A Compilation of Updates plus Preferences*

José Júlio Alferes†, Pierangelo Dell'Acqua†*, and Luís Moniz Pereira†

† Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
{jja,lmp}@di.fct.unl.pt

* Department of Science and Technology - ITN
Linköping University, 601 74 Norrköping, Sweden
pier@itn.liu.se

Abstract. We show how to compile programs formalizing update plus preference reasoning into standard generalized logic programs and show the correctness of the transformation.

1 Introduction

Update reasoning and *preference reasoning* are two pervasive forms of reasoning. Update reasoning is used to model dynamically evolving worlds. Given a piece of knowledge describing the world, and given a change in the world, the problem is how to modify the knowledge to cope with that change. The key issue is how to accommodate, in the represented knowledge, any changes in the world. In this setting it may well happen that change in the world contradicts previous knowledge, i.e. the union of the previous knowledge with the representation of the new knowledge has no model. It is up to updates to remove from the prior knowledge representation a piece that changed, and to replace it by the new one.

The addition of mechanisms for handling preferences in Logic Programming has proven essential in several different application areas, such as legal reasoning [14], data cleaning [8], and intelligent agents for e-commerce [11]. In general, preference information is used along with incomplete knowledge. In such a setting, due to the incompleteness of the knowledge, several models may be possible. Preference reasoning acts by choosing among those possible models. A classical example is the birds-fly problem where the incomplete knowledge contains the rules that "birds normally fly" (1) and "penguins normally don't fly" (2). Given an individual which is both a penguin and a bird, two models are possible: one model obtained by using the first rule, and the other obtained by using the second rule. Preference information among the rules can then be used to choose one model. Cf. [5, 6] for additional motivation, comparisons, applications, and references.

* This work was partially supported by POCTI project 40958 "FLUX - FleXible Logical Updates". The second author acknowledges a Research Grant by the *Stiftelsen Lars Hiertas Minne*.

Alferes and Pereira [3] proposed an approach that combines these two separate forms of reasoning in the context of Logic Programming (for an overview see [4]). They showed how they complement each other, in that preferences select among pre-existing models, and updates actually create new models. Moreover, preferences may be enacted on the results of updates, and updates may be pressed into service for the purpose of changing preferences. Brewka and Eiter's preferred stable models [7] are generalized in [3], for the ground case, to take into account the language of generalized logic programs and to allow for updating the priority information.

Consider the following example, where rules as well as preferences change over time. It requires a combination of preferences and updates, including the updating of preferences themselves. This example could form the basis of an e-commerce application where, depending on the user's preferences, a number of alternative choices could be offered. Such an application would require that the knowledge base rules as well as the user's preferences change over time. An example would be a pay-TV company that wants to target its marketing by maintaining user's profiles in order to inform its users about their favourite programs.

Example. (TV-program) Suppose a scenario where a user, say Stefano, watches programs about football, tennis or news. (1) In the initial situation, being a typical Italian, Stefano prefers both football and tennis to news, and in case of international competitions he prefers tennis to football. In this situation, Stefano has two alternative TV programmes equally preferable: football and tennis. (2) Next, suppose that a US-open tennis competition takes place. Now, Stefano's favourite programme is tennis. (3) Finally, suppose that Stefano's preferences change and he becomes interested in international news. Then, in case of breaking news he will prefer news over both football and tennis.

In this paper we show how to compile programs formalizing update plus preference reasoning into standard generalized logic programs. Lack of space prevents us from showing here the correctness of the transformation, though it can be found in [1]. The proposed transformation has the advantage of being modular: it is a combination of a transformation for update reasoning and a transformation for preference reasoning. Moreover, we have implemented this transformation and tested on several examples.

2 Logic Programming Framework

We consider propositional Horn theories. In particular, we represent default negation $not\,A$ as standard propositional variable (atom). Suppose that $\mathcal{K}$ is an arbitrary set of propositional variables whose names do not begin with a "not". By the propositional language $\mathcal{L}_{\mathcal{K}}$ generated by $\mathcal{K}$ we mean the language whose set of propositional variables consists of $\{A, not\,A : A \in \mathcal{K}\}$. Atoms $A \in \mathcal{K}$ are called *objective atoms* while the atoms $not\,A$ are called *default atoms*. From the definition it follows that the two sets are disjoint. Objective and default atoms are generically called *literals*.

Definition 1 (Generalized logic program). *A generalized logic program in the language $\mathcal{L}_{\mathcal{K}}$ is a finite or infinite set of ground generalized rules:*

$$L_0 \leftarrow L_1, \ldots, L_n \qquad (n \geq 0)$$

where each L_i is a literal over $\mathcal{L}_{\mathcal{K}}$.

By $head(r)$ we mean L_0, by $body(r)$ the set of literals $\{L_1, \ldots, L_n\}$, by $body^+(r)$ the set of all objective atoms in $body(r)$, and by $body^-(r)$ the set of all default atoms in $body(r)$. We refer to $body^+(r)$ as the prerequisites of r. Whenever a literal L is of the form $not\,A$, $not\,L$ stands for the objective atom A.

The semantics of generalized logic programs [12, 13] is defined as a generalization of the stable model semantics [10]. Here we use the definition that appeared in [2] (proven there equivalent to [12, 13]).

Definition 2 (Default assumptions). *Let M be an interpretation of P. Then:*

$$Default(P, M) = \{not\,A \mid \nexists r \in P : head(r) = A \text{ and } M \models body(r)\}.$$

Definition 3 (Stable Models of Generalized Programs). *Let P be a generalized logic program and M an interpretation of P. M is a (regular) stable model of P iff $M = least(P \cup Default(P, M))$.*

To express preference information in logic programs, we introduce the notion of priority rule and prioritized logic program. Let $N = \{n_{r_1}, \ldots, n_{r_k}\}$ be a name set containing a unique name for every generalized rule in the language $\mathcal{L}_{\mathcal{K}}$. Given a rule r, we write n_r to indicate the constant in N that names r. Let $<$ be a binary predicate symbol[1] whose set of constants is N. $n_r < n_u$ means that rule r is preferred to rule u. Denote by $\bar{\bar{\mathcal{K}}}$ the following superset of the set $\mathcal{K}$ of propositional variables:

$$\bar{\bar{\mathcal{K}}} = \mathcal{K} \cup \{n_r < n_u : \{n_r, n_u\} \subseteq N\}.$$

Definition 4 (Priority rule). *A priority rule over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$ is a generalized rule of the form:*

$$Z \leftarrow L_1, \ldots, L_n \qquad (n \geq 0)$$

where Z is a literal of the form $n_r{<}n_u$ or its default complement, $not\ n_r{<}n_u$, and each L_i is a literal over $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$.

Note that the set of constants of $<$ does not include $<$ itself (that is, $< \notin N$), and that $<$ can only occur in the head and (possibly) in the body of priority rules.

Definition 5 (Prioritized logic program). *Let P be a generalized logic program over the language $\mathcal{L}_{\mathcal{K}}$ and R a set of priority rules over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Then (P, R) is a prioritized logic program.*

The generalized rules in P formalize the domain knowledge while the priority rules in R express preferences among the rules in P.

[1] In order to establish the preferred stable models (cf. Def. 14), we require the relation induced by $<$ to be a well-founded, strict partial ordering on program rules.

3 Dynamic Prioritized Programs

In this section we recall the approach of Dynamic Logic Programming [2] that can be used to model the evolution of prioritized logic programs through sequences of updates (including the update of priority rules).

In Dynamic Logic Programming, sequences of generalized programs $P_1 \oplus \cdots \oplus P_s$ are given. Intuitively a sequence may be viewed as the result of, starting with program P_1, updating it with program P_2, ..., and updating it with program P_s. In such a view, dynamic logic programs are to be used in knowledge bases that evolve. New rules (coming from new, or newly acquired, knowledge) can be added at the end of the sequence, bothering not whether they conflict with previous knowledge. The role of Dynamic Logic Programming is to ensure that these newly added rules are in force, and that previous rules are still valid (by inertia) as far as possible, i.e. they are kept for as long as they do not conflict with newly added ones. In the remaining, let $S = \{1, \ldots, s, \ldots\}$ be a set of natural numbers. We call the elements $i \in S$ *states*.

Definition 6 (Dynamic Logic Program). *Let S be a set of states. A dynamic logic program $\bigoplus\{P_i : i \in S\}$ is a set of a generalized logic programs P_i indexed by the states i in S.*

The semantics of dynamic logic programs is defined according to the rationale above. Given a model M of the program P_s, start by removing all the rules from previous programs whose head is the complement of some later rule with true body in M (i.e. by removing all rules which conflict with more recent ones). All other persist through by inertia. Then, as for the stable models of a single generalized program, add facts *not* A for all atoms A which have no rule at all with true body in M, and compute the least model. If M is a fixpoint of this construction, M is a stable model of the sequence up to P_s.

Definition 7 (Rejected rules). *Let $s \in S$ be a state. Let $P = \bigoplus\{P_i : i \in S\}$ be a dynamic logic program over the language $\mathcal{L}_{\mathcal{K}}$ and M an interpretation over $\mathcal{L}_{\mathcal{K}}$. Then:*

$$\begin{aligned}&\mathit{Reject}(s, M, P) = \\ &\quad \{r \in P_i \mid \exists r' \in P_j, \mathit{head}(r) = \mathit{not\,head}(r'), i < j \leq s \text{ and } M \models \mathit{body}(r')\}.\end{aligned}$$

To allow for querying a dynamic program at any state s, the definition of stable model is parameterized by the state.

Definition 8 (Stable Models of a DLP at state s). *Let $s \in S$ be a state. Let $P = \bigoplus\{P_i : i \in S\}$ be a dynamic logic program over the language $\mathcal{L}_{\mathcal{K}}$ and M an interpretation over $\mathcal{L}_{\mathcal{K}}$. M is a stable model of P at state s iff:*

$$M = \mathit{least}([\mathcal{P} - \mathit{Reject}(s, M, P)] \cup \mathit{Default}(\mathcal{P}, M))$$

where $\mathcal{P} = \bigcup_{i \leq s} P_i$.

To allow the updating the priority rules, the notion of dynamic prioritized programs has been introduced into Dynamic Logic Programming. Instead of a sequence of programs representing knowledge, the idea is to use a sequence of pairs:

of programs representing knowledge, and of programs describing the priority relation among rules of the knowledge representation. In general, an update of the priority rules may depend on some other predicate. To permit this generality, priority rules are allowed to refer to predicates defined in the programs that represent knowledge.

Definition 9 (Dynamic Prioritized Program). *Let S be a set of states. Let $P = \{P_i : i \in S\}$ be a dynamic logic program over the language $\mathcal{L}_{\mathcal{K}}$. Let $R = \{R_i : i \in S\}$ a dynamic logic program, where each R_i is a set of priority rules over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Then, $\bigoplus\{(P_i, R_i) : i \in S\}$ is a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$.*

The next example illustrates the use of dynamic prioritized programs.

Example 1. The TV-program example presented in the Introduction can be formalized via a dynamic prioritized program $\bigoplus\{(P_i, R_i) : i \in S\}$ with $S = \{1, 2, 3\}$. We abbreviate football by f, tennis by t, news by n, breaking news by bn and US-open by us. We use n_i to name a rule r_i.

$$P_1 = \left\{ \begin{array}{ll} f \leftarrow not\, t, not\, n & (r_1) \\ t \leftarrow not\, f, not\, n & (r_2) \\ n \leftarrow not\, f, not\, t & (r_3) \end{array} \right\} \qquad R_1 = \left\{ \begin{array}{l} n_1 < n_3 \\ n_2 < n_3 \\ n_2 < n_1 \leftarrow us \\ x < y \leftarrow x < z, z < y \end{array} \right\}$$

$$P_2 = \left\{\, us \quad (r_4) \,\right\} \qquad R_2 = \{\}$$

$$P_3 = \left\{\, bn \quad (r_5) \,\right\} \qquad R_3 = \left\{ \begin{array}{l} not\,(n_1 < n_3) \leftarrow bn \\ not\,(n_2 < n_3) \leftarrow bn \\ n_3 < n_1 \leftarrow bn \\ n_3 < n_2 \leftarrow bn \end{array} \right\}$$

The stable models of a dynamic prioritized program P are defined in terms of the stable models of the dynamic logic program obtained by the union of all the P_is and R_is in P. Consequently, the preference information expressed by the priority rules in the R_is is not taken into account here.

Definition 10 (Stable Models of a DPP at state s). *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$ and M an interpretation over $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. M is a stable model of P at state s iff M is a stable model of the dynamic logic program $\bigoplus\{P_i \cup R_i : i \in S\}$ at state s.*

Example 2. Let P be the dynamic prioritized program of Example 1. At state 2, P has three stable models:

$$M_{21} = \{f\} \cup X \qquad M_{22} = \{t\} \cup X \qquad M_{23} = \{n\} \cup X$$

where $X = \{us, n_1 < n_3, n_2 < n_3, n_2 < n_1\}$. At state 3, the stable models are:

$$M_{31} = \{f\} \cup Y \qquad M_{32} = \{t\} \cup Y \qquad M_{33} = \{n\} \cup Y$$

where $Y = \{bn, us, n_2 < n_1, n_3 < n_1, n_3 < n_2\}$.

4 Compiling Dynamic Programs

A transformational semantics for dynamic logic programs is presented in [2]. According to it, a sequence of generalized logic programs is translated into a single generalized program whose stable models are in one-to-one correspondence with the stable models of the dynamic logic program. In this section we adapt that transformation to cope with priority rules, and we add new generalized rules to make explicit which rules in the program are rejected. These modifications make the transformation suitable to be extended further to incorporate preference information (cf. Section 6).

By $\widehat{\mathcal{K}}$ we denote the following superset of the set $\bar{\bar{\mathcal{K}}}$ of propositional variables: $\widehat{\mathcal{K}} = \bar{\bar{\mathcal{K}}} \cup \{A^-, A_i, A_i^-, A_{F_i}, A_{F_i}^- : A \in \bar{\bar{\mathcal{K}}}, i \in S \cup \{0\}\} \cup \{\text{reject}(n_r) : n_r \in N\}$.

Definition 11. *Let s be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Then $DLP(s, P)$ is the normal logic program over the language $\mathcal{L}_{\widehat{\mathcal{K}}}$ consisting of the following rules.*

(RP) Rewritten program rules:

$$A_{F_i} \leftarrow A_1, \ldots, A_n, A_{n+1}^-, \ldots, A_m \tag{1}$$

or

$$A_{F_i}^- \leftarrow A_1, \ldots, A_n, A_{n+1}^-, \ldots, A_m^- \tag{2}$$

for any rule:

$$A \leftarrow A_1, \ldots, A_n, not\, A_{n+1}, \ldots, not\, A_m$$

respectively, for any rule:

$$not\, A \leftarrow A_1, \ldots, A_n, not\, A_{n+1}, \ldots, not\, A_m$$

in the program $F_i = P_i \cup R_i$, where $i \in S$. The rewritten rules are obtained from the original ones by replacing atoms A (resp. $not\, A$) occurring in their heads by the atoms A_{F_i} (resp. $A_{F_i}^-$) and by replacing default atoms $not\, A_i$ in their bodies by A_i^-.

(UR) Update rules:

$$A_i \leftarrow A_{F_i} \tag{3}$$

$$A_i^- \leftarrow A_{F_i}^- \tag{4}$$

for all objective atoms $A \in \bar{\bar{\mathcal{K}}}$ and all $i \in S$. The update rules state that an atom A must be true (resp. false) in the state $i \in S$ if it is true (resp. false) in the updating program F_i.

(IR) Inheritance rules:

$$A_i \leftarrow A_{i-1}, not\, A_{F_i}^- \tag{5}$$

$$A_i^- \leftarrow A_{i-1}^-, not\, A_{F_i} \tag{6}$$

for all objective atoms $A \in \bar{\bar{\mathcal{K}}}$ and all $i \in S$. The inheritance rules say that an atom A is true (resp. false) in a state $i \in S$ if it is true (resp. false) in the previous state $i - 1$ and it is not forced to be false (resp. true) by the updating program F_i.

(DR) Default Rules:

$$A_0^- \tag{7}$$

for all objective atoms $A \in \bar{\bar{\mathcal{K}}}$. Default rules describe the initial state 0 by making all objective atoms initially false.

(RE) Rejection rules:

$$\mathrm{reject}(n_r) \leftarrow A_{F_t}^- \tag{8}$$

or

$$\mathrm{reject}(n_r) \leftarrow A_{F_t} \tag{9}$$

for any rule:

$$A \leftarrow A_1, \ldots, A_n, \mathit{not}\, A_{n+1}, \ldots, \mathit{not}\, A_m$$

respectively, for any rule:

$$\mathit{not}\, A \leftarrow A_1, \ldots, A_n, \mathit{not}\, A_{n+1}, \ldots, \mathit{not}\, A_m$$

r in the program $F_i = P_i \cup R_i$, and for all $t \in S$ such that $i < t \leq s$. Rejection rules state that a rule r in the program F_i will be rejected if it is forced to be false (resp. true) by a subsequent updating program F_t.

(CS$_s$) Current State Rules:

$$A \leftarrow A_s \tag{10}$$

$$A^- \leftarrow A_s^- \tag{11}$$

$$\mathit{false} \leftarrow A, A^- \tag{12}$$

for all objective atoms $A \in \bar{\bar{\mathcal{K}}}$. Current state rules specify the current state s in which the dynamic prioritized program is being evaluated and determine the values of the atoms A, A^- and $\mathit{not}\, A$.

The next theorem proves the correctness of the $DLP(s, P)$ transformation. The proof of this and the next results can be found in [1].

Theorem 1 (Soundness and Completeness). *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. The (regular) stable models of $DLP(s, P)$, restricted to $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$, coincide with the stable models of P at state s.*

The following proposition shows that an atom $\mathrm{reject}(n_r)$ belongs to a stable model M of $DLP(s, P)$ iff the rule r is effectively rejected by the model.

Proposition 1. *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Let M be a (regular) stable model of $DLP(s, P)$ and $J = M \cap \mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Then* $\mathrm{reject}(n_r) \in M$ *iff $r \in \mathit{Reject}(s, J, Q)$, where $Q = \bigoplus\{P_i \cup R_i : i \in S\}$.*

5 Preferred Stable Models

We recapitulate the notion of preferred stable model presented in [3]. Intuitively, the priority information specified by the R_is in a dynamic prioritized program P is used to prefer among the stable models of P. This is achieved by first deleting (from P) all the rejected rules according to updates and then by deleting all the rules that are unpreferred according to preference information. The set of unpreferred rules contains (1) the rules defeated by the head of some more preferred rule, (2) the rules that attack a more preferred rule, and (3) the unsupported rules.

Definition 12 (Unsupported rules). *Let P be a set of generalized rules over the language $\mathcal{L}_{\overline{\mathcal{K}}}$ and M an interpretation over $\mathcal{L}_{\overline{\mathcal{K}}}$. Then:*

$$Unsup(P, M) = \{r \in P : M \models head(r) \text{ and } M \not\models body^-(r)\}.$$

Definition 13 (Unpreferred generalized rule). *Let P be a set of generalized rules over the language $\mathcal{L}_{\overline{\mathcal{K}}}$ and M an interpretation over $\mathcal{L}_{\overline{\mathcal{K}}}$. $Unpref(P, M)$ is a set of unpreferred generalized rules of P and M iff:*

$$Unpref(P, M) = least(Unsup(P, M) \cup \mathcal{X})$$

where:

$$\begin{aligned} \mathcal{X} = \{r \in P \mid\ & \exists r' \in (P - Unpref(P, M)) \text{ such that:} \\ & M \models r' < r,\ M \models body^+(r') \text{ and} \\ & [\ not\ \ head(r') \in body^-(r) \text{ or} \\ & \qquad (not\ \ head(r) \in body^-(r'),\ M \models body(r))\ ]\ \}. \end{aligned}$$

Note that it is not possible to express preferences among the rules for predicate $<$. Therefore, no rule r for $<$ in P can be unpreferred, that is, $r \notin Unpref(P, M)$, for any set P of generalized rules and any interpretation M of P.

Definition 14 (Preferred Stable Models at state s). *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program and M a stable model of P at state s. Then M is a preferred stable model of P at state s iff:*

- $\forall r : (r < r) \notin M$
- $\forall r_1, r_2, r_3 :$ *if* $(r_1 < r_2) \in M$ *and* $(r_2 < r_3) \in M$ *then* $(r_1 < r_3) \in M$
- $M = least(\,[\,\mathcal{X}_s - Unpref(\mathcal{X}_s, M)\,] \cup Default(\mathcal{PR}, M)\,)$

where $\mathcal{PR} = \bigcup_{i \leq s}(P_i \cup R_i)$, $Q = \bigoplus\{P_i \cup R_i : i \in S\}$, $\mathcal{X}_s = \mathcal{PR} - Reject(s, M, Q)$.

Notice that the definition above requires M to be a stable model of P in order for M to be a preferred stable model of P. This reflects the requirement that preference information is intended to select among alternative stable models of P. In general, there may exist several preferred stable models. If the priority relation induced by the priority rules is a total order over the generalized rules in P, then P will admit at most one preferred stable model.

Def. 12 of unsupported rules differs from the original one [3] in that we do not require the positive part of the body of r to be true in the model for r to be unsupported. Though this new definition may increase the number of unsupported rules, the preferred stable models are not affected. In fact, any extra rule r' in the new definition does not contribute to a model (since its body is false) nor can it be used to unprefer a less priority rule r (cf. Def. 13 where for r to be unpreferred by r', it is required that $M \models \mathit{body}^+(r')$).

Example 3. Let P be the dynamic prioritized program of Example 1. At state 2, P has a unique preferred stable model $M_{22} = \{t, us, n_1 < n_3, n_2 < n_3, n_2 < n_1\}$. At state 3, the preferred stable model of P is $M_{33} = \{n, bn, us, n_2 < n_1, n_3 < n_1, n_3 < n_2\}$.

6 Compiling Dynamic Prioritized Programs

In this section, we show how the Updates Plus Preferences approach [3] for dynamic prioritized programs can be encoded within standard stable model semantics. The encoding consists essentially of two parts: the encoding $DLP(s, P)$ for updating generalized and priority rules, and the new encoding $\tau(r)$ for taking into consideration preference information. The encoding $\tau(r)$ is based on the encoding developed in [9] for preferences alone. This encoding relies on several predicates that enable us to detect when a rule r has been applied $\mathrm{ap}(n_r)$, blocked $\mathrm{bl}(n_r)$, unsupported $\mathrm{ko}(n_r)$, and a predicate $\mathrm{ok}(n_r)$ that enables application of rules. If r is OK, then r can be applied or found to be inapplicable, that is, r is blocked. $\mathrm{ok}(n_r)$ relies on an auxiliary predicate $\mathrm{ry}(n_r, n_u)$. The idea of the translation is to control the order of rule application with respect to the priority information. This is achieved in the following way. If a rule r is preferable to a rule u (i.e., $n_r < n_u$), then $\mathrm{ok}(n_u)$ holds if r is OK and either r has been applied (i.e., $\mathrm{ap}(n_r)$) or found to be inapplicable (i.e., $\mathrm{bl}(n_r)$). Basically, the translation delays the consideration of less preferred rules until the applicability question has been settled for higher ranked rules. If there exists a rule in the program that cannot be proved OK, then the model does not respect the preference information and therefore it is not preferred.

In order to combine the $\tau(r)$ encoding with the $DLP(s, P)$ encoding, we have adapted the encoding in [9] to take into account the language $\mathcal{L}_{\widehat{\mathcal{K}}}$ of $DLP(s, P)$. Thus, we have substituted each default atom *not* A in a rule r with A^-. As in the updating phase rules can be rejected, and therefore cannot be used within the preferring phase, we need to consider them not in $\tau(r)$. This is achieved via the predicate $\mathrm{reject}(n_r)$. For the head of a rule r to be true, we further require that r is not rejected. Moreover, a rule is only enabled to be applicable if each more preferred rule has been applied, has been blocked or has been rejected. The integrity constraint in $\tau(r)$ rules out unpreferred models. A model is unpreferred if there exists a rule that is not rejected in the updating phase and cannot be proved OK.

By $\overline{\mathcal{K}}$ we denote the following superset of the set $\mathcal{K}$ of propositional variables: $\overline{\mathcal{K}} = \widehat{\mathcal{K}} \cup \{\tilde{A}, \tilde{A}^- : A \in \mathcal{K}\} \cup \{\mathrm{ko}(n_r), \mathrm{ap}(n_r), \mathrm{bl}(n_r), \mathrm{ry}(n_r, n_u) : \{n_r, n_u\} \subseteq N\}$.

Given a rule r, we write $\tilde{r}$ to indicate the rule obtained from r by replacing each objective atom A occurring in r with $\tilde{A}$ and each default atom $not\,A$ with $not\,\tilde{A}$. For instance, if r is the rule $not\,A \leftarrow A_1, \ldots, A_n, not\,A_{n+1}, \ldots, not\,A_m$, then $\tilde{r}$ is $not\,\tilde{A} \leftarrow \tilde{A}_1, \ldots, \tilde{A}_n, not\,\tilde{A}_{n+1}, \ldots, not\,\tilde{A}_m$.

Let $\lceil . \rceil$ be a function from literals to objective atoms, where $\lceil A \rceil = A$ for every objective atom A and $\lceil not\,A \rceil = A^-$ for every default atom $not\,A$. We abbreviate $\lceil L_1 \rceil, \ldots, \lceil L_n \rceil$ by $\lceil L_1, \ldots, L_n \rceil$.

Definition 15. *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Let $\mathcal{P} = \bigcup_{i \le s} P_i$. Suppose $\mathcal{P} = \{r_1, \ldots, r_k\}$. Then $\Gamma(s, P)$ is the following generalized logic program over the language $\mathcal{L}_{\overline{\mathcal{K}}}$:*

$$\Gamma(s, P) = DLP(s, P) \cup \bigcup_{r \in \mathcal{P}} \tau(r) \cup DA \cup SPO.$$

$\tau(r)$ **Rules:** *consists of the following collection of rules, for $A \in body^+(r)$, $not\,B \in body^-(r)$ and any rule $u \in \mathcal{P}$:*

$$\begin{aligned}
\lceil head(\tilde{r}) \rceil &\leftarrow \mathrm{ap}(n_r), not\,\mathrm{reject}(n_r) \\
\mathrm{ap}(n_r) &\leftarrow \mathrm{ok}(n_r), \lceil body(r) \rceil, \lceil body^-(\tilde{r}) \rceil \\
\mathrm{bl}(n_r) &\leftarrow \mathrm{ok}(n_r), A^-, \tilde{A}^- \\
\mathrm{bl}(n_r) &\leftarrow \mathrm{ok}(n_r), B, \tilde{B}
\end{aligned}$$

$$\begin{aligned}
\mathrm{ok}(n_r) &\leftarrow \mathrm{ry}(n_r, n_{r_1}), \ldots, \mathrm{ry}(n_r, n_{r_k}) \\
\mathrm{ry}(n_r, n_u) &\leftarrow not\,(n_u < n_r) \\
\mathrm{ry}(n_r, n_u) &\leftarrow (n_u < n_r), \mathrm{ap}(n_u) \\
\mathrm{ry}(n_r, n_u) &\leftarrow (n_u < n_r), \mathrm{bl}(n_u) \\
\mathrm{ry}(n_r, n_u) &\leftarrow \mathrm{ko}(n_u) \\
\mathrm{ry}(n_r, n_u) &\leftarrow \mathrm{reject}(n_u)
\end{aligned}$$

$$false \leftarrow not\,\mathrm{ok}(n_r), not\,\mathrm{reject}(n_r)$$

$$\mathrm{ko}(n_r) \leftarrow \lceil head(r) \rceil, B$$

(DA) Default Atom Rules:

$$\tilde{A}^- \leftarrow not\,\tilde{A} \tag{13}$$

for all objective atoms $A \in \mathcal{K}$.

(SPO) Strict Partial Order Constraints:

$$false \leftarrow n_{r_1} < n_{r_1} \tag{14}$$

$$false \leftarrow n_{r_1} < n_{r_2}, n_{r_2} < n_{r_3}, (n_{r_1} < n_{r_3})^- \tag{15}$$

for every generalized rule $\{r_1, r_2, r_3\} \subseteq \mathcal{P}$.

The next result proves the correctness of the $\Gamma(s, P)$ transformation.

Theorem 2 (Soundness and Completeness). *Let $s \in S$ be a state. Let P be a dynamic prioritized program over the language $\mathcal{L}_{\overline{\mathcal{K}}}$. An interpretation M of P is a (regular) stable model of $\Gamma(s, P)$ iff M, restricted to $\mathcal{L}_{\overline{\mathcal{K}}}$, is a preferred stable model of P at state s.*

Next we summarize the technical properties of the $\tau(r)$ translation.

Proposition 2. *Let $s \in S$ be a state. Let $P = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program over the language $\mathcal{L}_{\overline{\mathcal{K}}}$ and M a (regular) stable model of $\Gamma(s, P)$. Let $\mathcal{P} = \bigcup_{i \leq s} P_i$. Then the following properties hold:*

1. $\forall r \in \mathcal{P}$ if $\mathrm{reject}(n_r) \notin M$, then $\mathrm{ok}(n_r) \in M$.
2. $\forall r \in \mathcal{P}$ if $\mathrm{reject}(n_r) \notin M$, then $[\ \mathrm{ap}(n_r) \in M$ iff $\mathrm{bl}(n_r) \notin M\ ]$.
3. $\forall r \in \mathcal{P}$ $\mathrm{ko}(n_r) \in M$ iff $r \in \mathit{Unsup}(\mathcal{P}, M)$.
4. $\forall r \in \mathcal{P}$ if $\mathrm{reject}(n_r) \notin M$, then $[\ \mathrm{ko}(n_r)$ implies $\mathrm{bl}(n_r)\ ]$

7 Conclusions and Future Work

The technical motivation for our work consisted in obtaining a correct compilation into normal programs of logic programs subjected to updates and preferences combined under a stable models semantics. The transformation, achieved by a preprocessor, can be submitted to a stable models implementation to obtain the solutions, and we have done that too[2]. We started from three previous results, under a stable models semantics: (1) a semantical characterization of general logic programs updates plus preferences combined, including the updating of preferences [3] (2) a transformation into normal programs (and its implementation) of sequences of general logic program updates [2] (3) a transformation into normal programs of logic programs with preferences [9]. We then took (1) and extended (2) on the basis of a generalization of (3).

Importantly, the preference part of our transformation is modular or incremental with respect to the update part of the transformation, in the sense that successive updates only require incremental additions to the preference part of the transformation. Moreover, the size of the transformed program $\Gamma(s, P)$ in the worst case is quadratic on the size of the original dynamic prioritized program P. Ongoing work addresses the issue of garbage collection in DLP programs and extensions thereof. Namely, one wants to know under what conditions update rules can be ignored or permanently deleted, or how the transformed program may be compacted without loss of information for a class of queries. Related to this is the topic of making the most of tabling techniques to avoid repeated recomputations via the inertial rule.

The applications motivation of this technical work stems from some important application areas: e-commerce rule updating and preferring, including the updating of preferences; abductive reasoning with updatable preferences on the

[2] An implementation of updates plus reference reasoning is available at: http://centria.fct.unl.pt/~jja/updates.

abducibles; updatable preferring among actions in planning; dynamically reconfigurable web-sites which adapt to updatable user profiles; reasoning under non-monotonic uncertainty, where a solution must be sought and preferred among changing options.

Future work, apart from exploring some of the above mentioned application areas (e.g., the web-site one in project FLUX), will involve theoretical and technical solutions towards combining preferences and updates within the purview of well-founded semantics, namely by preferring among its partial stable models.

References

1. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. Technical report, LiTH-ITN-R-2002-7, Dept. of Science and Technology, Linköping University, Sweden, 2002. Available at: http://www.itn.liu.se/english/research/.
2. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The J. of Logic Programming*, 45(1-3):43–70, 2000.
3. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzman, G. Brewka, and L. M. Pereira (eds.) *Logics in AI, Procs. JELIA'00*, LNAI 1919, pp. 345–360, Berlin, 2000. Springer.
4. J. J. Alferes and L. M. Pereira. Logic Programming Updating - a guided approach - Essays in honour of Robert Kowalski. In A. Kakas and F. Sadri (eds.) *Computational Logic: From Logic Programming into the Future*. Springer, 2002. To appear. Available at: http://centria.di.fct.unl.pt/∼lmp/.
5. G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4, 1996.
6. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109, 1999.
7. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
8. B. Cui and T. Swift. Preference logic grammars: Fixed-point semantics and application to data standardization. *Artificial Intelligence*, 2002. To appear.
9. J. Delgrande, T. Schaub, and H. Tompits. A compilation of Brewka and Eiter's approach to prioritization. In M. O. Aciego, I. P. de Guzman, G. Brewka, and L. M. Pereira (eds.), *Logics in AI, Procs. JELIA'00*, LNAI 1919, pp. 376–390, 2000.
10. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen (eds.) *Proceedings of the Fifth International Conference on Logic Programming*, pp. 1070–1080, 1988. The MIT Press.
11. B. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Research Report RC 20836, IBM, 1997.
12. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.
13. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In B. Nebel, C. Rich, and W. Swartout (eds.) *KR'92*. Morgan-Kaufmann, 1992.
14. H. Prakken. *Logical Tools for Modelling Legal Argument: A Study of Defeasible Reasoning in Law*, volume 32 of *Law and Philosophy Library*. Kluwer academic publishers, 1997.