3 is a 3-valued <P,U>-justified update. $\Diamond$

The theorem below shows that the (2-valued) <P,U>-justified updates are a special case of the 3-valued <P,U>-justified updates:

Theorem 7 (Generalization of <P,U>-justified updates) *Whenever dealing with total interpretations, the 3-valued <P,U>-justified update definition reduces to that of (2-valued) <P,U>-justified update.* $\Diamond$

The 3-valued <P,U>-justified updates can be determined from the updated program constructed from the initial program and the update program, as per Def. 7, if we consider its partial stable models (or WFSX models[3]).

Example 7 *Take P and U of Ex.6. The updated program P_U of P by U is (where the rules for A stand for all their ground instances):*

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

whose partial stable models (modulo A', A^U and explicitly negated atoms) are:

$$\begin{array}{ll} M_1 = \{pacifist, reasonable, peace, not\ war\} & M_3 = \{\} \\ M_2 = \{not\ pacifist, not\ reasonable, not\ peace, war\} & \end{array}$$

coinciding with the three 3-valued <P,U>-justified updates determined in ex.6. $\diamondsuit$

The following theorem establishes the relationship between the partial stable models of the update transformation of a program and its 3-valued <P,U>-justified updates.

Theorem 8 (Correctness of the update transformation) *Let P be a normal logic program and U an update program, both in the language $\mathcal{L}_\neg$. Modulo any primed, L^U and explicitly negated literals, the partial stable models of the updated program P_U are exactly the 3-valued <P,U>-Justified Updates of P updated by U.* $\diamondsuit$

6 Conclusions and Future Work

Throughout this paper we have motivated and formalized the generalization of 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. This was achieved both for the stable as well as for the partial stable semantics. Another important result set forth in this work is the extension to the case where we want to update a given program more than once, i.e. iterated updating. This is important inasmuch as it 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[2].

Program updating is a new and crucial notion deserving future foundational work and opening up a whole new range of applications and problems such as:

Software specification: One of the major tasks in the software industry is the updating of software. This process would be made much easier if we could just simply specify the update i.e. the parts of the program specific to the new version, and automatically generate the updated version.

Generalize the scope of update rules: Just as interpretation updates were extended to updating by means of arbitrary inference rules [13], logic program updating should also be generalized, possibly using super logic programs [5].

[2]For a generalization of the results presented here for the case of extended (with explicit negation) logic programs, the reader is referred to [9].

Reasoning about the past: Since no information is lost during the update process, i.e. after any iteration we can still characterize any previous state, we can use this framework to reason about the past in various ways:

- The scope of update rules could be extended to allow for updates to depend on some past state. This would be done by allowing the body of update rules to refer to previous states, thus increasing the expressibility of updates.

- The iterated updated program at state t can be used to directly determine *what might have happened in the past with present knowledge.* This could be done by checking the truth value of atoms A_{t_1} and $\neg\_A_{t_1}$ where t_1 refers to the previous state we want reason about.

Self updates Throughout this work we assumed that updates were externally driven. This need not be so. We could allow for internal self updating. This would be of the form of update rules scheduled to be triggered at some state, under certain conditions. Informally, an example of such kind of self update rules would be:

$$up\,[in(L) \leftarrow in(L_1), out(L_2)] \leftarrow in(L_3), out(L_4)$$

with the intended meaning of: if at state t $in(L_3), out(L_4)$ is verified, perform the update $in(L) \leftarrow in(L_1), out(L_2)$ at state $t+1$.

Improved expressibility could be obtained by combining these self updates with the externally driven ones. A simple application example of this combination would be the modeling of an automata receiving external input by means of an update, changing its state by means of a self update, and performing some output. By allowing internal self update rules one opens up the concept of self-evolving programs, that could be used to model artificial life.

Dealing with contradiction: Contradiction may have its origin at several stages of the update process: both the program to be updated and the update program may be contradictory, and even if they are both non-contradictory, the updated program may be so. One might apply well known contradiction avoidance/removal techniques [2][3] to updates. Furthermore updates should be generalized to include integrity constraints.

Also updates could be used as a means to remove contradiction. Note that the update of a contradictory program need not necessarily be contradictory.

Reasoning about actions: Updates pave the way for the dynamics of logic programming as opposed to its statics which has received most attention till now. Areas such as temporal databases, production systems, event calculus and situation calculus can benefit and be fertilized by this updates theory.

Acknowledgements

This work was partially supported by PRAXIS XXI project MENTAL. The work of João Alexandre Leite was partially supported by PRAXIS XXI scholarship no. BD/13514/97. We thank the three anonymous referees for their helpful comments.

References

[1] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinski and T. C. Przymusinski. *Dynamic Logic programming.* In Sixth International Conference on Principles of Knowledge Representation and Reasoning, KR'98, Morgan Kaufmann 1998.

[2] J. J. Alferes and L. M. Pereira. *Contradiction: when avoidance equals removal, I and II.* In Dyckhoff, editor, 4th ELP, volume 798 of LNAI. Springer-Verlag, 1994.

[3] J. J. Alferes and L. M. Pereira *Reasonning with Logic Programming.* LNAI 1111, Berlin. Springer-Verlag, 1996.

[4] J. J. Alferes and 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.

[5] S. Brass, J. Dix and T. Przymusinski. *Super Logic Programs.* In L. C. Aiello, J. Doyle, and S. C. Shaphiro, editors, Principles of Knowledge Representation and Reasoning, Proc. of the Third Int'l Conf (KR96), pages 529-541. San Francisco, CA, Morgan-Kaufmann, 1996.

[6] 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.

[7] 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., pages 230-237, Morgan Kaufmann 1991.

[8] A. Keller and M. Winslett Wilkins. *On the use of an extended relational model to handle changing incomplete information.* IEEE Trans. on Software Engineering, SE-11:7, pages 620-633, 1985.

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

[10] J. A. Leite and L. M. Pereira. *Generalizing updates: from models to programs.* In LPKR'97: ILPS'97 workshop on Logic Programming and Knowledge Representation, Port Jefferson, NY, USA, October 13-16, 1997.

[11] 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.

[12] 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."*.

[13] 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.

[14] M. Winslett. *Reasoning about action using a possible models approach.* In AAAI'88, pages 89-93, 1988.