User Preference Information in Query Answering*

Pierangelo Dell'Acqua[†*], Luís Moniz Pereira[*], and Aida Vitória[†]

[†] Department of Science and Technology - ITN
Linköping University, 601 74 Norrköping, Sweden
{pier,aidvi}@itn.liu.se

[*] Centro de Inteligência Artificial - CENTRIA
Departamento de Informática, Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Abstract. This paper discusses the use of usage information to enhance cooperative behaviour from query answering systems. A user can pose a query to the system by providing (at query time) update and preference information. Updates allow us to model dynamically evolving worlds and preferences allow us to facilitate the retrieval of information by targeting the answers of the system with respect to users' interests in a given context.

1 Introduction

In the field of cooperative answering, Minker [9] surveys foundational work done toward developing query answering systems with the ability to exhibit cooperative behaviour. He argues that query answering systems are often difficult to use because they do not attempt to cooperate with their users, and he advocates among other things the use of additional information about the users, e.g., their goals, interests and preferences.

In query answering systems, queries have typically several answers and selecting the desired ones is often hard for users. In many applications it is desirable to enhance the system with preference information that can be employed to derive the intended conclusions. Consider a system whose knowledge is defined as a pair (P, R), where P is a set of rules formalizing the background knowledge of the system and R expresses preference information over the rules in P, meaning that when rules conflict and exclude one another, then some are preferred over others according to a partial order. Now the system can use the knowledge in P to derive conclusions and the preferences in R to derive the preferred ones.

The system will exhibit an extra level of flexibility if further it permits the user to provide a temporary preference information update at query time, contextual to some goal. A query can be defined as (G, Pref), where the idea is to

* Pereira acknowledges the support of POCTI project 40958 "FLUX - FleXible Logical Updates" and Dell'Acqua a Research Grant by the *Stiftelsen Lars Hiertas Minne*.

prove the goal G by taking into account the preferences in *Pref*. Now, if the knowledge of the system is (P, R), then the system has to derive the conclusions that satisfy the preferences in R (i.e., the preferences applying to every query) but which are updated by the contextual preferences in *Pref* (i.e, the preferences specific to that particular query). Finally, it is desirable to make the knowledge (P, R) of the system updatable in a way that it can be modified to reflect changes in the world, including the general preferences.

In this paper, we elaborate on the usage of user information in cooperative answering, as advocated by Minker. We present a novel framework that allows the user to specify his/her preferences and context information. Both user preferences and context information are used for building an answer. Context information provided by a query is taken into account through the mechanism of updates. Updates are used to model dynamically evolving worlds and preferences to enhance the retrieval of information. 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 changes in the world contradict 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 any piece of knowledge that changed, and to replace it with the newer one.

Preference reasoning is a pervasive form of reasoning. Preference information is used along with incomplete knowledge. In such a setting, due to the incompleteness of the knowledge, several semantic models may be possible. Preference reasoning acts by choosing among those possible models. An example is the "vacation" problem where the incomplete knowledge contains the rules: "go to the beach if not go to the mountain" (1) and "go to the mountain if not go to the beach" (2). Then, two models are possible: one model obtained by preferring the first rule, and the other obtained by preferring the second rule. Thus, preference information among the rules can be used to select some of the models on the basis of any chosen criteria expressed by the user (e.g., utility, plausibility, etc.). Cf. [4, 5] for additional motivation, comparisons, applications, and references. To the best of our knowledge the approach in [1, 3] is the only approach proposed so far that combines update and preference reasoning, the updating of preferences included.

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 over the language $\mathcal{L}_{\mathcal{K}}$ is a set of ground generalized rules of the form $L_0 \leftarrow L_1, \ldots, L_n$ ($n \geq 0$), 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). Whenever a literal L is of the form *not* A, *not* L stands for the objective atom A.

The semantics of generalized logic programs [7,8] is defined as a generalization of the stable model semantics [6]. Here we use the definition that appeared in [2] (proven there equivalent to [7,8]). In the remainder of the paper, by (2-valued) *interpretation* M of $\mathcal{L}_{\mathcal{K}}$ we mean any set of propositional variables from $\mathcal{L}_{\mathcal{K}}$ such that for any A in $\mathcal{L}_{\mathcal{K}}$ precisely one A or *not* A belongs to M.

Definition 2 (Default assumptions). *Let P be a set of generalized rules and M 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 over the language $\mathcal{L}_{\mathcal{K}}$. Given a rule r, we write $n_r \in N$ to indicate its name. Let $<$ be a binary predicate symbol[1] whose arguments are names in 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 $Z \leftarrow L_1, \ldots, L_n$ with $n \geq 0$, where Z is a literal of the form $n_r{<}n_u$ or its default negation (not $n_r{<}n_u$) and each L_i is a literal over $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$.*

Note that the set N does not include $<$, 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 $\Sigma = (P, R)$ 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.

Definition 6 (Goal). *Let $L_1, \ldots, L_n$ be literals over the language $\mathcal{L}_{\bar{\bar{\mathcal{K}}}}$. Then $?{-}L_1, \ldots, L_n$ is a goal.*

[1] In order to establish the preferred stable models (cf. Def. 12), 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 newly acquired knowledge) can be added at the end of the sequence without bothering 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.

To allow the updating of 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 sets of priority rules 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. In the remainder, let $S = \{1, \ldots, s, \ldots\}$ be a set of natural numbers. We call the elements $i \in S$ *states*.

Definition 7 (Dynamic Prioritized Program). *Let S be a set of states. For every state $i \in S$, let P_i be the generalized logic program over the language $\mathcal{L}_{\mathcal{K}}$, and R_i the set of priority rules over the language $\mathcal{L}_{\overset{=}{\mathcal{K}}}$. Then, $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ is a dynamic prioritized program over the language $\mathcal{L}_{\overset{=}{\mathcal{K}}}$.*

The semantics of a dynamic prioritized program 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 8 (Rejected rules). *Let $s \in S$ be a state. Let $\Gamma = \bigoplus\{(P_i \cup R_i) : i \in S\}$ be a dynamic prioritized program and M an interpretation of Γ. Then*

$$Reject(s, M, \Gamma) = \\ \{r \in T_i \mid \exists r' \in T_j, head(r) = not\,head(r'), i < j \leq s \text{ and } M \models body(r')\}$$

where $T_i = P_i \cup R_i$, for every $i \in S$.

The stable models of a dynamic prioritized program Γ 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 Γ. Consequently, the preference information expressed by the priority rules in the R_is is not yet taken into account here. (In Section 4 we define the notion of preferred stable models that takes priority rules into consideration.)

Definition 9 (Stable Models of a DPP at state s). *Let $s \in S$ be a state. Let $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program and M an interpretation of Γ. M is a stable model of Γ at state s iff*

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

where $\mathcal{PR} = \bigcup_{i \leq s}(P_i \cup R_i)$.

The next example illustrates the stable models of a dynamic prioritized program. Note that the preference information expressed in the R_is is not used to select the stable models.

Example 1. Let $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ be the dynamic prioritized program:

$$P_1 = \left\{ \begin{array}{ll} a \leftarrow not\, b & (r_1) \\ b \leftarrow not\, a & (r_2) \end{array} \right\} \qquad R_1 = \{\, n_1 < n_2 \,\}$$

$$P_2 = \{\, c \,\} \qquad\qquad\qquad R_2 = \{\, not\,(n_1 < n_2) \,\}$$

The stable models of Γ at state 1 are $M_1 = \{a, n_1 < n_2\}$ and $M_2 = \{b, n_1 < n_2\}$, and at state 2 are $M_3 = \{a, c\}$ and $M_4 = \{b, c\}$.

4 Preferred Stable Models

We present the notion of preferred stable model (for a more detailed presentation see [3]). Intuitively, the priority information specified by the R_is in a dynamic prioritized program Γ is used to prefer among the stable models of Γ. This is achieved by first deleting (from Γ) 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 10 (Unsupported rules). *Let P be a set of generalized rules and M an interpretation of P. Then*

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

Definition 11 (Unpreferred generalized rule). *Let P be a set of generalized rules and M an interpretation of P. $Unpref(P, M)$ is a set of unpreferred generalized rules of P and M iff*

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

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

As the set N of constants of $<$ does not include $<$ itself, it is not possible to express preferences on priority rules themselves. Therefore, no priority rule r in P can be unpreferred, that is, $r \notin \mathit{Unpref}(P, M)$, for any set P of generalized rules and any interpretation M of P.

Definition 12 (Preferred Stable Models at state s). *Let $s \in S$ be a state. Let $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program and M a stable model of Γ at state s. Then M is a preferred stable model of Γ 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* $(r_1 < r_3) \in M$
- $M = \mathit{least}(\,[\,\mathcal{X} - \mathit{Unpref}(\mathcal{X}, M)\,] \cup \mathit{Default}(\mathcal{PR}, M)\,)$

where $\mathcal{PR} = \bigcup_{i \le s}(P_i \cup R_i)$ *and* $\mathcal{X} = \mathcal{PR} - \mathit{Reject}(s, M, \Gamma)$.

Notice that the definition above requires M to be a stable model of Γ in order for M to be a preferred stable model of Γ. This reflects the requirement that preference information is intended to select among alternative stable models of Γ. 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 Γ, then Γ will admit at most one preferred stable model.

Example 2. (Contextual preferences) This example illustrates the use of contextual preferences to select preferred models. For simplicity, we employ only one state. Consider a scenario where John wants to buy a magazine. He can buy either a sport magazine (*sm*), a travel magazine (*tm*) or a financial magazine (*fm*). Suppose that John's preferences are context sensitive, i.e., he may have distinct preferences depending on time, location, etc. This example can be formalized via $\Gamma = \bigoplus\{(P_1, R_1)\}$, where:

$$P_1 = \left\{ \begin{array}{ll} \mathit{sm} \leftarrow \mathit{not\,fm}, \mathit{not\,tm} & (r_1) \\ \mathit{tm} \leftarrow \mathit{not\,fm}, \mathit{not\,sm} & (r_2) \\ \mathit{fm} \leftarrow \mathit{not\,sm}, \mathit{not\,tm} & (r_3) \end{array} \right\} \qquad R_1 = \left\{ \begin{array}{l} n_1 < n_3 \leftarrow \mathit{holiday} \\ n_2 < n_3 \leftarrow \mathit{holiday} \\ n_3 < n_1 \leftarrow \mathit{office} \\ n_3 < n_2 \leftarrow \mathit{office} \end{array} \right\}.$$

Since the priorities in R_1 cannot be used (both *holiday* and *office* being false), the preferred stable models of Γ coincide with its stable models (i.e., Γ has three models containing *sm*, *tm* and *fm*, respectively). If *holiday* would hold (for example via a subsequent update), then Γ would have two preferred stable models containing *sm* and *tm*, respectively.

5 Preferred Conclusions

The work briefly reviewed in the previous sections, proposes an approach for defining and reason with theories that capture preference information. Moreover, these theories are intended to model a dynamic world. Hence, the notion of update is also incorporated in the theory. In this section we address the problem of querying a dynamic prioritized program. A novel aspect is that queries can be equipped with the user's preferences and context information. That is, both context and preference information can be provided at query time. The ability to take into account the user information during the process of answering a query enhances the cooperative behaviour of the system. In fact, the system is now able to target its answers to the user's goal and interests.

Definition 13 (Query with preferences). *Let $s \in S$ be a state, G a (ground) goal, Σ a prioritized logic program, and $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ a dynamic prioritized program. Then $Q = (G, \Sigma)$ is a query w.r.t Γ.*

The aim of the prioritized logic program $\Sigma = (P, R)$ in a query $Q = (G, \Sigma)$ is to allow the user's knowledge or context information P to be taken into account when answering the query and also to allow the system to consider the user's preferences R[2]. Note that user knowledge evolution[3] could be easily captured by allowing Σ to be a dynamic prioritized program (instead of a prioritized logic program) in queries. Our framework could be easily extended to deal with this case. In the remainder, we assume that $S^+ = S \cup \{\max(S) + 1\}$.

Definition 14 (Joinability at state s). *Let $s \in S^+$ be a state. Given a dynamic prioritized program $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ and a prioritized logic program $\Sigma = (P_q, R_q)$, the joinability function at state s, written as $\diamond_s$, is defined as*

$$\Gamma \diamond_s \Sigma = \bigoplus\{(P'_i, R'_i) : i \in S^+\}$$

where

$$(P'_i, R'_i) = \begin{cases} (P_i, R_i) & \text{if } 1 \leq i < s, \\ (P_Q, R_Q) & \text{if } i = s, \\ (P_{i-1}, R_{i-1}) & \text{if } s < i \leq max(S^+). \end{cases}$$

From the definition above, the reader can see that joinability is parameterized by a state s. Its usefulness can be illustrated by the following examples. If we want to give priority to the user preferences in Σ over the preferences in Γ, then the joinability function $\diamond_{\max(S^+)}$ must be used. Several applications require this type of joinability, e.g., a web-site application of a travel agency whose database (formalized as a dynamic prioritized program) maintains information about holiday resorts and preferences among touristy locations, kinds of holidays, etc. When a user asks a query Q, the system has to give priority to the user interests and preferences expressed in Q.

On the other hand, other applications require, for instance, the joinability function $\diamond_1$. As an example, consider a program automating the procedure of allocating holiday periods to employees of a company. The database stores information about holiday periods already allocated and preferences among alternative possibilities over periods of the year, e.g., it is preferable to have at

[2] Flexible preferences in FQAS pose the problem of guaranteeing the respect of dynamic preferences for partial order postulates (antisymmetry, anti-reflexivity, transitivity). For reasons of space, this problem is not discussed here.

[3] For example, this ability allows us to model user evolving profiles (see [10] for a Web application).

least three weeks of holidays in August; in any other month of the year, it is preferable that not too many employees of the same division take holiday at the same time, etc. When a user asks a query, he provides preference information about the most suitable holiday periods for him or her, and the system must take them into account, but now the company preferences (expressed in the program) have more priority than the employee preferences.

Definition 15 (Preferred conclusions). *Let $s \in S^+$ be a state and $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program. The preferred skeptical conclusions of Γ with joinability function $\diamond_s$ are*

$$\{Q \ : Q \textit{ is a query } (G, \Sigma) \textit{ and } G \textit{ is included in every preferred stable model of } \Gamma \diamond_s \Sigma \textit{ at state } max(S^+)\}\,.$$

The preferred credulous conclusions of Γ with joinability function $\diamond_s$ are

$$\{Q \ : Q \textit{ is a query } (G, \Sigma) \textit{ and } G \textit{ is included in a preferred stable model of } \Gamma \diamond_s \Sigma \textit{ at state } max(S^+)\}\,.$$

Example 3. Let Γ be the dynamic prioritized program $\bigoplus\{(P_1, R_1)\}$ of Example 2, and $\diamond_2$ a joinability function. Then
$Q_1 = (?{-}fm, (\{office\}, \{\}))$ is a skeptical preferred conclusion,
$Q_2 = (?{-}sm, (\{holiday\}, \{\}))$ is a credulous preferred conclusion, and
$Q_3 = (?{-}sm, (\{holiday, footballMatch\}, \{n_1 < n_2 \leftarrow footballMatch\}))$ is a skeptical preferred conclusion.

Definition 16 (Answer). *Let $s \in S^+$ be a state and $\Gamma = \bigoplus\{(P_i, R_i) : i \in S\}$ be a dynamic prioritized program. The skeptical (credulous) answer to the query $Q = (G, \Sigma)$ w.r.t Γ is* `yes` *if Q is a preferred skeptical (credulous) conclusion of Γ with joinability function $\diamond_s$. Otherwise, the answer is* `no`.

We have considered ground queries. But the notion of answer can be easily extended to non ground queries. As usual, an answer to a non ground query $Q = (G, \Sigma)$ (i.e. G is non ground) is the set of ground instances $\sigma(G)$ of G (σ is a substitution) such that the answer to $(\sigma(G), \Sigma)$ is `yes`.

Notice that every query $Q = (G, \Sigma)$ w.r.t a program Γ is evaluated in the most recent state of $\Gamma \diamond_s \Sigma$ (i.e., in the state $\max(S^+)$). However, in some applications it may be needed to evaluate queries in previous states (i.e., in the past). This capability could be introduced in our framework as well.

6 Example: Car Dealer

Consider the following (sketched) program that exemplifies the process of quoting prices for second-hand cars. For simplicity, the price of a car can be either 200 or 250 depending on how long the car has been in stock and on the colour (either orange or black) of the car. The policy of the company is to sell a car cheaper if the car has been in stock for a period equal or longer than 10 months, unless the customer likes the colour of the car. In such a case the company tries to sell the car with the (higher) price 250. Below we write non ground rules, and with that we intend their ground instances.

$$price(Car, 200) \leftarrow stock(Car, Colour, T), not\ price(Car, 250), not\ offer \qquad (r1)$$
$$price(Car, 250) \leftarrow stock(Car, Colour, T), not\ price(Car, 200), not\ offer \qquad (r2)$$
$$prefer(orange) \leftarrow not\ prefer(black) \qquad (r3)$$
$$prefer(black) \leftarrow not\ prefer(orange) \qquad (r4)$$
$$stock(Car, Colour, T) \leftarrow bought(Car, Colour, Date), T = today - Date \qquad (r5)$$

When the company buys a car, the company must add (via an update) the car to the stock together with its colour and the date of purchase. For instance,

$$bought(fiat, orange, d1) \leftarrow$$

On the contrary, any time the company sells a car, the company must remove all the information about that car from the stock. For instance,

$$not\ bought(volvo, black, d2) \leftarrow$$

The following priority rules formalize the selling strategy of the company. By default, we assume that the customers prefer the colour black to orange.

$$\begin{aligned}
n_2 < n_1 &\leftarrow stock(Car, Colour, T), T < 10\\
n_1 < n_2 &\leftarrow stock(Car, Colour, T), T \geq 10, not\ prefer(Colour)\\
n_2 < n_1 &\leftarrow stock(Car, Colour, T), T \geq 10, prefer(Colour)\\
not\ (n_3 < n_4) &\leftarrow\\
n_4 < n_3 &\leftarrow
\end{aligned}$$

Suppose that later on the company adopts the policy to offer a special price for cars at certain times of the year. This is achieved via the update[4]

$$price(Car, 100) \leftarrow stock(Car, Colour, T), offer \qquad (r6)$$
$$not\ offer \leftarrow$$

As soon as *offer* holds (via a subsequent update), the rule (r6) will be in force by making the price of every car to be 100. Suppose that the company has in stock a fiat bought at date *d1* whose colour is orange, and that *offer* does not hold at the current state. Now, independently of the joinability function used, the answer to the query $Q1 = (?{-}price(fiat, P), (\{\}, \{\}))$ is $P = 250$ in case $today - d1 < 10$, or $P = 200$ in case $today - d1 \geq 10$. Things change if a customer asks the query $Q2 = (?{-}price(fiat, P), (\{\}, \{not\,(n_4 < n_3), n_3 < n_4\}))$, stating that he prefers the colour orange. The intended answer is $P = 250$ irrespectively of how long the car has been in stock. Note that for this query it is relevant which joinability function is used. In fact, if we use $\diamond_1$, then we do not get the intended answer because the user preferences (regarding the preferred colour) would be overwritten by the default preferences of the company. On the other hand, it would not be so appropriate (from the company's perspective) to use $\diamond_{\max(S^+)}$. Indeed, a customer could ask $Q3 = (?{-}price(fiat, P), (\{offer\}, \{\}))$.

[4] Note that we can express more complex rules for pricing cars depending for example on whether or not the customer wants to sell his old car. In such a case, the customer must specify (in P_Q) at query time the kind of car he wants to sell.

7 Conclusions and Future Work

We have presented a novel logical framework where update and preference information can be specified and used in query answering systems. We have shown through some examples how the proposed approach can enhance the cooperative behaviour of the system. The procedural semantics of the approach is based on a syntactic transformation presented in [1].[5]

The approach can be extended in several ways. For example, the user knowledge evolution could be easily captured by allowing Σ (in queries) to be a dynamic prioritized program. Another extension would be to equip queries with a state to allow the users to query the past. An interesting line of research is to investigate (1) how to incorporate abduction into our framework: we may have abductive preferences leading to conditional answers depending on accepting a preference; and (2) how to tackle the problem arising when we have several users query the system together (i.e., when we need to take into account all their preferences). From a practical point of view, we may fornish the user with a preference metalanguage that compiles into the tedious pairwise preference specification. Additionally, we can detect inconsistent preference specifications and cater for interactive removal of such specifications.

References

1. J. J. Alferes, P. Dell'Acqua, and L. M. Pereira. A compilation of updates plus preferences. To appear in: *Logic in Artificial Intelligence* (Jelia'02), LNAI, 2002.
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. A short version titled *Dynamic Logic Programming* appeared in A. Cohn and L. Schubert (eds.), *KR'98*, Morgan Kaufmann.
3. J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego et al. (eds.), *Logics in AI, Procs. JELIA'00*, LNAI 1919, pp. 345–360, Berlin, 2000.
4. G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4, 1996.
5. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109, 1999.
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen (eds.), *Proc. of the Fifth Int. Conf. on Logic Programming*, pp. 1070–1080, Cambridge, MA, 1988. The MIT Press.
7. K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.
8. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In B. Nebel et al. (eds.), *KR'92*. Morgan-Kaufmann, 1992.
9. J. Minker. An overview of cooperative answering in databases. (Invited paper.) Int. Conf. on Flexible Query Answering Systems (FQAS), 1998.
10. J. Pagonis and M. C. Sinclair. Evolving user profiles to reduce internet information overload. In R. John and R. Birkenhead (eds.), *Int. Conf. on Recent Advances in Soft Computing*, Advances in Soft Computing 232, pp. 100–107. Springer, 2000.

[5] A prototype implementation of updates plus preferences implemented by J. J. Alferes is available at: http://centria.di.fct.unl.pt/~jja/updates.