A Compilation of Brewka and Eiter's Approach to Prioritization

James P. Delgrande[1], Torsten Schaub[2⋆], and Hans Tompits[3]

[1] School of Computing Science, Simon Fraser University,
Burnaby, B.C., Canada V5A 1S6, jim@cs.sfu.ca
[2] Institut für Informatik, Universität Potsdam,
Postfach 601553, D–14415 Potsdam, Germany, torsten@cs.uni-potsdam.de
[3] Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Wien, Austria, tompits@kr.tuwien.ac.at

Abstract. In previous work, we developed a framework for expressing general preference information in default logic and logic programming. Here we show that the approach of Brewka and Eiter can be captured within this framework. Hence, the present results demonstrate that our framework is general enough to capture other independently-developed methodologies. As well, since the extended logic program framework has been implemented, we provide an implementation of the Brewka and Eiter approach via an encoding of their approach.

1 Introduction

In previous work [6], we presented a general framework based on default logic for expressing general preference information. There, we addressed the problem of representing preferences among individual and aggregated properties in default logic. In this approach, one begins with an ordered default theory, in which preferences are specified on default rules. This is transformed into a second, standard, default theory in which the preferences are respected, in the sense that the obtained default extensions contain just those conclusions that accord with the order expressed by the original preference information. The approach is fully general: One may specify preferences that hold by default, or give preferences among preferences, or give preferences among sets of defaults.

We adapted this approach in [8] for logic programming under the answer set semantics [10]. While the original approach is usable for full-fledged theorem provers for default logic, like `DeReS` [5], this subsequent approach applies to logic programming systems, such as `dlv` [9] or `smodels` [13]. In fact, we have provided an implementation of the approach in extended logic programs, serving as a front-end for `dlv` and `smodels` (see [7] for details).

In the context of default logic, our methodology involves the appropriate "decomposition" of default rules, so that one can detect the applicability conditions of default rules and control their actual application. In our framework, this is carried out *within* a default theory. This is accomplished, first, by associating a unique name with each

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Canada.

default rule, so that it can be referred to within a theory. Second, special-purpose predicates are introduced for detecting conditions in a default rule, and for controlling rule invocation. This in turn allows a fine-grained control over what default rules are applied and in what cases. By means of these named rules and special-purpose predicates, one can formalise various phenomena of interest.

Given an ordered default theory $(D, W, <)$, where $<$ is a strict partial order on D, the intuition is that one applies the $<$-maximal default(s), if possible, then the next $<$-greatest, and so on. Thus we adopt a *prescriptive* interpretation of the ordering, in that $<$ prescribes the order in which rules are applied. This can be contrasted with a *descriptive* interpretation, in which the preference order represents a ranking on desired outcomes: the desirable (or: preferred) situation is one where the most preferred default(s) are applied.

The approach of Brewka and Eiter [3], first developed with respect to extended logic programs and subsequently generalized for default logic in [4], arguably fits the "descriptive" interpretation. In common with previous work, Brewka and Eiter begin with a partial order on a rule base, but define preference with respect to total orders that conform to the original partial order. As well, answer sets or extensions, respectively, are first generated and the "prioritized" answer sets (extensions) are selected subsequently. In contrast, in our approach, we deal only with the original partial order, which is translated into the object theory. As well, only "preferred" extensions are produced in our approach; there is no need for meta-level filtering of extensions.

However, we show here that the approach of Brewka and Eiter is expressible in our framework. Consequently, this serves to show the scope and generality of our framework. As well, this result enables a straightforward implementation of the Brewka and Eiter approach.

In the next subsection we briefly introduce default logic, while Sections 3 and 4 introduce our approach and Brewka and Eiter's, respectively. Section 5 describes the translation of their approach expressed in default logic, while Section 6 does the same for the case of extended logic programs. Section 7 gives brief concluding remarks.

2 Background

Default logic [15] augments classical logic by *default rules* of the form

$$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$$

where α, β_1, ..., β_n, γ are sentences of first-order or propositional logic. Here we mainly deal with *singular* defaults for which $n = 1$. A singular rule is *normal* if β is equivalent to γ; it is *semi-normal* if β implies γ. [11] shows that any default rule can be transformed into a set of semi-normal defaults. We sometimes denote the *prerequisite* α of a default δ by *Prereq*(δ), its *justification* β by *Justif*(δ), and its *consequent* γ by *Conseq*(δ). Accordingly, *Prereq*(D) is the set of prerequisites of all default rules in D; *Justif*(D) and *Conseq*(D) are defined analogously. Empty components, such as no prerequisite or even no justifications, are assumed to be tautological (we speak in such cases of *prerequisite-free* and *justification-free* defaults, respectively). *Open defaults*

with unbound variables are taken to stand for all corresponding instances. A set of default rules D and a set of sentences W form a *default theory* (D, W) that may induce a single, multiple, or even zero *extensions* in the following way:

Definition 1. *Let (D, W) be a default theory and let E be a set of sentences. Define $E_0 = W$ and for $i \geq 0$:*

$$GD_i = \left\{ \frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D \;\middle|\; \alpha \in E_i, \neg\beta_1 \notin E, \ldots, \neg\beta_n \notin E \right\};$$
$$E_{i+1} = Th(E_i) \cup \{\mathit{Conseq}(\delta) \mid \delta \in GD_i\}.$$

Then, E is an extension for (D, W) iff $E = \bigcup_{i=0}^{\infty} E_i$.

($Th(E)$ refers to the logical closure of set E of sentences.) Any such extension represents a possible set of beliefs about the world at hand. The above procedure is not constructive since E appears in the specification of GD_i. We define $GD(D, E) = \bigcup_{i=0}^{\infty} GD_i$ as the set of *generating defaults* of extension E. An enumeration $\langle \delta_i \rangle_{i \in I}$ of default rules is *grounded* in a set of sentences W, if we have for every $i \in I$ that $W \cup \mathit{Conseq}(\{\delta_0, \ldots, \delta_{i-1}\}) \vdash \mathit{Prereq}(\delta_i)$.

For simplicity, we restrict our attention in what follows to finite, singular default theories, consisting of finite sets of default rules and sentences.

3 Preference-Handling in Standard Default Logic

For adding preferences among default rules, a default theory is usually extended with an ordering on the set of default rules. In accord with [4], we define:

Definition 2. *A prioritized default theory is a triple $(D, W, <)$ where (D, W) is a default theory and $<$ is a strict partial order on D.*

In contrast to [4], however, we use the ordering $<$ in the sense of "higher priority", i.e., $\delta < \delta'$ expresses that δ' has "higher priority" than δ.

The methodology of [6] provides a translation, $\mathcal{T}$, that takes such a prioritized theory $(D, W, <)$ and translates it into a regular default theory $\mathcal{T}((D, W, <)) = (D', W')$ such that the explicit preferences in $<$ are "compiled" into D' and W' and such that the extensions of (D', W') correspond to the "preferred" extensions of $(D, W, <)$. Moreover, the approach admits not only "static" preferences as discussed here—where the ordering of the defaults is specified at the meta-level—but also "dynamic" preferences *within the object language*.

In [6], to begin with, a unique name is associated with each default rule. This is done by extending the original language by a set of constants[1] N such that there is a bijective mapping $n : D \to N$. We write n_δ instead of $n(\delta)$ (and abbreviate n_{δ_i} by n_i to ease notation). Also, for default rule δ with name n, we sometimes write $n : \delta$ to render naming explicit. To encode the fact that we deal with a finite set of distinct default

[1] McCarthy effectively first suggested the naming of defaults using a set of *aspect* functions [12]; Theorist [14] uses atomic propositions to name defaults.

rules, we adopt a unique names assumption (UNA_N) and domain closure assumption (DCA_N) with respect to N. That is, for a name set $N = \{n_1, \ldots, n_m\}$, we add axioms

$$\begin{array}{ll} \text{UNA}_N: & (n_i \neq n_j) \text{ for all } n_i, n_j \in N \text{ with } i \neq j; \\ \text{DCA}_N: & \forall x.\, name(x) \equiv (x = n_1 \vee \cdots \vee x = n_m). \end{array}$$

For convenience, we write $\forall x \in N.\, P(x)$ instead of $\forall x.\, name(x) \supset P(x)$.

Given $\delta_i < \delta_j$, we want to ensure that, before δ_i is applied, δ_j can be applied or found to be inapplicable.

More formally, we wish to exclude the case where $\delta_i \in GD_n$ but $\delta_j \notin GD_n$ although $\delta_j \in GD_m$ for some $m > n$ in Definition 1. For this purpose, we need to be able to (i) detect when a rule has been applied or when a rule is blocked, and (ii) control the application of a rule based on other antecedent conditions. For a default rule $\frac{\alpha : \beta}{\gamma}$ there are two cases for it to not be applied: it may be that the antecedent is not known to be true (and so its negation is consistent), or it may be that the justification is not consistent (and so its negation is known to be true). For detecting this case, we introduce a new, special-purpose predicate $\mathsf{bl}(\cdot)$. Similarly we introduce a predicate $\mathsf{ap}(\cdot)$ to detect when a rule has been applied. To control application of a rule we introduce predicate $\mathsf{ok}(\cdot)$. Then, a default rule $\delta = \frac{\alpha : \beta}{\gamma}$ is mapped to

$$\frac{\alpha \wedge \mathsf{ok}(n_\delta) : \beta}{\gamma \wedge \mathsf{ap}(n_\delta)}, \quad \frac{\mathsf{ok}(n_\delta) : \neg\alpha}{\mathsf{bl}(n_\delta)}, \quad \frac{\neg\beta \wedge \mathsf{ok}(n_\delta) :}{\mathsf{bl}(n_\delta)}\,. \tag{1}$$

These rules are sometimes abbreviated by $\delta_a, \delta_{b_1}, \delta_{b_2}$, respectively. While δ_a is more or less the image of the original rule δ, rules δ_{b_1} and δ_{b_2} capture the non-applicability of the rule.

None of the three rules in the translation can be applied unless $\mathsf{ok}(n_\delta)$ is true. Since $\mathsf{ok}(\cdot)$ is a new predicate symbol, it can be expressly made true in order to potentially enable the application of the three rules in the image of the translation. If $\mathsf{ok}(n_\delta)$ is true, the first rule of the translation may potentially be applied. If a rule has been applied, then this is indicated by asserting $\mathsf{ap}(n_\delta)$. The last two rules give conditions under which the original rule is inapplicable: either the negation of the original antecedent α is consistent (with the extension) or the justification β is known to be false; in either such case $\mathsf{bl}(n_\delta)$ is concluded.

We can assert that default $n_j : \frac{\alpha_j : \beta_j}{\gamma_j}$ is preferred to $n_i : \frac{\alpha_i : \beta_i}{\gamma_i}$ in the object language by introducing a new predicate, $\prec$, and then asserting that $n_i \prec n_j$. However, this translation so far does nothing to control the order of rule application. Nonetheless, for $\delta_i < \delta_j$ we can now control the order of rule application: we can assert that if δ_j has been applied (and so $\mathsf{ap}(n_j)$ is true), or known to be inapplicable (and so $\mathsf{bl}(n_j)$ is true), then it is ok to apply δ_i. The idea is thus to *delay* the consideration of less preferred rules until the applicability question has been settled for the higher ranked rules. Formally, this is realized by adding the axiom

$$\forall x \in N. \left[\forall y \in N. (x \prec y) \supset (\mathsf{bl}(y) \vee \mathsf{ap}(y))\right] \supset \mathsf{ok}(x) \tag{2}$$

to the translation.

To summarize, let $\mathcal{T}((D, W, <)) = (\tilde{D}, \tilde{W})$ be the translation obtained in this way, for a given prioritized default theory $(D, W, <)$. Then, the prioritized extensions of $(D, W, <)$ are determined by the (regular) extensions of $(\tilde{D}, \tilde{W})$, modulo the original language.

It is important to note that this translation schema is just one possible preference strategy. Changes to the conditions when a default is considered to be applicable (realized by the specific form of the decomposed defaults $\delta_a, \delta_{b_1}, \delta_{b_2}$ and axiom (2)) result in different preference strategies. Also, further rules and special-purpose predicates can be added, if needed. For instance, in Sections 5 and 6 we rely on an additional predicate $\mathsf{ko}(\cdot)$ that aims at eliminating rules from the reasoning process.

4 Brewka and Eiter's Approach to Preference

We now describe the approach to dealing with a prioritized default theory introduced in [4]. First, partially ordered default theories are reduced to totally ordered ones.[2]

Definition 3. *A fully prioritized default theory is a prioritized default theory $(D, W, <)$ where $<$ is a total ordering.*

The general case of arbitrary prioritized default theories is reduced to this restricted case as follows.

Definition 4. *Let $(D, W, <)$ be a prioritized default theory. Then, E is a prioritized extension of $(D, W, <)$ iff E is a prioritized extension of some fully prioritized default theory $(D, W, <')$ such that $<\subseteq<'$.*

Conclusions of prioritized default theories are defined in terms of prioritized extensions, which are a subset of the regular extensions of a default theory, i.e., the extensions of (D, W) according to [15].

The construction of prioritized extensions relies on the notion of *activeness* [1, 2]. A default δ is *active* in a set of formulas S, if (i) *Prereq*$(\delta) \in S$, (ii) $\neg$*Justif*$(\delta) \notin S$, and (iii) *Conseq*$(\delta) \notin S$ hold. Intuitively, a default is active in S if it is applicable with respect to S but has not yet been applied.

Definition 5. *Let $\Delta = (D, W, <)$ be a fully prioritized prerequisite-free default theory. The operator C is defined as follows: $C(\Delta) = \bigcup_{i \geq 0} E_i$, where $E_0 = Th(W)$, and for every $i > 0$,*

$$E_i = \begin{cases} \bigcup_{j<i} E_j & \text{if no default from } D \text{ is active in } \bigcup_{j<i} E_j; \\ Th(\bigcup_{j<i} E_j \cup \{\textit{Conseq}(\delta)\}) & \text{otherwise, where } \delta \in D \text{ is the maximal} \\ & \text{default (w.r.t. } <) \text{ active in } \bigcup_{j<i} E_j. \end{cases}$$

In the case of prerequisite-free, normal default theories, the operator C always produces an extension in the sense of [15] and thus can directly be used to define prioritized extensions:

[2] In fact, [4] deal with so-called *well-orderings*, which are generalised total orderings, needed for treating infinite domains.

Definition 6. *Let $\Delta = (D, W, <)$ be a fully prioritized prerequisite-free, normal default theory. Then, E is the prioritized extension of Δ iff $E = C(\Delta)$.*

The next definition addresses the more general class of prerequisite-free theories:

Definition 7. *Let $\Delta = (D, W, <)$ be a fully prioritized prerequisite-free default theory. Then, a set E of formulas is a prioritized extension of Δ iff $E = C(\Delta^E)$, where $\Delta^E = (D^E, W, <)$ and $D^E = D \setminus \{\delta \in D \mid$ Conseq$(\delta) \in E$ and $\neg$Justif$(\delta) \in E\}$.*

That is, Δ^E is obtained from Δ by deleting all defaults whose consequents are in E and which are defeated in E. Clearly, this leaves normal rules unaffected. The purpose of this filter is illustrated in [4] by the following default theory:

$$\Delta_3 = \left(\left\{ n_1 : \tfrac{:\neg B}{A}, n_2 : \tfrac{:\neg A}{\neg A}, n_3 : \tfrac{:A}{A}, n_4 : \tfrac{:B}{B} \right\}, \emptyset, \{\delta_j < \delta_i \mid i < j\}\right). \quad (3)$$

This theory has two regular extensions, $Th(\{A, B\})$ and $Th(\{\neg A, B\})$. Applying operator C to Δ_3 yields the first extension. However, it is argued in [4] that this extension does not preserve priorities because default δ_2 is defeated in E by applying a default which is less preferred than δ_2, namely default δ_3. This extension is ruled out by the filter in Definition 7 because $Th(\{A, B\}) \neq Th(\{\neg A, B\}) = C(\Delta_3^{Th(\{A,B\})})$. Theory Δ_3 has therefore no prioritized extension.

The next definition accounts for the general case by reducing it to the prerequisite-free one. For checking whether a given regular extension E is prioritized, Brewka and Eiter evaluate the prerequisites of the default rules according to the extension E. To this end, for a default δ, define $\delta^\top$ as the prerequisite-free version of δ, i.e., $\delta^\top$ results from δ by replacing *Prereq*(δ) by $\top$.

Definition 8. *Let $\Delta = (D, W, <)$ be a fully prioritized default theory and E a set of formulas. The default theory $\Delta_E = (D_E, W, <_E)$ is obtained from Δ as follows:*

1. *$D_E = \{\delta^\top \mid \delta \in D$ and Prereq$(\delta) \in E\}$;*
2. *for any $\zeta_1, \zeta_2 \in D_E$, $\zeta_1 <_E \zeta_2$ iff $\delta_1 < \delta_2$ where $\delta_i = \max_<\{\delta \in D \mid \delta^\top = \zeta_i\}$.*

In other words, D_E is obtained from D by (i) eliminating every default $\delta \in D$ such that *Prereq*$(\delta) \notin E$, and (ii) replacing *Prereq*(δ) by $\top$ in all remaining defaults δ.

Definition 9. *Let $\Delta = (D, W, <)$ be a fully prioritized default theory. Then, E is a prioritized extension of Δ, if (i) E is a classical extension of Δ, and (ii) E is a prioritized extension of Δ_E.*

That is, (*ii*) is equivalent to $E = C((\Delta_E)^E)$.

For illustration, consider [4, Example 4]:

$$\tfrac{:A}{A} < \tfrac{:\neg B}{\neg B} < \tfrac{A:B}{B}, \quad (4)$$

and where $W = \emptyset$. This theory, Δ, has two regular extensions: $E_1 = Th(\{A, B\})$ and $E_2 = Th(\{A, \neg B\})$. Δ_{E_1} amounts to $\frac{:A}{A} < \frac{:\neg B}{\neg B} < \frac{:B}{B}$. Clearly, $(\Delta_{E_1})^{E_1} = \Delta_{E_1}$. Also, we obtain that $C(\Delta_{E_1}) = E_1$, that is, E_1 is a prioritized extension. In contrast to this, E_2 is not prioritized. While $\Delta_{E_2} = \Delta_{E_1}$ and $(\Delta_{E_2})^{E_2} = \Delta_{E_1}$, we get $C((\Delta_{E_2})^{E_2}) = E_1 \neq E_2$. That is, $C((\Delta_{E_2})^{E_2})$ reproduces E_1 rather than E_2.

This example reveals the difference between the prescriptive methodology of [6] discussed in the previous section, and Brewka and Eiter's descriptive approach discussed here, insofar as the former method actually selects *no prioritized extension*. Intuitively, this can be explained by the observation that for the highest-ranked default $\frac{A:B}{B}$, neither applicability nor blockage can be asserted: Either of these properties relies on the applicability of lesser-ranked defaults, effectively resulting in a circular situation destroying any possible extension. Nonetheless, as we show next, the methodology of [6] is general enough to admit a suitable preference strategy enforcing the simulation of prioritized extensions in the sense of Definition 9.

5 Prioritized Extensions via Standard Default Logic

Given an alphabet $\mathcal{P}$ of some language $\mathcal{L}_\mathcal{P}$, we define a disjoint alphabet $\mathcal{P}'$ as $\mathcal{P}' = \{p' \mid p \in \mathcal{P}\}$ (so implicitly there is an isomorphism between $\mathcal{P}$ and $\mathcal{P}'$). Then, for $\alpha \in \mathcal{L}_\mathcal{P}$, we define $\alpha' \in \mathcal{L}_{\mathcal{P}'}$ as the result of replacing in α each proposition p from $\mathcal{P}$ by the corresponding proposition p' in $\mathcal{P}'$. This is defined analogously for sets of formulas, default rules and sets of default rules. We abbreviate $\mathcal{L}_\mathcal{P}$ and $\mathcal{L}_{\mathcal{P}'}$ by $\mathcal{L}$ and $\mathcal{L}'$, respectively.

We obtain the following translation mapping prioritized default theories in some language $\mathcal{L}$ onto standard default theories in the language $\mathcal{L}^\circ$ obtained by extending $\mathcal{L} \cup \mathcal{L}'$ by new predicates symbols $(\cdot \prec \cdot)$, $\mathsf{ok}(\cdot)$, $\mathsf{ko}(\cdot)$, $\mathsf{bl}(\cdot)$, and $\mathsf{ap}(\cdot)$, and a set of associated default names:

Definition 10. *Given a prioritized default theory $\Delta = (D, W, <)$ over $\mathcal{L}$ and its set of default names $N = \{n_\delta \mid \delta \in D\}$, define $\mathcal{T}_{BE}(\Delta) = (D^\circ, W^\circ)$ over $\mathcal{L}^\circ$ by:*

$$D^\circ = D \cup \left\{ \tfrac{\mathsf{ok}(n_\delta) \wedge \alpha : \beta, \beta'}{\gamma' \wedge \mathsf{ap}(n_\delta)},\ \tfrac{\mathsf{ok}(n_\delta) : \neg\alpha, \neg\alpha'}{\mathsf{bl}(n_\delta)},\ \tfrac{\mathsf{ok}(n_\delta) \wedge \neg\beta \wedge \neg\beta' :}{\mathsf{bl}(n_\delta)} \;\middle|\; \delta = \tfrac{\alpha:\beta}{\gamma} \in D \right\} \quad (5)$$

$$\cup \left\{ \tfrac{: \neg(x \prec y)}{\neg(x \prec y)} \right\} \cup \left\{ \tfrac{\gamma \wedge \neg\beta :}{\mathsf{ko}(n_\delta)} \;\middle|\; \delta = \tfrac{\alpha:\beta}{\gamma} \in D \right\} \cup \left\{ \tfrac{: \exists x \in N.\, \neg\mathsf{ok}(x)}{\bot} \right\} \quad (6)$$

$$W^\circ = W \cup W' \quad (7)$$

$$\cup \{n_1 \prec n_2 \mid (\delta_1, \delta_2) \in <\} \cup \{\mathrm{DCA}_N, \mathrm{UNA}_N\} \quad (8)$$

$$\cup \{\forall x \in N.\, [\forall y \in N.\, \mathsf{ko}(y) \vee [(x \prec y) \supset (\mathsf{bl}(y) \vee \mathsf{ap}(y))]] \supset \mathsf{ok}(x)\} \quad (9)$$

We denote the second group of rules in (5) by δ°_a, $\delta^\circ_{b_1}$, and $\delta^\circ_{b_2}$; those in (6) are abbreviated by $\delta^\circ_\prec$, δ°_{ko}, and $\delta^\circ_\bot$, respectively.

It is important to note that the inclusions $D \subseteq D^\circ$ and $W \subseteq W^\circ$ hold. As we show in Theorem 2, this allows us to construct regular extensions of (D, W) within extensions of (D°, W°). Such an extension can be seen as the *guess* in a guess-and-check approach; it corresponds to Condition (*i*) in Definition 9.

The salient part of the corresponding *check*, viz. Condition (*ii*) in Definition 9, is accomplished by the second group of rules in (5) and the remaining facts in W°. Together with $W' \subseteq W^\circ$, the rules of form δ°_a aim at rebuilding the guessed extension in $\mathcal{L}'$. They form the prerequisite-free counterpart of the original default theory in $\mathcal{L}'$. In fact, the prerequisite of δ°_a refers via α to the guessed extension in $\mathcal{L}$; no formula in $\mathcal{L}'$ must be derived for applying δ°_a. This accounts for the elimination of prerequisites in

Condition (1) of Definition 8. Moreover, the elimination of rules whose prerequisites are not derivable is accomplished by rules of form $\delta^{\circ}_{b_1}$. Rules of form $\delta^{\circ}_{b_2}$ guarantee that defaults are only defeatable by rules with higher priority. In fact, it is $\neg\beta'$ that must be derivable in such a way only.

The application of rules according to the given preference information is enforced by axiom (9): For every n_i, we derive $\mathsf{ok}(n_i)$ whenever, for every n_j, either $\mathsf{ko}(n_j)$ is true, or, if $n_i \prec n_j$ holds, either $\mathsf{ap}(n_j)$ or $\mathsf{bl}(n_j)$ is true. This axiom allows us to derive $\mathsf{ok}(n_i)$, indicating that δ_i may potentially be applied, whenever we have for all δ_j with $\delta_i < \delta_j$ that δ_j has been applied or cannot be applied, or δ_j has already been eliminated from the preference handling process. This elimination of rules is in accord with Definition 7 and realized by δ°_{ko}. The preference information in (8) is rendered complete through rules of form $\delta^{\circ}_{\prec}$. This completion is necessary for the formula in (9) to work properly: whenever $(\delta_i, \delta_j) \notin <$, rule $\delta^{\circ}_{\prec}$ allows us to conclude (in the extension) that $\neg(n_i \prec n_j)$ holds.

Lastly, $\delta^{\circ}_{\bot}$ rules out unsuccessful attempts in rebuilding the regular extension from $\mathcal{L}$ within $\mathcal{L}'$ according to the given preference information. In this way, we eliminate all regular extensions that do not respect preference.

For illustration, reconsider theory (4), viz.

$$n_3 : \frac{:A}{A} < n_2 : \frac{:\neg B}{\neg B} < n_1 : \frac{A:B}{B}$$

and $W = \emptyset$. Recall that this theory has two regular extensions: one containing $\{A, \neg B\}$ and another containing $\{A, B\}$; but that only the latter is a prioritized extension according to [3]. We get:

$$\frac{:A}{A} \qquad \frac{:\neg B}{\neg B} \qquad \frac{A:B}{B}$$

$$\frac{\mathsf{ok}(n_3):A,A'}{A'\wedge\mathsf{ap}(n_3)} \quad \frac{\mathsf{ok}(n_2):\neg B,\neg B'}{\neg B'\wedge\mathsf{ap}(n_2)} \quad \frac{\mathsf{ok}(n_1)\wedge A:B,B'}{B'\wedge\mathsf{ap}(n_1)} \quad \frac{:\neg\mathsf{ok}(n_1)\vee\neg\mathsf{ok}(n_2)\vee\neg\mathsf{ok}(n_3)}{\bot}$$

$$\frac{\mathsf{ok}(n_1):\neg A,\neg A'}{\mathsf{bl}(n_1)}$$

$$\frac{\mathsf{ok}(n_3)\wedge\neg A\wedge\neg A':}{\mathsf{bl}(n_3)} \quad \frac{\mathsf{ok}(n_2)\wedge B\wedge B':}{\mathsf{bl}(n_2)} \quad \frac{\mathsf{ok}(n_1)\wedge\neg B\wedge\neg B':}{\mathsf{bl}(n_1)}$$

For brevity, we omit all defaults of form $\frac{\bot:}{\mathsf{ko}(n)}$.

First, suppose there is an extension with A and $\neg B$. Clearly, $\frac{:A}{A}$ and $\frac{:\neg B}{\neg B}$ contribute to such an extension. Having $\neg B$ denies the derivation of $\mathsf{ap}(n_1)$. Also, we do not get $\mathsf{bl}(n_1)$ since we can neither derive $\neg B'$ nor is $\neg A$ consistent. Therefore, we do not obtain $\mathsf{ok}(n_2)$; thus, $\neg\mathsf{ok}(n_2)$ is consistent and we obtain $\bot$ which destroys the putative extension at hand.

Next, consider a candidate extension with A and B. In this case, $\frac{:A}{A}$ and $\frac{A:B}{B}$ apply. Given $\mathsf{ok}(n_1)$ and A, we may derive $B' \wedge \mathsf{ap}(n_1)$. This gives $\mathsf{ok}(n_2)$ and then $\mathsf{ok}(n_2) \wedge B \wedge B'$, from which we get $\mathsf{bl}(n_2)$. Finally, we derive $\mathsf{ok}(n_3)$ and $A' \wedge \mathsf{ap}(n_3)$. Unlike the above, we cannot derive $\bot$ and we obtain an extension containing A and B.

For another example, consider the theory obtained from example (3):

$$\frac{:\neg B}{A} \quad \frac{:\neg A}{\neg A} \quad \frac{:A}{A} \quad \frac{:B}{B}$$

$$\frac{\mathsf{ok}(n_1):\neg B,\neg B'}{A'\wedge\mathsf{ap}(n_1)} \quad \frac{\mathsf{ok}(n_2):\neg A,\neg A'}{\neg A'\wedge\mathsf{ap}(n_2)} \quad \frac{\mathsf{ok}(n_3):A,A'}{A'\wedge\mathsf{ap}(n_3)} \quad \frac{\mathsf{ok}(n_4):B,B'}{B'\wedge\mathsf{ap}(n_3)} \quad \frac{:\exists x{\in}N.\,\neg\mathsf{ok}(x)}{\bot}$$

$$\frac{\mathsf{ok}(n_1)\wedge B\wedge B':}{\mathsf{bl}(n_1)} \quad \frac{\mathsf{ok}(n_2)\wedge A\wedge A':}{\mathsf{bl}(n_2)} \quad \frac{\mathsf{ok}(n_3)\wedge\neg A\wedge\neg A':}{\mathsf{bl}(n_3)} \quad \frac{\mathsf{ok}(n_4)\wedge\neg B\wedge\neg B':}{\mathsf{bl}(n_3)}$$

$$\frac{A\wedge B:}{\mathsf{ko}(n_1)}$$

While this theory has two regular extensions, it has no prioritized extension under the ordering imposed in (3). Suppose there is a prioritized extension containing A and B. This yields $\mathsf{ko}(n_1)$ and then (9) gives $\mathsf{ok}(n_2)$. Having A excludes $(\delta_2)^\circ_a$. Moreover, we cannot apply $(\delta_2)^\circ_{b_2}$ since A' is not derivable (by higher-ranked rules). We thus cannot derive $\mathsf{ok}(n_3)$, which leads to a destruction of the current extension through $\delta^\circ_\bot$.

The next theorem gives the major result of our paper.

Theorem 1. *Let $\Delta = (D, W, <)$ be a prioritized default theory over $\mathcal{L}$ and E a set of formulas over $\mathcal{L}$.*

E is a prioritized extension of Δ iff $E = F \cap \mathcal{L}$ and F is a (regular) extension of $\mathcal{T}_{BE}(\Delta)$.

In what follows, we elaborate upon the structure of the encoded default theories:

Theorem 2. *Let $\Delta = (D, W, <)$ be a prioritized default theory over $\mathcal{L}$ and let E° be a regular extension of $\mathcal{T}_{BE}(\Delta) = (D^\circ, W^\circ)$. Then, we have the following results:*

1. *$E^\circ \cap \mathcal{L}$ is a (regular) extension of (D, W);*
2. *$(E^\circ \cap \mathcal{L})' = E^\circ \cap \mathcal{L}'$ (or $\varphi \in E^\circ$ iff $\varphi' \in E^\circ$ for $\varphi \in \mathcal{L}$);*
3. *$\delta \in D \cap GD(D^\circ, E^\circ)$ iff $\delta^\circ_a \in GD(D^\circ, E^\circ)$;*
4. *$\delta \in D \setminus GD(D^\circ, E^\circ)$ iff $\delta^\circ_{b_1} \in GD(D^\circ, E^\circ)$ or $\delta^\circ_{b_2} \in GD(D^\circ, E^\circ)$;*
5. *if $\delta^\circ_{ko} \in GD(D^\circ, E^\circ)$, then $\delta^\circ_{b_2} \in GD(D^\circ, E^\circ)$.*

The last property shows that eliminated rules are eventually found to be inapplicable. This illustrates another choice of our translation: instead of using the second group of rules in (5), we could have used

$$\left\{ \frac{\mathsf{ok}(n)\wedge\alpha:\beta,\beta',\neg\mathsf{ko}(n)}{\gamma'\wedge\mathsf{ap}(n)},\ \frac{\mathsf{ok}(n):\neg\alpha,\neg\alpha',\neg\mathsf{ko}(n)}{\mathsf{bl}(n)},\ \frac{\mathsf{ok}(n)\wedge\neg\beta\wedge\neg\beta':\neg\mathsf{ko}(n)}{\mathsf{bl}(n)} \;\middle|\; n:\frac{\alpha:\beta}{\gamma}\in D \right\}.$$

Although this renders the derivation of $\mathsf{ap}(n)$, $\mathsf{bl}(n)$, and $\mathsf{ko}(n)$ mutually exclusive, the additional justification $\neg\mathsf{ko}(n)$ is not needed. That is, it is sufficient to remove $\frac{\alpha:\beta}{\gamma}$ from the preference handling process; the rule is found to be blocked anyway.

The following theorem summarizes some technical properties of our translation:

Theorem 3. *Let E be a consistent extension of $\mathcal{T}_{BE}(\Delta)$ for prioritized default theory $\Delta = (D, W, <)$. We have for all $\delta, \delta' \in D$ that*

1. *$n_\delta \prec n_{\delta'} \in E$ iff $\neg(n_\delta \prec n_{\delta'}) \notin E$;*
2. *$\mathsf{ok}(n_\delta) \in E$;*
3. *$\mathsf{ap}(n_\delta) \in E$ iff $\mathsf{bl}(n_\delta) \notin E$.*

The two last results reveal an alternative choice for $\delta^\circ_\bot$, namely $\frac{:\exists x{\in}N.\,\neg\mathsf{ap}(x){\wedge}\neg\mathsf{bl}(x)}{\bot}$.

One may wonder how our translation avoids the explicit use of total extensions of the given partial order. The next theorem shows that these total extensions are reflected by the grounded enumerations of the second group of rules in (5):

Theorem 4. *Given the same prerequisites as in Theorem 2, let $\langle\delta^\circ_i\rangle_{i\in I}$ be some grounded enumeration of GD(D°, E°). For all $\delta_1, \delta_2 \in D^{E^\circ\cap\mathcal{L}}$, define $\delta_1 \ll \delta_2$ iff $k_2 < k_1$ where $k_j = \min\{i \in I \mid \delta^\circ_i = (\delta_j)^\circ_x$ for $x \in \{a, b_1, b_2\}\}$ for $k = 1, 2$. Then, $\ll$ is a total ordering on $D^{E^\circ\cap\mathcal{L}}$ such that $\ll \subseteq (< \cap (D^{E^\circ\cap\mathcal{L}} \times D^{E^\circ\cap\mathcal{L}}))$.*

That is, whenever $\Delta = \Delta^E$ according to Definition 7, we have that $\ll$ is a total ordering on D such that $\ll{\subseteq}<$.

Finally, one may ask why we do not need to account for the "inherited" ordering in Condition 2 of Definition 8. In fact, this is taken care of through the "tags" $\mathsf{ap}(n_\delta)$ in the consequents of rules δ°_a that guarantee an isomorphism between D and D_E in Definition 8. More generally, such a "tagging of consequents" provides an effective correspondence between the applicability of default rules and the presence of their consequents in an extension at hand. As a side effect, this facilitates the notion of activeness in Section 4 by rendering Condition (iii) unnecessary.

6 Compiling Prioritized Answer Sets

In this section, we describe how Brewka and Eiter's preference approach [3] for extended logic programs can be encoded within standard answer set semantics, following the methodology developed in [8]. We commence with a recapitulation of the necessary concepts.

As usual, a *literal*, L, is an expression of the form p or $\neg p$, where p is an atom. The set of all literals is denoted by Lit. A *rule*, r, is an expression of the form

$$L_0 \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, \tag{10}$$

where $n \geq m \geq 0$, and each L_i $(0 \leq i \leq n)$ is a literal. The symbol "not" denotes *negation as failure*, or *weak negation*. Accordingly, the classical negation sign "$\neg$" is in this context also said to represent *strong negation*. The literal L_0 is called the *head* of r, and the set $\{L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\}$ is the *body* of r. We use $head(r)$ to denote the head of rule r, and $body(r)$ to denote the body of r. Furthermore, let $body^+(r) = \{L_1, \ldots, L_m\}$ and $body^-(r) = \{L_{m+1}, \ldots, L_n\}$. The elements of $body^+(r)$ are referred to as the *prerequisites* of r. If $body^+(r) = \emptyset$, then r is a *prerequisite-free rule*; if $body(r) = \emptyset$, then r is a *fact*; if r contains no variables, then r is *ground*. We say that a rule r is *defeated* by a set of literals X iff $body^-(r) \cap X \neq \emptyset$. As well, each literal in $body^-(r) \cap X$ is said to *defeat* r. We define $not\,X$ as the set $\{not\ L \mid L \in X\}$.

A set of literals X is *consistent* iff it does not contain a complementary pair p, $\neg p$ of literals. We say that X is *logically closed* iff it is either consistent or equals Lit.

A *rule base* is any collection of rules; an (*extended*) *logic program*, or simply a *program*, is a finite rule base. A rule base (program) is *prerequisite-free* (*ground*) if all rules in it are prerequisite-free (ground).

For a rule base R, we denote by R^* the ground instantiation of R over the Herbrand universe of the language $\mathcal{L}$ of R.

The answer set semantics interprets ground rules of the form (10) as defaults

$$\frac{L_1 \wedge \ldots \wedge L_m \ : \ \neg L_{m+1}, \ldots, \neg L_n}{L_0}. \tag{11}$$

A set X of ground literals is called an *answer set* of the ground program P iff X is of the form $E \cap Lit$, where E is an extension of the default theory obtained by identifying each rule $r \in P$ as a default of the form (11). Answer sets of programs not necessarily ground are obtained by taking the answer sets of the ground instantiation P^* of P.

A *prioritized logic program* is a pair $\Pi = (P, <)$, where P is a logic program and $<$ is a strict partial order. Following [3], the ground instantiation of a prioritized logic program $(P, <)$ is obtained as follows: Let P^* be the ground instantiation of P and define $r^* <^* s^*$ for $r^*, s^* \in P^*$ providing r^*, s^* are instances of $r, s \in P$, respectively, such that $r < s$. If $<^*$ is a strict partial order, then the pair $(P^*, <^*)$ defines the ground instantiation of $(P, <)$; otherwise, the ground instantiation of $(P, <)$ is undefined. In the sequel, we will be concerned with ground prioritized programs only.

A *fully prioritized logic program* is a prioritized logic program $(P, <)$ where $<$ is a total ordering. Prioritized answer sets of prioritized logic programs are defined similarly to prioritized extensions of prioritized default theories. That is to say, first the prerequisite-free case is treated, and afterwards the general case is addressed in terms of the prerequisite-free case.

For fully prioritized ground programs, Definitions 5 and 7 boil down to the following operator: Let $\Pi = (P, <)$ be a fully prioritized ground prerequisite-free logic program, $\langle r_i \rangle_{i \in I}$ be an enumeration of the ordering $<$, and X be a set of literals. Then, $C_\Pi(X)$ is the smallest logically closed set of literals containing $\bigcup_{i \in I} X_i$, where

$$X_i = \begin{cases} \bigcup_{j<i} X_j & \text{if } r_i \text{ is defeated by } \bigcup_{j<i} E_j, \text{ or} \\ & head(r_i) \in X \text{ and } r_i \text{ is defeated by } X; \\ \bigcup_{j<i} X_j \cup \{head(r_i)\} & \text{otherwise.} \end{cases}$$

As in the default logic case, this construction is unique in the sense that for a fully prioritized prerequisite-free ground program Π, there is at most one answer set X of P such that $C_\Pi(X) = X$ (cf. [3, Lemma 4.1]). Accordingly, this set is referred to as the *prioritized answer set* of Π, if it exists. Prioritized answer sets of an arbitrary (i.e., not necessarily prerequisite-free) ground fully prioritized program $\Pi = (P, <)$ are given by sets X of ground literals which are prioritized answer sets of the prioritized program $\Pi_X = (P_X, <_X)$, where $<_X$ is constructed just as the ordering $<_E$ of Definition 8, and P_X results from P by (i) deleting any rule $r \in P$ such that $body^+(r) \not\subseteq X$, and (ii) removing any prerequisites in the body of the remaining rules. Lastly, X is a prioritized answer set of a ground prioritized logic program $(P, <)$ iff (i) X is a (regular) answer set of P and (ii) X is a prioritized answer set of some fully prioritized program $(P, <')$ such that $< \subseteq <'$.

This concludes the review of prioritized answer sets according to [3]; we continue with a compilation of this approach in standard answer set semantics.

As in Section 5, given a ground prioritized program Π over language $\mathcal{L}$, we assume a disjoint language $\mathcal{L}'$ containing literals L' for each L in $\mathcal{L}$. Likewise, rule r' results from r by replacing each literal L in r by L'. We maintain for rules the same naming convention as for defaults, i.e., the term n_r serves as name for rule r, similarly writing $n : r$ as before. As well, the language $\mathcal{L}^\circ$ extends $\mathcal{L} \cup \mathcal{L}'$ by new ground atoms $(n_r \prec n_s)$, $\mathsf{ok}(n_r)$, $\mathsf{ko}(n_r)$, $\mathsf{ry}(n_r, n_s)$, $\mathsf{bl}(n_r)$, and $\mathsf{ap}(n_r)$, for each r, s in Π.

Definition 11. *Let $\Pi = (P, <)$ be a prioritized ground logic program over $\mathcal{L}$ such that $P = \{r_1, \ldots, r_k\}$. Then, the logic program $\mathcal{T}^{lp}_{BE}(\Pi)$ over $\mathcal{L}^\circ$ is given by*

$$P \cup \bigcup\nolimits_{r \in P} \tau(r) \cup \{(n_1 \prec n_2) \leftarrow \mid (r_1, r_2) \in <\},$$

where $\tau(r)$ consists of the following collection of rules, for $L \in body^+(r)$, $K \in body^-(r)$, and $s \in P$:

$$\begin{array}{rl}
a_1(r): & head(r') \leftarrow \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) \leftarrow \mathsf{ok}(n_r), body(r), not\ body^-(r') \\
b_1(r, L): & \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), not\ L, not\ L' \\
b_2(r, K): & \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), K, K' \\
c_1(r): & \mathsf{ok}(n_r) \leftarrow \mathsf{ry}(n_r, n_{r_1}), \ldots, \mathsf{ry}(n_r, n_{r_k}) \\
c_2(r, s): & \mathsf{ry}(n_r, n_s) \leftarrow not\ (n_r \prec n_s) \\
c_3(r, s): & \mathsf{ry}(n_r, n_s) \leftarrow (n_r \prec n_s), \mathsf{ap}(n_s) \\
c_4(r, s): & \mathsf{ry}(n_r, n_s) \leftarrow (n_r \prec n_s), \mathsf{bl}(n_s) \\
c_5(r, s): & \mathsf{ry}(n_r, n_s) \leftarrow \mathsf{ko}(n_s) \\
d(r): & \bot \leftarrow not\ \mathsf{ok}(n_r) \\
e(r, K): & \mathsf{ko}(n_r) \leftarrow head(r), K
\end{array}$$

The first group of rules in $\tau(r)$ expresses applicability and blocking conditions of r and contains the counterparts of the defaults δ°_a, $\delta^\circ_{b_1}$, and $\delta^\circ_{b_2}$ in Definition 10, respectively. To wit, applicability of r is captured by the two rules $a_1(r)$ and $a_2(r)$, while k rules of the form $b_1(r, L)$ and $b_2(r, K)$ detect blockage of r, where k is the number of literals in $body(r)$. The second group of rules unfolds axiom (9) and relies on auxiliary atoms $\mathsf{ry}(\cdot, \cdot)$ ("ready"), taking care of instantiating the quantification over names expressed in (9). Finally, rules $d(r)$ and $e(r, K)$ correspond to δ°_{ko}, and $\delta^\circ_\bot$, respectively.

We obtain the following result corresponding to Theorem 1:

Theorem 5. *Let $\Pi = (P, <)$ be a prioritized ground logic program over $\mathcal{L}$ and X a set of literals over $\mathcal{L}$.*

X is a prioritized answer set of Π iff $X = Y \cap \mathcal{L}$ and Y is a (regular) answer set of $\mathcal{T}^{lp}_{BE}(\Pi)$.

Additionally, given suitable concepts for the present case, analogous results to Theorems 2, 3, and 4 can be shown. We just note the counterpart of Theorem 3:

Theorem 6. *Let X be a consistent answer set of $\mathcal{T}^{lp}_{BE}(\Pi)$ for prioritized logic program $\Pi = (P, <)$. We have for all $r \in P$ that*

1. $\mathsf{ok}(n_\delta) \in X$;

2. $\mathsf{ap}(n_\delta) \in X$ *iff* $\mathsf{bl}(n_\delta) \notin X$.

The approach is implemented in Prolog and serves as a front-end to the logic programming systems `dlv` [9] and `smodels` [13]. Our current prototype, called plp, is available at `http://www.cs.uni-potsdam.de/~torsten/plp/`. This URL contains also diverse examples taken from the literature. The implementation differs from the approach described here, in that the translation applies to named rules only; it thus leaves unnamed rules unaffected.

For illustration, consider the logic programming counterpart of Example (4) in the syntax of plp :

```
 b :- name(1), not -b, a.
-b :- name(2), not b.              2<1.
 a :- name(3), not -a.             3<2.
```

We use '`-`' (or '`neg`') for classical negation and '`not`' (or '`~`') for negation as failure. Furthermore, `name(·)` is used to identify rule names; and natural numbers serve as names. Note that our implementation handles transitivity implicitly, so that there is no need to specify `3<1`.

This is then translated into the following (intermediate) standard program:

```
(1)    b :- not neg b, a.
(2)    b1 :- ap(1).
(3)    ap(1) :- name(1), ok(1), not neg b, not neg b1, a.
(4)    bl(1) :- ok(1), neg b, neg b1.
(5)    bl(1) :- ok(1), not a, not a1.
(6)    ko(1) :- b, neg b.
(7)    neg b :- not b.
(8)    neg b1 :- ap(2).
(9)    ap(2) :- name(2), ok(2), not b, not b1.
(10)   bl(2) :- ok(2), b, b1.
(11)   ko(2) :- neg b, b.
(12)   a :- not neg a.
(13)   a1 :- ap(3).
(14)   ap(3) :- name(3), ok(3), not neg a, not neg a1.
(15)   bl(3) :- ok(3), neg a, neg a1.
(16)   ko(3) :- a, neg a.
(17)   2 < 1.
(18)   3 < 2.
(19)   neg M < N :- name(N), name(M), N < M.
(20)   N < M :- name(N), name(M), name(O), N < O, O < M.
(21)   ok(N) :- name(N), ry(N, 1), ry(N, 2), ry(N, 3).
(22)   ry(N, M) :- name(N), name(M), not N < M.
(23)   ry(N, M) :- name(N), name(M), N < M, ap(M).
(24)   ry(N, M) :- name(N), name(M), N < M, bl(M).
(25)   ry(N, M) :- name(N), name(M), ko(M).
(26)   false :- name(N), not ok(N).
```

The original rules, viz. r_1, r_2, and r_3, are given by `(1)`,`(7)`, and `(12)`. The additional encoding of, e.g., rule `(1)` is given by `(2)` to `(6)`. We append the symbol '`1`'

for priming here, e.g., `b1` is the primed version of `b`. In detail, `(2)` and `(3)` correspond to $a_1(r_1)$ and $a_2(r_1)$, `(4)` and `(5)` correspond to $b_2(r_1, B)$ and $b_1(r_1, A)$, and finally `(6)` corresponds to $e(r_1, B)$. Rules `(19)` and `(20)` are additional rules enforcing a strict partial order. Rules `(21)` to `(25)` account for $c_1(r)$ to $c_5(r, s)$. Lastly, `(26)` implements $d(r)$.

The above program is then refined once more in order to account for some special features of `dlv` and `smodels`, like implementation of classical negation '`neg`' and '`false`'. Also, an extensional database for rule names is provided.

Calling one of these provers with the respective input corresponding to the above program, we obtain the desired prioritized answer set containing the literals A and B (i.e., represented by `a` and `b`).

7 Conclusion

We have shown how the approach of Brewka and Eiter, both with respect to extended logic programs [3] and to default logic [4], can be expressed in our general framework for preferences [6, 8]. On the one hand, this illustrates the generality of our framework; on the other hand, it sheds light on Brewka and Eiter's approaches, since it provides a translation and encoding of their approaches into extended logic programs and default logic, respectively. As well, our encoding allows a straightforward implementation of [3] via a translation into extended logic programs.

Lastly, we note that our approach described in [8] used *dynamic* preference information, in that preferences were expressed within a logic program. As well, in the case of default logic, [6] also describes the incorporation of dynamic preferences. Thus in these approaches, preferences can be encoded as holding only in specific contexts, holding by default, and so on. Such a dynamic setting was also sketched in [4]. It is a straightforward matter to extend Definitions 10 and 11 to handle this dynamic case as well.

Acknowledgements The first author was partially supported by a Research Grant from the Natural Sciences and Engineering Research Council of Canada. The second author was partially supported by the German Science Foundation (DFG) under grant FOR 375/1-1, TP C. The third author was partially supported by the Austrian Science Fund (FWF) under grants N Z29-INF and P13871-INF.

References

[1] F. Baader and B. Hollunder. How to prefer more specific defaults in terminological default logic. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 669–674, 1993.

[2] G. Brewka. Reasoning about priorities in default logic. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, volume 2, pages 940–945. The AAAI Press/The MIT Press, 1994.

[3] G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.

[4] G. Brewka and T. Eiter. Prioritizing default logic. In St. Hölldobler, editor, *Intellectics and Computational Logic — Papers in Honour of Wolfgang Bibel*. Kluwer Academic Publishers, 2000. To appear.

[5] P. Cholewiński, V. Marek, and M. Truszczyński. Default reasoning system DeReS. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning*, pages 518–528. Morgan Kaufmann Publishers, 1996.

[6] J. Delgrande and T. Schaub. Compiling reasoning with and about preferences into default logic. In M. Pollack, editor, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 168–174. Morgan Kaufmann Publishers, 1997.

[7] J. Delgrande, T. Schaub, and H. Tompits. A compiler for ordered logic programs. In C. Baral and M. Truszczyński, editors, *Proceedings of the Eighth International Workshop on Non-Monotonic Reasoning*. `arXiv.org` e-Print archive, 2000. System Abstract.

[8] J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In C. Baral and M. Truszczyński, editors, *Proceedings of the Eighth International Workshop on Non-Monotonic Reasoning*. `arXiv.org` e-Print archive, 2000.

[9] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 363–374. Springer Verlag, 1997.

[10] M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991.

[11] T. Janhunen. Classifying semi-normal default logic on the basis of its expressive power. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 19–33. Springer Verlag, 1999.

[12] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[13] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the Fourth International Conference on Logic Programing and Nonmonotonic Reasoning*, pages 420–429. Springer, 1997.

[14] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

[15] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2):81–132, 1980.