TLP **3** (2): 129–187, March 2003. © 2003 Cambridge University Press
DOI: 10.1017/S1471068402001539 Printed in the United Kingdom

A framework for compiling preferences in logic programs

JAMES P. DELGRANDE

School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6
(*e-mail:* jim@cs.sfu.ca)

TORSTEN SCHAUB*

Institut für Informatik, Universität Potsdam, Postfach 90 03 27, D–14439 Potsdam, Germany
(*e-mail:* torsten@cs.uni-potsdam.de)

HANS TOMPITS

Institut für Informationssysteme 184/3, Technische Universität Wien, Favoritenstraße 9–11, A–1040 Vienna, Austria
(*e-mail:* tompits@kr.tuwien.ac.at)

Abstract

We introduce a methodology and framework for expressing general preference information in logic programming under the answer set semantics. An *ordered* logic program is an extended logic program in which rules are named by unique terms, and in which preferences among rules are given by a set of atoms of form $s \prec t$ where s and t are names. An ordered logic program is transformed into a second, regular, extended logic program wherein the preferences are respected, in that the answer sets obtained in the transformed program correspond with the preferred answer sets of the original program. Our approach allows the specification of *dynamic* orderings, in which preferences can appear arbitrarily within a program. *Static* orderings (in which preferences are external to a logic program) are a trivial restriction of the general dynamic case. First, we develop a specific approach to reasoning with preferences, wherein the preference ordering specifies the order in which rules are to be applied. We then demonstrate the wide range of applicability of our framework by showing how other approaches, among them that of Brewka and Eiter, can be captured within our framework. Since the result of each of these transformations is an extended logic program, we can make use of existing implementations, such as dlv and smodels. To this end, we have developed a publicly available compiler as a front-end for these programming systems.

KEYWORDS: answer sets semantics, preference, compilation

1 Introduction

In commonsense reasoning in general, and in logic programming in particular, one frequently prefers one conclusion over another, or the application of one rule over

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

another. For example, in buying a car one may have various desiderata in mind (inexpensive, safe, fast, etc.) where these preferences come with varying degrees of importance. In legal reasoning, laws may apply in different situations, but laws may also conflict with each other. Conflicts are resolved by appeal to higher-level principles such as authority or recency. So federal laws will have a higher priority than state laws, and newer laws will take priority over old. Further preferences, such as authority holding sway over recency, may also be required.

In logic programming, in *basic* logic programs, which do not employ negation as failure, there is no issue with preferences, since a given basic logic program is guaranteed to have a single, unique, set of conclusions. However, basic logic programs are expressively weak. Once negation as failure is introduced, as in *extended* logic programs (Gelfond and Lifschitz, 1991), we are no longer guaranteed a single set of conclusions, but rather may have several *answer sets*, each giving a feasible set of conclusions. There is no a priori reason to accept one answer set over another, yet, as noted above, we may have reasons to prefer one over another.

In this situation, preferences are usually expressed by a strict partial order on the set of rules. For example, consider the following program (technical definitions and notation are introduced in the next section):

$$\begin{array}{rcrcl} r_1 & = & \neg a & \leftarrow & \\ r_2 & = & b & \leftarrow & \neg a, not\ c \\ r_3 & = & c & \leftarrow & not\ b. \end{array}$$

Each r_i identifies the respective rule. This program has two regular answer sets, one given by $\{\neg a, b\}$ and the other given by $\{\neg a, c\}$. For the first answer set, rules r_1 and r_2 are applied; for the second, r_1 and r_3. However, assume that we have reason to prefer r_2 to r_3, expressed by $r_3 < r_2$. In this case we would want to obtain just the first answer set.

There have been numerous proposals for expressing preferences in extended logic programs (Sakama and Inoue, 1996; Brewka, 1996; Gelfond and Son, 1997; Zhang and Foo, 1997; Brewka and Eiter, 1999; Wang *et al.*, 2000). The general approach in such work has been to employ meta-formalisms for characterising 'preferred answer sets'. For instance, a common approach is to generate all answer sets for a program and then, in one fashion or other, select the most preferred set(s). Consequently, non-preferred as well as preferred answer sets are first generated, and the preferred sets next isolated by a filtering step. Such approaches generally have a higher complexity than the underlying logic programming semantics (see section 7 for details).

Our goal in this paper is to (i) present a general methodology and framework for expressing and implementing preferences where (ii) only preferred answer sets are generated. We do this by describing a general methodology in which a logic program with preferences is translated into a second 'regular' logic program, such that the answer sets of the second program in a precise sense correspond to and express only the preferred answer sets of the first. This makes it possible to encode preferences within the very same logic programming framework. As we argue, the framework is very general and admits the encoding of different preference strategies. Moreover, we are able to express *dynamic* preferences within a logic program, in contrast to most previous work, which adopts an external *static* preference order. The complexity of our approach is in the same complexity class as the underlying logic programming semantics. This approach is suited to a *prescriptive* interpretation of preference, wherein the preference ordering specifies the order in which the rules are to be applied.

We begin by developing and exploring our ‘preferred’ interpretation of preference, using a strongly prescriptive interpretation of preference. However, we also show how it is possible to encode other interpretations. To this end, we show how Brewka and Eiter’s approach to preference (Brewka and Eiter, 1999) can be expressed in the framework. This encoding shows that we can also handle more descriptive-oriented approaches. Encodings of other approaches, such as that given in Wang *et al.* (2000), are briefly described as well.

The general framework then also provides a common basis in which different approaches can be expressed and compared. Equivalently, the framework provides a common setting in which various different strategies can be encoded. Thus it provides a uniform way of capturing the different strategies that are originally given in rather heterogeneous ways. The translations then in a sense *axiomatise* how a preference ordering is to be understood. While the notion of semantics as such is not our major concern, we do discuss various encodings in connection with the notion of *order preservation*. In fact, the strategy discussed in section 4 was developed from just this concept of order preservation.

Finally, since we translate a program into an extended logic program, it is straightforward implementing our approach. To this end, we have developed a translator for ordered logic programs that serves as a front-end for the logic programming systems dlv (Eiter *et al.*, 1997) and smodels (Niemelä and Simons, 1997). The possibility to utilise existing logic programming systems for implementation purposes is a major advantage of our framework. In contrast, most other approaches are based on a change of semantics and thus require dedicated algorithms to solve the respective reasoning tasks at hand.

The next section gives background terminology and notation, while section 3 describes the overall methodology. We develop our central approach in section 4 and explore its formal properties. Section 5 presents our encoding of Brewka and Eiter’s approach, while section 6 deals with our implementation. Section 7 considers other work, and section 8 concludes with some further issues and a brief discussion. This paper regroups and strongly extends the work found in Delgrande, Schaub and Tompits (2000c; 2000b; 2000d; 2000a; 2001).

2 Definitions and notation

We deal with extended logic programs (Lifschitz, 1996) that contain the symbol $\neg$ for *classical negation* in addition to *not* used for *negation as failure*. This allows for distinguishing between goals that fail in the sense that they *do not succeed* and goals that fail in the stronger sense that their *negation succeeds*. Classical negation is thus also referred to as *strong negation*, whilst negation as failure is termed *weak negation*.

Our formal treatment is based on propositional languages. Let $\mathscr{A}$ be a non-empty set of symbols, called *atoms*. The choice of $\mathscr{A}$ determines the language of the programs under consideration. As usual, a *literal*, L, is an expression of form A or $\neg A$, where A is an atom. We assume a possibly infinite set of such atoms. The set of all literals over $\mathscr{A}$ is denoted by $\mathscr{L}_{\mathscr{A}}$. A literal preceded by the negation as failure sign *not* is said to be a *weakly negated literal*. A *rule*, r, is an expression of form

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

where $n \geqslant m \geqslant 0$, and each L_i $(0 \leqslant i \leqslant n)$ is a literal. 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. If $n = m$, then r is said to be a *basic rule*; if $n = 0$, then r is called a *fact*. An (*extended*) *logic program*, or simply a *program*, is a finite set of rules. A program is *basic* if all rules in it are basic. For simplicity, we associate the language of a program with the set of literals $\mathscr{L}_{\mathscr{A}}$, rather than the set of all rules induced by $\mathscr{L}_{\mathscr{A}}$.

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 $body^-(r) = \{L_{m+1}, \ldots, L_n\}$ for r as in (1). The elements of $body^+(r)$ are referred to as the *prerequisites* of r. Thus, if $body^+(r) = \emptyset$ (or $m = 0$), then r is said to be *prerequisite free*. We say that rule r is *defeated* by a set X of literals iff $body^-(r) \cap X \neq \emptyset$. Given that $body(r) \neq \emptyset$, we also allow the situation where r has an empty head, in which case r is called an *integrity constraint*, or *constraint*, for short. A constraint with body as in (1) appearing in some program Π can be regarded as a rule of form

$$p \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, not\ p,$$

where p is an atom not occurring elsewhere in Π.

A set of literals $X \subseteq \mathscr{L}_{\mathscr{A}}$ is *consistent* iff it does not contain a complementary pair A, $\neg A$ of literals. We say that X is *logically closed* iff it is either consistent or equals $\mathscr{L}_{\mathscr{A}}$. Furthermore, X is *closed under* a basic program Π iff, for any $r \in \Pi$, $head(r) \in X$ whenever $body(r) \subseteq X$. In particular, if X is closed under the basic program Π, then $body(r) \nsubseteq X$ for any constraint $r \in \Pi$. The smallest set of literals which is both logically closed and closed under a basic program Π is denoted by $Cn(\Pi)$.

Let Π be a basic program and $X \subseteq \mathscr{L}_{\mathscr{A}}$ a set of literals. The operator T_Π is defined as follows:

$$T_\Pi X = \{head(r) \mid r \in \Pi \text{ and } body(r) \subseteq X\}$$

if X is consistent, and $T_\Pi X = \mathscr{L}_{\mathscr{A}}$ otherwise. Iterated applications of T_Π are written as T^j_Π for $j \geqslant 0$, where $T^0_\Pi X = X$ and $T^i_\Pi X = T_\Pi T^{i-1}_\Pi X$ for $i \geqslant 1$. It is well-known that $Cn(\Pi) = \bigcup_{i \geqslant 0} T^i_\Pi \emptyset$, for any basic program Π.

Let r be a rule. Then r^+ denotes the basic rule obtained from r by deleting all weakly negated literals in the body of r, that is, $r^+ = head(r) \leftarrow body^+(r)$. Accordingly, we define for later usage $r^- = head(r) \leftarrow body^-(r)$. The *reduct*, Π^X, of a program Π relative to a set X of literals is defined by

$$\Pi^X = \{r^+ \mid r \in \Pi \text{ and } body^-(r) \cap X = \emptyset\}.$$

In other words, Π^X is obtained from Π by:

1. deleting any $r \in \Pi$ which is defeated by X, and
2. deleting each weakly negated literal occurring in the bodies of the remaining rules.

We say that a set X of literals is an *answer set* of a program Π iff $Cn(\Pi^X) = X$. Clearly, for each answer set X of a program Π, it holds that $X = \bigcup_{i \geqslant 0} T^i_{\Pi^X}\emptyset$.

The set Γ^X_Π of all *generating rules* of an answer set X from Π is given by

$$\Gamma^X_\Pi = \{r \in \Pi \mid body^+(r) \subseteq X \text{ and } body^-(r) \cap X = \emptyset\}.$$

That is, Γ^X_Π comprises all rules $r \in \Pi$ such that r is not defeated by X and each prerequisite of r is in X. Finally, a sequence $\langle r_i \rangle_{i \in I}$ of rules is *grounded* iff, for all $i \in I$, $body^+(r_i) \subseteq \{head(r_j) \mid j < i\}$ providing $\{head(r_j) \mid j < i\}$ is consistent. We say that a rule r is *grounded*[1] in a set R of rules iff there is an grounded enumeration of R and $body^+(r) \subseteq \{head(r) \mid r \in R\}$.

The answer set semantics for extended logic programs has been defined in Gelfond and Lifschitz (1991) as a generalisation of the stable model semantics (Gelfond and Lifschitz, 1988) for *general logic programs* (i.e., programs not containing classical negation, $\neg$). The reduct Π^X is often called the *Gelfond–Lifschitz reduction.*

3 From ordered to tagged logic programs

A logic program Π over a propositional language $\mathscr{L}$ is said to be *ordered* if $\mathscr{L}$ contains the following pairwise disjoint categories:

- a set N of terms serving as *names* for rules;
- a set $\mathscr{A}$ of regular (propositional) atoms of a program; and
- a set $\mathscr{A}_\prec$ of *preference atoms* $s \prec t$, where $s, t \in N$ are names.

We assume, furthermore, a bijective[2] function $n(\cdot)$ assigning to each rule $r \in \Pi$ a name $n(r) \in N$. To simplify our notation, we usually write n_r instead of $n(r)$ (and we sometimes abbreviate n_{r_i} by n_i). Also, the relation $t = n(r)$ is written as $t : r$, leaving the naming function $n(\cdot)$ implicit. The elements of $\mathscr{A}_\prec$ express preferences among rules. Intuitively, $n_r \prec n_{r'}$ asserts that r' has 'higher priority' than r. Thus, r' is viewed as having precedence over r. That is, r' should, in some sense, always be considered 'before' r. (Note that some authors use $\prec$ or $<$ in the opposite sense from us.)

Formally, given an alphabet $\mathscr{A}$, an ordered logic program can be understood as a triple (Π, N, n), where Π is an extended logic program over $\mathscr{L}_{\mathscr{A} \cup \mathscr{A}_\prec}$ and n is a bijective function between Π and the set of names N.[3] In what follows, we leave the

[1] Note that some authors (for example, Niemelä and Simons (1997)) use *grounded* to refer to the process of eliminating variables from a rule by replacing it with its set of ground instances. We use the term *grounded* in reference to grounded enumerations only. When we discuss replacing variables with ground terms in section 6, we will refer to an *instantiation* of a rule.

[2] In practice, function n is only required to be injective since only rules participating in the preference relation require names.

[3] Note that $\mathscr{A}_\prec$ is determined by N.

set of names N and the naming function n implicit and rather associate the notion of an ordered logic program with the underlying extended logic program Π.

It is important to note that we impose no restrictions on the occurrences of preference atoms. This allows for expressing preferences in a very flexible, dynamic way. For instance, we may specify

$$n_r \prec n_{r'} \leftarrow p, not\ q$$

where p and q may themselves be (or rely on) preference atoms.

A special case is given by programs containing preference atoms only among their facts. We say that a logic program Π over $\mathscr{L}$ is *statically ordered* if it is of form $\Pi = \Pi' \cup \Pi''$, where Π' is an extended logic program over $\mathscr{L}_{\mathscr{A}}$ and $\Pi'' \subseteq \{(n_r \prec n_{r'}) \leftarrow \ \mid r, r' \in \Pi'\}$. The static case can be regarded as being induced from an external order, $<$, where the relation $r < r'$ between two rules holds iff the fact $(n_r \prec n_{r'}) \leftarrow$ is included in the ordered program. We make this explicit by denoting a statically ordered program Π as a pair $(\Pi', <)$, representing the program $\Pi' \cup \{(n_r \prec n_{r'}) \leftarrow \ \mid r, r' \in \Pi', r < r'\}$. We stipulate for each statically ordered program $(\Pi, <)$ that $<$ is a strict partial order. The static concept of preference corresponds to most previous approaches to preference handling in logic programming and nonmonotonic reasoning, where the preference information is specified as a fixed relation at the meta-level (Baader and Hollunder, 1993; Brewka, 1994; Zhang and Foo, 1997; Brewka and Eiter, 1999; Wang *et al*., 2000).

The idea behind our methodology for compiling preferences is straightforward. Given a preference handling strategy σ, our approach provides a mapping $\mathscr{T}_\sigma$ that transforms an ordered logic program Π into a standard logic program $\mathscr{T}_\sigma(\Pi)$, such that the answer sets of Π preferred by σ correspond to the (standard) answer sets of $\mathscr{T}_\sigma(\Pi)$. Intuitively, the translated program $\mathscr{T}_\sigma(\Pi)$ is constructed in such a way that the resulting answer sets comply with σ's interpretation of the preference information (induced by the original program Π). This is achieved by adding sufficient control elements to the rules of Π that will guarantee that successive rule applications are in accord with the intended order. Such control elements, or *tags* for short, are given through newly introduced atoms that allow us to detect and control rule applications within the object language. Hence, a translation $\mathscr{T}_\sigma$ maps ordered logic programs onto standard logic programs whose language is extended by appropriate tags.

Given the relation $r < r'$ (or the atom $n_r \prec n_{r'}$, respectively), we want to ensure that r' is considered before r, in the sense that rule r' is known to be applied or blocked ahead of r (with respect to the order of rule application). We do this by first translating rules so that the order of rule application can be explicitly controlled. For this purpose, we need to be able to detect when a rule applies or when a rule is defeated or ungroundable; as well we need to be able to control the application of a rule based on other antecedent conditions.

First, we introduce, for each rule r in the given program Π, a new special-purpose atom $\mathsf{ap}(n_r)$ to detect the case where a rule's applicability conditions are satisfied. For instance, the rule

$$r_{42} \quad = \quad p \leftarrow q, not\ w$$

is mapped onto the rule

$$\mathsf{ap}(n_{42}) \quad \leftarrow \quad q, not\ w\ . \tag{2}$$

The consequent of the original rule, p, is replaced by the tag $\mathsf{ap}(n_{42})$, just recording the fact that r_{42} is applicable. The addition of

$$p \quad \leftarrow \quad \mathsf{ap}(n_{42}) \tag{3}$$

then 'restores' the effect of the original rule. This mapping separates the applicability of a rule from its actual application.

Second, for detecting when a rule's applicability conditions cannot be satisfied (i.e., the rule will be blocked), we introduce, for each rule r in Π, another new atom $\mathsf{bl}(n_r)$. Unlike the above, however, there are two cases for a rule r not to be applied: it may be that some literal in $body^+(r)$ does not appear in the answer set, or it may be that a literal in $body^-(r)$ is in the answer set. For rule r_{42} we thus get two rules:

$$\mathsf{bl}(n_{42}) \quad \leftarrow \quad not\ q\ , \tag{4}$$
$$\mathsf{bl}(n_{42}) \quad \leftarrow \quad w\ . \tag{5}$$

As made precise in section 4, this tagging technique provides us already with complete information about the applicability status of each rule $r \in \Pi$ with respect to any answer set X of the 'tagged program' Π':

$$\mathsf{ap}(n_r) \in X \text{ iff } \mathsf{bl}(n_r) \notin X\ . \tag{6}$$

(Informally, the 'tagged program' Π' is obtained from Π by treating each rule in Π as described above for r_{42}, that is, by replacing each rule by the appropriate rules corresponding to (2)–(5).) Although this is all relatively straightforward, it can be seen that via these tags we can detect when a rule is or is not applied. We note that a more fine-grained approach is obtainable by distinguishing the two causes of blockage by means of two different tags, like $\mathsf{bl}^+(n)$ and $\mathsf{bl}^-(n)$.

Finally, for controlling the application of rule r, we introduce an atom $\mathsf{ok}(n_r)$ among the body literals of r. Then, clearly the transformed rule can (potentially) be applied only if $\mathsf{ok}(n_r)$ is asserted. More generally, we can combine this with the preceding mapping and so have $\mathsf{ok}(n_r)$ appear in the prerequisite of rules (2), (4), and (5). In our example, we get

$$\begin{aligned}
\mathsf{ap}(n_{42}) \quad &\leftarrow \quad \mathsf{ok}(n_{42}), q, not\ w\ ,\\
\mathsf{bl}(n_{42}) \quad &\leftarrow \quad \mathsf{ok}(n_{42}), not\ q\ ,\\
\mathsf{bl}(n_{42}) \quad &\leftarrow \quad \mathsf{ok}(n_{42}), w\ .
\end{aligned}$$

With this tagging, (6) can be refined so that for each rule $r \in \Pi$ and any answer set X of the 'tagged program" Π'' we have

$$\text{if } \mathsf{ok}(n_r) \in X\ , \text{ then } \mathsf{ap}(n_r) \in X \text{ iff } \mathsf{bl}(n_r) \notin X\ .$$

For preference handling, informally, we conclude $\mathsf{ok}(n_r)$ for rule r just if it is 'ok' with respect to every $<$-greater rule r'. The exact meaning of this process is fixed by the given preference handling strategy. For instance, in the strategy of section 4,

$\mathsf{ok}(n_r)$ is concluded just when each $<$-greater rule r' is known to be blocked or applied. Using tags, a single preference like $r_2 < r_4$, saying that r_4 is preferred to r_2, could thus be encoded directly in the following way (together with the appropriately tagged rules):

$$\begin{array}{rcl} \mathsf{ok}(n_4) & \leftarrow & , \\ \mathsf{ok}(n_2) & \leftarrow & \mathsf{ap}(n_4)\ , \\ \mathsf{ok}(n_2) & \leftarrow & \mathsf{bl}(n_4)\ . \end{array}$$

Similar to the addition of ok-literals to the rules' prerequisites, we can introduce literals of form *not* $\mathsf{ko}(n)$ among the weakly negated body literals of a rule. Now the transformed rule behaves exactly as the original, except that we can defeat (or 'knock out') this rule by asserting $\mathsf{ko}(n)$. For instance, adding *not* $\mathsf{ko}(n_{42})$ to the body of (3) would allow us to assert the applicability of r_{42} without asserting its original consequent, whenever $\mathsf{ko}(n_{42})$ is derivable. This manner of blocking a rule's application has of course appeared earlier in the literature, where ko was most commonly called *ab* (for 'abnormal' (McCarthy, 1986)).

4 Order preserving logic programs

This section elaborates upon a fully prescriptive strategy for preference handling, having its roots in Delgrande and Schaub (1997). The idea of this approach is to select those answer sets of the program that can be generated in an 'order preserving way'. This allows us to enforce the ordering information during the construction of answer sets.

To guarantee that rules are treated in an 'order preserving way' our strategy stipulates that the question of applicability must be settled for higher ranked rules before that of lower ranked rules. In this way, a rule can never be defeated or grounded by lower ranked rules. While a lower ranked rule may of course depend on the presence (or absence) of a literal appearing in the head of a higher ranked rule, the application of lower ranked rules is independent of *the fact* that a higher ranked rule has been applied or found to be blocked. This is important to guarantee the selection among existing answer sets, since otherwise not *all* rules of the original program are considered (cf. Proposition 2 and 5).

Let us illustrate this by means of the following statically ordered program taken from Baader and Hollunder (1993):[4]

$$\begin{array}{rcrcl} r_1 & = & \neg f & \leftarrow & p, not\ f \\ r_2 & = & w & \leftarrow & b, not\ \neg w \\ r_3 & = & f & \leftarrow & w, not\ \neg f \\ r_4 & = & b & \leftarrow & p \\ r_5 & = & p & \leftarrow & \end{array} \qquad r_2 < r_1\ . \tag{7}$$

[4] Letters b, p, f, w stand, as usual, for *birds*, *penguins*, *flies*, and *wings*, respectively. Baader and Hollunder (1993) formulate their approach in default logic (Reiter, 1980).

Program $\Pi_7 = \{r_1, \dots, r_5\}$ has two answer sets: $X_1 = \{p, b, w, \neg f\}$ and $X_2 = \{p, b, w, f\}$. Consider

$$\Pi_7^{X_1} = \left\{ \begin{array}{lcl} \neg f & \leftarrow & p \\ w & \leftarrow & b \\ & & \\ b & \leftarrow & p \\ p & \leftarrow & \end{array} \right\} \quad \text{and} \quad \Pi_7^{X_2} = \left\{ \begin{array}{lcl} & & \\ w & \leftarrow & b \\ f & \leftarrow & w \\ b & \leftarrow & p \\ p & \leftarrow & \end{array} \right\} .$$

This gives rise to the following constructions:

$$\begin{array}{lcl} T^1_{\Pi_7^{X_1}}\emptyset & = & \{p\} \\ T^2_{\Pi_7^{X_1}}\emptyset & = & \{p, b, \neg f\} \\ T^3_{\Pi_7^{X_1}}\emptyset & = & \{p, b, \neg f, w\} \\ & & \end{array} \quad \text{and} \quad \begin{array}{lcl} T^1_{\Pi_7^{X_2}}\emptyset & = & \{p\} \\ T^2_{\Pi_7^{X_2}}\emptyset & = & \{p, b\} \\ T^3_{\Pi_7^{X_2}}\emptyset & = & \{p, b, w\} \\ T^4_{\Pi_7^{X_2}}\emptyset & = & \{p, b, w, f\} . \end{array}$$

Let us now take a closer look at the grounded enumerations associated with X_1 and X_2. The construction of X_1 leaves room for three grounded enumerations:

$$\langle r_5, r_4, r_2, r_1 \rangle, \langle r_5, r_4, r_1, r_2 \rangle, \text{ and } \langle r_5, r_1, r_4, r_2 \rangle . \tag{8}$$

According to our strategy only the last two enumerations are order preserving since they reflect the fact that r_1 is treated before r_2. The construction of X_2 induces a single grounded enumeration

$$\langle r_5, r_4, r_2, r_3 \rangle . \tag{9}$$

Unlike the above, r_1 does not occur in this sequence. So the question arises whether it was blocked in an order preserving way. In fact, once r_5 and r_4 have been applied both r_1 and r_2 are applicable, that is, their prerequisites have been derived and neither of them is defeated at this point. Clearly, the application of r_2 rather than r_1 violates the preference $r_2 < r_1$ (in view of the given strategy). Therefore, only X_1 can be generated in an order preserving way, which makes our strategy select it as the only preferred answer set of $(\Pi_7, <)$.

Let us make this intuition precise for statically ordered programs.

Definition 1
Let $(\Pi, <)$ be a statically ordered program and let X be a consistent answer set of Π.

Then, X is called $<$-*preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of Γ_Π^X such that, for every $i, j \in I$, we have that:

1. $body^+(r_i) \subseteq \{head(r_k) \mid k < i\}$;
2. if $r_i < r_j$, then $j < i$; and
3. if $r_i < r'$ and $r' \in \Pi \setminus \Gamma_\Pi^X$, then
   (a) $body^+(r') \not\subseteq X$ or
   (b) $body^-(r') \cap \{head(r_k) \mid k < i\} \neq \emptyset$.

While Condition 2 guarantees that all generating rules are applied according to the given order, Condition 3 assures that any preferred yet inapplicable rule is either blocked due to the non-derivability of its prerequisites or because it is defeated by higher ranked or unrelated rules.

Condition 1 makes the property of *groundedness* explicit. Although any standard answer set is generated by a grounded sequence of rules, we will see in the following sections that this property is sometimes weakened when preferences are at issue.

For simplicity, our motivation was given in a static setting. The generalisation of order preservation to the dynamic case is discussed in section 4.2.

4.1 Encoding

We now show how the selection of $<$-preserving answer sets can be implemented via a translation of ordered logic programs back into standard programs. Unlike the above, this is accomplished in a fully dynamic setting, as put forward in section 3.

Given an ordered program Π over $\mathscr{L}$, let $\mathscr{L}^+$ be the language obtained from $\mathscr{L}$ by adding, for each $r, r' \in \Pi$, new pairwise distinct propositional atoms $\mathsf{ap}(n_r)$, $\mathsf{bl}(n_r)$, $\mathsf{ok}(n_r)$, and $\mathsf{rdy}(n_r, n_{r'})$.[5] Then, our translation $\mathscr{T}$ maps an ordered program Π over $\mathscr{L}$ into a regular program $\mathscr{T}(\Pi)$ over $\mathscr{L}^+$ in the following way.

Definition 2
Let $\Pi = \{r_1, \dots, r_k\}$ be an ordered logic program over $\mathscr{L}$.

Then, the logic program $\mathscr{T}(\Pi)$ over $\mathscr{L}^+$ is defined as

$$\mathscr{T}(\Pi) = \textstyle\bigcup_{r\in\Pi} \tau(r) \ ,$$

where the set $\tau(r)$ consists of the following rules, for $L^+ \in body^+(r)$, $L^- \in body^-(r)$, and $r', r'' \in \Pi$:

$$\begin{array}{rrcl}
a_1(r): & head(r) & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body(r) \\
b_1(r, L^+): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), not\ L^+ \\
b_2(r, L^-): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), L^- \\
\\
c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \dots, \mathsf{rdy}(n_r, n_{r_k}) \\
c_2(r, r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & not\ (n_r \prec n_{r'}) \\
c_3(r, r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{ap}(n_{r'}) \\
c_4(r, r'): & \mathsf{rdy}(n_r, n_{r'}) & \leftarrow & (n_r \prec n_{r'}), \mathsf{bl}(n_{r'}) \\
\\
t(r, r', r''): & n_r \prec n_{r''} & \leftarrow & n_r \prec n_{r'}, n_{r'} \prec n_{r''} \\
as(r, r'): & \neg(n_{r'} \prec n_r) & \leftarrow & n_r \prec n_{r'} \ .
\end{array}$$

We write $\mathscr{T}(\Pi, <)$ instead of $\mathscr{T}(\Pi')$ whenever we deal with a statically ordered program $\Pi' = \Pi \cup \{(n_r \prec n_{r'}) \leftarrow \ \mid r, r' \in \Pi, r < r'\}$.

The first four rules of $\tau(r)$ express applicability and blocking conditions of the original rules: For each rule $r \in \Pi$, we obtain two rules, $a_1(r)$ and $a_2(r)$, along with $|body^+(r)|$ rules of form $b_1(r, L^+)$ and $|body^-(r)|$ rules of form $b_2(r, L^-)$. A rule r is thus represented by $|body(r)| + 2$ rules in $\tau(r)$.

[5] $\mathsf{rdy}(\cdot, \cdot)$, for 'ready', are auxiliary atoms for $\mathsf{ok}(\cdot)$.

The second group of rules encodes the strategy for handling preferences. The first of these rules, $c_1(r)$, 'quantifies' over the rules in Π, in that we conclude $\mathsf{ok}(n_r)$ just when r is 'ready' to be applied with respect to all the other rules. This is necessary when dealing with dynamic preferences since preferences may vary depending on the corresponding answer set. The three rules $c_2(r,r')$, $c_3(r,r')$, and $c_4(r,r')$ specify the pairwise dependency of rules in view of the given preference ordering: For any pair of rules r, r', we derive $\mathsf{rdy}(n_r,n_{r'})$ whenever $n_r \prec n_{r'}$ fails to hold, or else whenever either $\mathsf{ap}(n_{r'})$ or $\mathsf{bl}(n_{r'})$ is true. This allows us to derive $\mathsf{ok}(n_r)$, indicating that r may potentially be applied whenever we have for all r' with $n_r \prec n_{r'}$ that r' has been applied or cannot be applied. It is important to note that this is only one of many strategies for dealing with preferences: different strategies are obtainable by changing the specification of $\mathsf{ok}(\cdot)$ and $\mathsf{rdy}(\cdot,\cdot)$, as we will see in the subsequent sections.

The last group of rules renders the order information a strict partial order.

Given that occurrences of n_r can be represented by variables, the number of rules in $\mathscr{T}(\Pi)$ is limited by $(|\Pi| \cdot (max_{r\in\Pi}|body(r)| + 2)) + 4 + 2$. A potential bottleneck in this translation is clearly rule $c_1(r)$ whose body may contain $|\Pi|$ literals. In practice, however, this can always be reduced to the number of rules involved in the preference handling (cf. section 6).

As an illustration of our approach, consider the following program, Π_{10}:

$$\begin{array}{lcrcl} r_1 & = & \neg a & \leftarrow & \\ r_2 & = & b & \leftarrow & \neg a, not\ c \\ r_3 & = & c & \leftarrow & not\ b \\ r_4 & = & n_3 \prec n_2 & \leftarrow & not\ d\ , \end{array} \tag{10}$$

where n_i denotes the name of rule r_i $(i = 1,\ldots,4)$. This program has two regular answer sets: $X_1 = \{\neg a, b, n_3 \prec n_2\}$ and $X_2 = \{\neg a, c, n_3 \prec n_2\}$. While $n_3 \prec n_2$ is in both answer sets, it is only respected by X_1, in which r_2 overrides r_3.

In fact, X_1 corresponds to the single answer set obtained from $\mathscr{T}(\Pi_{10})$. To see this, observe that for any $X \subseteq \{head(r) \mid r \in \mathscr{T}(\Pi_{10})\}$, we have $n_i \prec n_j \notin X$ whenever $(i,j) \neq (3,2)$. We thus get for such X and i,j that $\mathsf{rdy}(n_i,n_j) \in T^1_{\mathscr{T}(\Pi_{10})^X}\emptyset$ by (reduced) rules $c_2(r_i,r_j)^+$, and so $\mathsf{ok}(n_i) \in T^2_{\mathscr{T}(\Pi_{10})^X}\emptyset$ via rule $c_1(r_i)^+ = c_1(r_i)$ for $i = 1,2,4$. Analogously, we get that $\mathsf{ap}(n_1), \mathsf{ap}(n_4), \neg a$, and $n_3 \prec n_2$ belong to any answer set of $\mathscr{T}(\Pi_{10})$.

Now consider the following rules from $\mathscr{T}(\Pi_{10})$:

$$\begin{array}{rcrcl} a_2(r_2) & : & \mathsf{ap}(n_2) & \leftarrow & \mathsf{ok}(n_2), \neg a, not\ c \\ b_1(r_2,\neg a) & : & \mathsf{bl}(n_2) & \leftarrow & \mathsf{ok}(n_2), not\ \neg a \\ b_2(r_2,c) & : & \mathsf{bl}(n_2) & \leftarrow & \mathsf{ok}(n_2), c \\[1ex] a_2(r_3) & : & \mathsf{ap}(n_3) & \leftarrow & \mathsf{ok}(n_3), not\ b \\ b_2(r_3,b) & : & \mathsf{bl}(n_3) & \leftarrow & \mathsf{ok}(n_3), b \\[1ex] c_3(r_3,r_2) & : & \mathsf{rdy}(n_3,n_2) & \leftarrow & (n_3 \prec n_2), \mathsf{ap}(n_2) \\ c_4(r_3,r_2) & : & \mathsf{rdy}(n_3,n_2) & \leftarrow & (n_3 \prec n_2), \mathsf{bl}(n_2)\ . \end{array}$$

Given $\mathsf{ok}(n_2)$ and $\neg a$, rule $a_2(r_2)$ leaves us with the choice between $c \notin X$ or $c \in X$.

First, assume $c \notin X$. We get $\mathsf{ap}(n_2)$ from $a_2(r_2)^+ \in \mathscr{T}(\Pi_{10})^X$. Hence, we get b, $\mathsf{rdy}(n_3, n_2)$, and finally $\mathsf{ok}(n_3)$, which results in $\mathsf{bl}(n_3)$ via $b_2(r_3, b)$. Omitting further details, this yields an answer set containing b while excluding c.

Secondly, assume $c \in X$. This eliminates $a_2(r_2)$ when turning $\mathscr{T}(\Pi_{10})$ into $\mathscr{T}(\Pi_{10})^X$. Also, $b_1(r_2, \neg a)$ is defeated since $\neg a$ is derivable. $b_2(r_2, c)$ is inapplicable, since c is only derivable (from $\mathsf{ap}(n_3)$ via $a_1(r_3)$) in the presence of $\mathsf{ok}(n_3)$. But $\mathsf{ok}(n_3)$ is not derivable since neither $\mathsf{ap}(n_2)$ nor $\mathsf{bl}(n_2)$ is derivable. Since this circular situation is unresolvable, there is no answer set containing c.

Whenever one deals exclusively with static preferences, a simplified version of translation $\mathscr{T}$ can be used. Given a statically ordered program $(\Pi, <)$, a static translation $\mathscr{T}'(\Pi, <)$ is obtained from $\mathscr{T}(\Pi, <)$ by

1. replacing $c_1(r)$, $c_2(r, r')$, $c_3(r, r')$, and $c_4(r, r')$ in $\tau(r)$ by

$$\begin{array}{rrcl} c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{s_1}), \ldots, \mathsf{rdy}(n_r, n_{s_k}) \\ c_3(r, s_i): & \mathsf{rdy}(n_r, n_{s_i}) & \leftarrow & \mathsf{ap}(n_{s_i}) \\ c_4(r, s_i): & \mathsf{rdy}(n_r, n_{s_i}) & \leftarrow & \mathsf{bl}(n_{s_i}) \end{array}$$

   for $i = 1, \ldots, k$ where $\{s_1, \ldots, s_k\} = \{s \mid r < s\}$, and
2. deleting $t(r, r', r'')$ and $as(r, r')$ in $\tau(r)$.

Note that the resulting program $\mathscr{T}'(\Pi, <)$ does not contain any preference atoms anymore. Rather the preferences are directly 'woven' into the resulting program in order to enforce order preservation.

4.2 Formal elaboration

Our first result ensures that the dynamically generated preference information enjoys the usual properties of partial orderings. To this end, we define the following relation: for each set X of literals and every $r, r' \in \Pi$, the relation $r <_X r'$ holds iff $n_r \prec n_{r'} \in X$.

Proposition 1
Let Π be an ordered logic program and X a consistent answer set of $\mathscr{T}(\Pi)$. Then, we have:

1. $<_X$ is a strict partial order; and
2. if Π has only static preferences, then $<_X = <_Y$, for any answer set Y of $\mathscr{T}(\Pi)$.

For statically ordered programs, one can show that the addition of preferences can never increase the number of $<$-preserving answer sets.

The next result ensures that we consider *all* rules and that we gather complete knowledge on their applicability status.

Proposition 2
Let Π be an ordered logic program and X a consistent answer set of $\mathscr{T}(\Pi)$. Then, we have for any $r \in \Pi$:

1. $\mathsf{ok}(n_r) \in X$; and
2. $\mathsf{ap}(n_r) \in X$ iff $\mathsf{bl}(n_r) \notin X$.

The following properties shed light on the program induced by translation $\mathscr{T}$; they elaborate upon the logic programming operator $T_{\mathscr{T}(\Pi)^X}$ of a reduct $\mathscr{T}(\Pi)^X$:

Proposition 3

Let Π be an ordered logic program and X a consistent answer set of $\mathscr{T}(\Pi)$.

Let $\Omega = \mathscr{T}(\Pi)^X$. Then, we have for any $r \in \Pi$:

1. if r is not defeated by X, $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$, and $body^+(r) \subseteq T^j_\Omega\emptyset$, then $\mathsf{ap}(n_r) \in T^{\max(i,j)+1}_\Omega\emptyset$;
2. $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$ and $body^+(r) \nsubseteq X$ implies $\mathsf{bl}(n_r) \in T^{i+1}_\Omega\emptyset$;
3. if r is defeated by X and $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$, then $\mathsf{bl}(n_r) \in T^j_\Omega\emptyset$ for some $j > i$;
4. $\mathsf{ok}(n_r) \notin T^i_\Omega\emptyset$ implies $\mathsf{ap}(n_r) \notin T^j_\Omega\emptyset$ and $\mathsf{bl}(n_r) \notin T^k_\Omega\emptyset$ for all $j,k < i+2$.

The next result captures the prescriptive principle underlying the use of ok-literals and their way of guiding the inference process along the preference order. In fact, it shows that their derivation strictly follows the partial order induced by the given preference relation. Clearly, this carries over to the images of the original program rules, as made precise in the subsequent corollary.

Theorem 1

Let Π be an ordered logic program, X a consistent answer set of $\mathscr{T}(\Pi)$, and $\langle r_i \rangle_{i \in I}$ a grounded enumeration of the set $\Gamma^X_{\mathscr{T}(\Pi)}$ of generating rules of X from $\mathscr{T}(\Pi)$.

Then, we have for all $r, r' \in \Pi$:

$$\text{If } r <_X r', \text{ then } j < i \text{ for } r_i = c_1(r) \text{ and } r_j = c_1(r') \,.$$

This result obviously extends to the respective images of rules r and r'.

Corollary 1

Given the same prerequisites as in Theorem 1, we have for all $r, r' \in \Pi$:

$$\text{If } r <_X r', \text{ then } j < i$$

for all r_i equaling $a_2(r)$ or $b_k(r, L)$ for some $k = 1, 2$ and some $L \in body(r)$, and some r_j equaling $a_2(r')$ or $b_{k'}(r', L')$ for some $k' = 1, 2$ and some $L' \in body(r')$.

The last results reflect nicely how the prescriptive strategy is enforced. Whenever a rule r' is preferred to another rule r, one of its images, say $b_2(r', \neg d)$, occurs necessarily before all images of the less preferred rule, e.g. $a_2(r)$. (Note that the existential quantification of r_j is necessary since r' could be blocked in several ways.)

For static preferences, our translation $\mathscr{T}$ amounts to selecting $<$-preserving answer sets of the underlying (unordered) program, as given by Definition 1.

Theorem 2

Let $(\Pi, <)$ be a statically ordered logic program and let X be a consistent set of literals.

Then, X is a $<$-preserving answer set of Π iff $X = Y \cap \mathscr{L}_{\mathscr{A}}$ for some answer set Y of $\mathscr{T}(\Pi, <)$.

This result provides semantics for statically ordered programs; it provides an exact correspondence between answer sets issued by our translation and regular answer sets of the original program.

Note that inconsistent answer sets are not necessarily reproduced by $\mathscr{T}$. To see this, consider $\Pi = \{p \leftarrow, \neg p \leftarrow\}$. While Π has an inconsistent answer set, $\mathscr{T}(\Pi)$ has no answer set. This is due to the fact that rules like $c_2(r,r')$ are removed from $\mathscr{T}(\Pi)^{\mathscr{L}}$. In such a case, there is then no way to derive 'ok'-literals via $c_1(r)$. On the other hand, one can show that $\mathscr{T}(\Pi,<)$ has an inconsistent answer set only if Π has an inconsistent answer set, for any statically ordered program $(\Pi,<)$.[6]

We obtain the following corollary to Theorem 2, demonstrating that our strategy implements a selection function among the standard answer sets of the underlying program.

Corollary 2
Let $(\Pi,<)$ be a statically ordered logic program and X a set of literals.

If $X = Y \cap \mathscr{L}_{\mathscr{A}}$ for some answer set Y of $\mathscr{T}(\Pi,<)$, then X is an answer set of Π.

Note that the last two results do not directly carry over to the general (dynamic) case, since a dynamic setting admits no way of selecting answer sets in an undifferentiated way due to the lack of a uniform preference relation.

The key difference between static and dynamic preference handling boils down to the availability of preferences. While the full preference information is available right from the start in the static case, it develops with the formation of answer sets in the dynamic case. Moreover, from the viewpoint of rule application, a dynamic setting not only necessitates that the question of applicability has been settled for all higher ranked rules before a rule is considered for application but also that one knows about all higher ranked rules at this point. That is, before considering a rule r for application, all preferences of form $n_r \prec n_{r'}$ must have been derived. This adds a new requirement to the concept of order preservation in the dynamic case, expressed by Condition 2 below. A major consequence of this additional requirement is that we cannot restrict our attention to the generating rules of an answer set anymore, as in the static case, but that we have to take all rules of the program into account, no matter whether they eventually apply or not.

Preference information is now drawn from the considered answer sets. We only have to make sure that the resulting preferences amount to a strict partial order. For this purpose, let $\Pi^\star$ denote the program obtained from Π by adding transitivity and antisymmetry rules, that is $\Pi^\star = \Pi \cup \{t(r,r',r''), as(r,r') \mid r,r',r'' \in \Pi\}$.

Taking all this into account, we arrive at the following concept of order preservation for dynamic preferences.

Definition 3
Let Π be an ordered program and let X be a consistent answer set of $\Pi^\star$.

Then, X is called $<_X$*-preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of $\Pi^\star$ such that, for every $i,j \in I$, we have that:

[6] Observe that the static translation $\mathscr{T}'(\Pi,<)$ preserves inconsistent answer sets of Π, however.

1. if $r_i <_X r_j$, then $j < i$;
2. if $r_i <_X r_j$, then there is some $r_k \in \Gamma^X_{\Pi^\star}$ such that
   (a) $k < i$ and
   (b) $head(r_k) = (n_{r_i} \prec n_{r_j})$;
3. if $r_i \in \Gamma^X_{\Pi^\star}$, then $body^+(r_i) \subseteq \{head(r_k) \mid r_k \in \Gamma^X_{\Pi^\star}, k < i\}$; and
4. if $r_i \in \Pi^\star \setminus \Gamma^X_{\Pi^\star}$, then
   (a) $body^+(r_i) \not\subseteq X$ or
   (b) $body^-(r_i) \cap \{head(r_k) \mid r_k \in \Gamma^X_{\Pi^\star}, k < i\} \neq \emptyset$.

Despite the fact that this conception of order preservation takes all rules into account, as opposed to generating rules only, it remains a generalisation of its static counterpart given in Definition 1. In fact, above Conditions 3, 1, and 4 can be seen as extension of Conditions 1, 2, and 3 in Definition 1 from sets of generating rules to entire programs.

For illustration, consider answer set $X = \{\neg a, b, n_3 \prec n_2, \neg(n_3 \prec n_2)\}$ of $\Pi^\star_{10}$; this answer set is $<_X$-preserving. To see this, consider the following enumeration of Π_{10} (omitting rules in $\Pi^\star_{10} \setminus \Pi_{10}$ for simplicity; we write $\bar{r}$ to indicate that $r \notin \Gamma^X_{\Pi^\star_{10}}$):

$$\langle r_1, r_4, r_2, \bar{r}_3 \rangle \ . \tag{11}$$

The only preference $\bar{r}_3 <_X r_2$ induces, through Condition 1, that r_2 occurs before $\bar{r}_3$ and that r_4 occurs before $\bar{r}_3$ via Condition 2. Furthermore, due to Condition 3, r_1 occurs before r_2, while Condition 4b necessitates (also) that r_2 occurs before $\bar{r}_3$. Intuitively, the enumeration in (11) corresponds to that of the images of Π_{10} generating the standard answer set of $\mathscr{T}(\Pi^\star_{10})$, viz.

$$\langle \dots, a_2(r_1), \dots, a_2(r_4), \dots, a_2(r_2), \dots, b_2(r_3, b), \dots \rangle \ .$$

As discussed above, the difference to the static case manifests itself in Condition 2, which reflects the fact that all relevant preference information must be derived before a rule is considered for application. This also mirrors the strong commitment of the strategy to a successive development of preferred answer sets in accord with groundedness. This is different from the approach discussed in section 5 that drops the groundedness requirement.

For illustrating Condition 2, consider programs $\Pi_{12a} = \{r_1, r_2, r_{3a}\}$ and $\Pi_{12b} = \{r_1, r_2, r_{3b}\}$, where

$$\begin{array}{lcrcl@{\qquad}lcrcl}
r_1 & = & a & \leftarrow & not\ \neg a & r_1 & = & a & \leftarrow & not\ \neg a \\
r_2 & = & b & \leftarrow & not\ \neg b & r_2 & = & b & \leftarrow & not\ \neg b \\
r_{3a} & = & n_1 \prec n_2 & \leftarrow & a & r_{3b} & = & n_1 \prec n_2 & \leftarrow & b\ .
\end{array} \tag{12}$$

Both programs have the same standard answer set $X_{12} = \{a, b, n_1 \prec n_2\}$. However, this answer set is only $<_{X_{12}}$-preserving for Π_{12b} but not for Π_{12a}. This is because Π_{12b} allows to derive the preference atom $n_1 \prec n_2$ before the lower ranked rule, r_1, is considered for application, while this is impossible with Π_{12a}. That is, while there is an enumeration of Π_{12b} satisfying both Condition 2 and 3 there is no such enumeration of Π_{12a}. A dynamic extension of the approach dropping Condition 2 is described in section 5.5.

We have the following result demonstrating soundness and completeness of translation $\mathscr{T}$ with respect to $<_X$-preserving answer sets.

Theorem 3

Let Π be an ordered logic program and let X be a consistent set of literals.

Then, X is a $<_X$-preserving answer set of $\Pi^\star$ iff $X = Y \cap \mathscr{L}$ for some answer set Y of $\mathscr{T}(\Pi)$.

As above, we obtain the following corollary.

Corollary 3

Let Π be an ordered logic program and let X be a set of literals.

If $X = Y \cap \mathscr{L}$ for some answer set Y of $\mathscr{T}(\Pi)$, then X is an answer set of $\Pi^\star$.

Also, if no preference information is present, transformation $\mathscr{T}$ amounts to standard answer set semantics. Moreover, the notions of statically ordered and (dynamically) ordered programs coincide in this case. The following result expresses this property in terms of order preserving answer sets.

Theorem 4

Let Π be a logic program over $\mathscr{L}_\mathscr{A}$ and let X be a consistent set of literals.

Then, the following statements are equivalent:

1. X is a $<_X$-preserving answer set of the dynamically ordered program Π and $<_X = \emptyset$.
2. X is a $<$-preserving answer set of the statically ordered program $(\Pi, <)$ and $< = \emptyset$.
3. X is a regular answer set of logic program Π.

From these properties, we immediately obtain the following result for our encoding $\mathscr{T}$, showing that preference-free programs yield regular answer sets.

Corollary 4

Let Π be a logic program over $\mathscr{L}_\mathscr{A}$ and let X be a consistent set of literals.

Then, X is an answer set of Π iff $X = Y \cap \mathscr{L}$ for some answer set Y of $\mathscr{T}(\Pi)$.

Brewka and Eiter (1999) have suggested two properties, termed *Principle I* and *Principle II*, which they argue any defeasible rule system handling preferences should satisfy. The next result shows that order preserving answer sets[7] enjoy these properties. However, since the original formulation of Principle I and II is rather generic – motivated by the aim to cover as many different approaches as possible – we must instantiate them in terms of the current framework. It turns out that Principle I is only suitable for statically ordered programs, whilst Principle II admits two guises, one for statically ordered programs, and another one for (dynamically) ordered programs.

[7] In the sense of Definitions 1 and 3.

Principles I and II, formulated for our setting, are as follows:

Principle I. Let $(\Pi, <)$ be a statically ordered logic program and let X_1 and X_2 be two (regular) answer sets of Π generated by $\Gamma_\Pi^{X_1} = R \cup \{r_1\}$ and $\Gamma_\Pi^{X_2} = R \cup \{r_2\}$, respectively, where $r_1, r_2 \notin R$.
If $r_1 < r_2$, then X_1 is not a $<$-preserving answer set of Π.

Principle II-S (Static Version). Let $(\Pi, <)$ be a statically ordered logic program and let X be a $<$-preserving answer set of Π. Let r be a rule where $body^+(r) \nsubseteq X$ and let $(\Pi \cup \{r\}, <')$ be a statically ordered logic program where $< = <' \cap (\Pi \times \Pi)$.
Then, X is a $<'$-preserving answer set of $\Pi \cup \{r\}$.

Principle II-D (Dynamic Version). Let Π be a (dynamically) ordered logic program and let X be a $<_X$-preserving answer set of $\Pi^\star$. Let r be a rule such that $body^+(r) \nsubseteq X$.
Then, X is a $<_X$-preserving answer set of $\Pi^\star \cup \{r\}$.

Theorem 5
We have the following properties.

1. Order preserving answer sets in the sense of Definition 1 satisfy Principles I and II-S.
2. Order preserving answer sets in the sense of Definition 3 satisfy Principle II-D.

Finally, we mention some properties concerning the computational complexity of order preserving answer sets. Since transformation $\mathscr{T}$ is clearly polynomial in the size of ordered logic programs, in virtue of Theorems 2, 3 and 4, it follows in a straightforward way that the complexity of answer set semantics under order preservation coincides with the complexity of standard answer set semantics. We note the following results.

Theorem 6
Order preserving answer sets enjoy the following properties:

1. Given an ordered program Π, checking whether Π has an answer set X which is $<_X$-preserving is NP-complete.
2. Given an ordered program Π and some literal L, checking whether Π has an answer set X which is $<_X$-preserving and which contains L is NP-complete.
3. Given an ordered program Π and some literal L, checking whether L is contained in any answer set X which is $<_X$-preserving is coNP-complete.

4.3 Variations

A strategy similar to the one described above has been advocated in Wang *et al.* (2000). There, fixed-point definitions are used to characterise preferred answer sets (of statically ordered programs). It turns out that this preferred answer set semantics can be implemented by a slight modification of the translation given in Definition 2.

Definition 4 (Schaub and Wang, 2001a)
Given the same prerequisites as in Definition 2, the logic program $\mathscr{W}(\Pi)$ over $\mathscr{L}^+$ is defined as

$$\mathscr{W}(\Pi) = \bigcup_{r\in\Pi}\tau(r) \cup \{c_5(r,r') \mid r,r' \in \Pi\},$$

where

$$c_5(r,r') : \quad \mathsf{rdy}(n_r, n_{r'}) \quad \leftarrow \quad (n_r \prec n_{r'}), head(r') \ .$$

The purpose of $c_5(r,r')$ is to eliminate rules from the preference handling process once their head has been derived. A corresponding soundness and completeness result can be found in Schaub and Wang (2001a).

In terms of order preservation, this amounts to weakening the integration of preferences and groundedness:

Definition 5 (Schaub and Wang, 2001a)
Let $(\Pi, <)$ be a statically ordered program and let X be a consistent answer set of Π.

Then, X is called $<^{\mathrm{WZL}}$*-preserving*, if there exists an enumeration $\langle r_i\rangle_{i\in I}$ of Γ_Π^X such that for every $i, j \in I$ we have that:

1. (a) $body^+(r_i) \subseteq \{head(r_j) \mid j < i\}$ or
   (b) $head(r_i) \in \{head(r_j) \mid j < i\}$;
2. if $r_i < r_j$, then $j < i$; and
3. if $r_i < r'$ and $r' \in \Pi \setminus \Gamma_\Pi^X$, then
   (a) $body^+(r') \nsubseteq X$ or
   (b) $body^-(r') \cap \{head(r_j) \mid j < i\} \neq \emptyset$ or
   (c) $head(r') \in \{head(r_j) \mid j < i\}$.

The primary difference of this concept of order preservation to the original one is clearly the weaker notion of groundedness. This involves the rules in Γ_Π^X (via Condition 1b) as well as those in $\Pi \setminus \Gamma_\Pi^X$ (via Condition 3c). The rest of the definition is the same as in Definition 1. We refer to Schaub and Wang (2001a) for formal details.

Informally, the difference between both strategies can be explained by means of the following program $\Pi_{13} = \{r_1, r_2, r_3\}$, where

$$\begin{array}{rclcl} r_1 & = & a & \leftarrow & not\ \neg a \\ r_2 & = & b & \leftarrow & a, not\ \neg b \\ r_3 & = & b & \leftarrow & \end{array} \qquad \text{and} \qquad r_1 < r_2\ . \tag{13}$$

Clearly, Π_{13} has a single standard answer set, $\{a, b\}$. This answer set cannot be generated by an enumeration of Π_{13} that preserves $r_1 < r_2$ in the sense of Definition 1. This is different after adding Condition 1b. In accord with the approach in Wang *et al.* (2000), the modified concept of order preservation accepts the enumeration $\langle r_3, r_2, r_1\rangle$.

All of the above strategies enforce a selection among the standard answer sets of the underlying program. That is, no new answer sets appear; neither are existing ones modified. While the former seems uncontroversial, the latter should not be taken for granted, as exemplified by a 'winner-takes-all' strategy. In such an approach, one wants to apply the highest ranked default, if possible, and only the highest ranked default. This can be accomplished by removing rule

$$c_3(r, r') : \ \mathsf{rdy}(n_r, n_{r'}) \ \leftarrow \ (n_r \prec n_{r'}), \mathsf{ap}(n_{r'}) \tag{14}$$

from Definition 2. Then, once a higher ranked rule, such as r', is applied, none of the lower ranked rules, like r, are considered for application anymore because there is no way to derive the indispensable 'ok-literal', $\mathsf{ok}(n_r)$. A similar strategy is developed in Gelfond and Son (1997) (cf. section 7).

A particular instance of a 'winner-takes-all' strategy is inheritance of default properties. In this case, the ordering on rules reflects a relation of *specificity* among the (default) rules prerequisites. Informally, for adjudicating among conflicting defaults, one determines the most specific (with respect to rule antecedents) defaults as candidates for application. Consider for example defaults concerning primary means of locomotion: "animals normally walk", "birds normally fly", "penguins normally swim". This can be expressed using (default) rules as follows:

$$\begin{array}{rcrcl} r_1 & = & w & \leftarrow & a, not\ \neg w \\ r_2 & = & f & \leftarrow & b, not\ \neg f \\ r_3 & = & s & \leftarrow & p, not\ \neg s \end{array} \qquad \begin{array}{rcrcl} r_a & = & a & \leftarrow & b \\ r_b & = & b & \leftarrow & p \\ \multicolumn{5}{l}{r_1 < r_2 < r_3\ .} \end{array} \tag{15}$$

If we learn that some object is a penguin, viz. $p \leftarrow$, (and so a bird and animal via r_a and r_b), then we would want to apply the highest ranked default, if possible, and only the highest ranked default. Significantly, if the penguins-swim default r_3 is blocked (say, the penguin in question has a fear of water, viz. $\neg s \leftarrow$) we *don't* try to apply the next default to see if it might fly.

So, given a chain of rules expressing default properties $r_1 < r_2 < \ldots < r_m$, the elimination of (14) in Definition 2 portrays the following comportment: We apply r_i, if possible, where r_i is the $<$-maximum default such that for every default r_j, $j = i+1, \ldots, m$, the prerequisite of r_j is not known to be true. Otherwise, no default in the chain is applicable. As mentioned above, the elimination of (14) allows lower ranked default rules to be applied only in case higher ranked rules are blocked because their prerequisite is not derivable. Otherwise, the propagation of $\mathsf{ok}(\cdot)$-atoms is interrupted so that no defaults below r_i are considered.

5 Brewka and Eiter's approach to preference

We now turn to a strategy for preference handling proposed in Brewka and Eiter (1998; 1999). Unlike above, this approach is not fully prescriptive in the sense that it does not enforce the ordering information during the construction of an answer set. Rather it relies on the existence of a regular answer set of the underlying non-ordered program, whose order preservation is then verified in a separate test (see Definition 10).

5.1 Original Definition

To begin with, we describe the approach to dealing with ordered logic programs, as introduced in Brewka and Eiter (1999). First, Brewka and Eiter deal with statically ordered logic programs only. Also, partially ordered programs are reduced to totally ordered ones:[8] A *fully ordered logic program* is a statically ordered logic program $(\Pi, \ll)$ where $\ll$ is a total ordering. The case of arbitrarily ordered programs is reduced to this restricted case in the following way.

Definition 6
Let $(\Pi, <)$ be a statically ordered logic program and let X be a set of literals.

Then, X is a BE-*preferred answer set* of $(\Pi, <)$ iff X is a BE-preferred answer set of some fully ordered logic program $(\Pi, \ll)$ such that $< \subseteq \ll$.

The construction of BE-preferred answer sets relies on an operator, defined for prerequisite-free programs.

Definition 7
Let $(\Pi, \ll)$ be a fully ordered prerequisite-free logic program, let $\langle r_i \rangle_{i \in I}$ be the enumeration of Π according to the ordering $\ll$, and let X be a set of literals.

Then,[9] $C(X)$ is the smallest logically closed set of literals containing $\bigcup_{i \in I} X_i$, where $X_j = \emptyset$ for $j \notin I$ and

$$X_i = \begin{cases} X_{i-1} & \text{if } body^-(r_i) \cap X_{i-1} \neq \emptyset; \\ X_{i-1} & \text{if } head(r_i) \in X \text{ and } body^-(r_i) \cap X \neq \emptyset; \\ X_{i-1} \cup \{head(r_i)\} & \text{otherwise.} \end{cases}$$

This construction is unique insofar that for any fully ordered prerequisite-free program $(\Pi, \ll)$, there is at most one (standard) answer set X of Π such that $C(X) = X$ (cf. Lemma 4.1 in Brewka and Eiter (1999)). Accordingly, this set is used to define a BE-preferred answer set for *prerequisite-free logic programs*, if it exists:

Definition 8
Let $(\Pi, \ll)$ be a fully ordered prerequisite-free logic program and let X be a set of literals.

Then, X is the BE-preferred answer set of $(\Pi, \ll)$ iff $C(X) = X$.

The second condition in Definition 7 amounts to eliminating from the above construction all rules whose heads are in X but which are defeated by X. This can be illustrated by the following logic program, adapted from Brewka and Eiter (2000):

$$\begin{array}{rcrcl} r_1 & = & a & \leftarrow & not\ b \\ r_2 & = & \neg a & \leftarrow & not\ a \\ r_3 & = & a & \leftarrow & not\ \neg a \\ r_4 & = & b & \leftarrow & not\ \neg b \end{array} \quad \text{with} \quad \{r_j < r_i \mid i < j\}\,. \tag{16}$$

[8] While Brewka and Eiter (1999) deal with potentially infinite well-orderings, we limit ourselves here to finite programs.
[9] Note that C is implicitly parameterised with $(\Pi, \ll)$.

Program $\Pi_{16} = \{r_1, \dots, r_4\}$ has two answer sets, $\{a, b\}$ and $\{\neg a, b\}$. The application of operator C relies on sequence $\langle r_1, r_2, r_3, r_4 \rangle$. For illustration, consider the process induced by $C(\{a, b\})$, with and without the second condition:

$$\begin{array}{llll} X_1 = \{\} & X_2 = \{\neg a\} & X_3 = \{\neg a\} & X_4 = \{\neg a, b\} \\ X_1' = \{a\} & X_2' = \{a\} & X_3' = \{a\} & X_4' = \{a, b\} \,. \end{array}$$

Thus, without the second condition, $\{a, b\}$ would be a preferred answer set. However, Brewka and Eiter (1999; 2000) argue that such an answer set does not preserve priorities because r_2 is defeated in $\{a, b\}$ by applying a rule which is less preferred than r_2, namely r_3. The above program has therefore no BE-preferred answer set.

The next definition accounts for the general case by reducing it to the prerequisite-free one. For checking whether a given regular answer set X is BE-preferred, Brewka and Eiter evaluate the prerequisites of the rules with respect to the answer set X.

Definition 9

Let $(\Pi, \ll)$ be a fully ordered logic program and X a set of literals.

The logic program $(\Pi_X, \ll_X)$ is obtained from $(\Pi, \ll)$ as follows:

1. $\Pi_X = \{r^- \mid r \in \Pi \text{ and } body^+(r) \subseteq X\}$; and
2. for any $r_1', r_2' \in \Pi_X$, $r_1' \ll_X r_2'$ iff $r_1 \ll r_2$ where $r_i = \max_{\ll}\{r \in \Pi \mid r^- = r_i'\}$.

In other words, Π_X is obtained from Π by first eliminating every rule $r \in \Pi$ such that $body^+(r) \not\subseteq X$, and then substituting all remaining rules r by $r^- = head(r) \leftarrow body^-(r)$. This results in a prerequisite-free logic program.

BE-preferred answer sets are then defined as follows.

Definition 10

Let $(\Pi, \ll)$ be a fully ordered logic program and X a set of literals.

Then, X is a BE-preferred answer set of $(\Pi, \ll)$, if

1. X is a (standard) answer set of Π, and
2. X is a BE-preferred answer set of $(\Pi_X, \ll_X)$.

For illustration, consider Example 5.1 of Brewka and Eiter (1999):

$$\begin{array}{rclcl} r_1 & = & b & \leftarrow & a, not\ \neg b \\ r_2 & = & \neg b & \leftarrow & not\ b \\ r_3 & = & a & \leftarrow & not\ \neg a \end{array} \qquad \text{with} \qquad \{r_j < r_i \mid i < j\}\,. \tag{17}$$

Program $\Pi_{17} = \{r_1, r_2, r_3\}$ has two standard answer sets: $X_1 = \{a, b\}$ and $X_2 = \{a, \neg b\}$. $(\Pi_{17})_{X_1}$ turns r_1 into $b \leftarrow not\ \neg b$ while leaving r_2 and r_3 unaffected. Also, we obtain that $C(X_1) = X_1$, that is, X_1 is a BE-preferred answer set. In contrast to this, X_2 is not BE-preferred. While $(\Pi_{17})_{X_2} = (\Pi_{17})_{X_1}$, we get $C(X_2) = X_1 \neq X_2$. That is, $C(X_2)$ reproduces X_1 rather than X_2.

5.2 *Order preservation*

Before giving an encoding for the approach of Brewka and Eiter, we would like to provide some insight into its semantics and its relation to the previous strategy in terms of the notion of order preservation. First, the approach is equivalent to the strategy developed in section 4 on normal prerequisite-free default theories (Delgrande and Schaub, 2000a). Interestingly, this result does not extend to either normal or to prerequisite-free theories. While the former difference is caused by the second condition in Definition 7, the latter is due to the different attitude towards groundedness (see below). Despite these differences it turns out that every order preserving extension in the sense of Definition 1 is also obtained in Brewka and Eiter's approach but not vice versa (cf. (Delgrande and Schaub, 2000a)).

To see the latter, consider the example in (7). Among the two answer sets $X_1 = \{p, b, w, \neg f\}$ and $X_2 = \{p, b, w, f\}$ of Π_7, only X_1 is order preserving in the sense of Definition 1. In contrast to this, both answer sets are BE-preferred. This is because groundedness is not at issue when verifying order preservation in the approach of Brewka and Eiter due to the removal of prerequisites in Definition 9. This can be made precise by means of a corresponding notion of order preservation, taken from Schaub and Wang (2001a):[10]

Definition 11
Let $(\Pi, <)$ be a statically ordered program and let X be a consistent answer set of Π.

Then, X is called $<^{\mathrm{BE}}$*-preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of Γ_Π^X such that, for every $i, j \in I$, we have that:

1. if $r_i < r_j$, then $j < i$; and
2. if $r_i < r'$ and $r' \in \Pi \setminus \Gamma_\Pi^X$, then
   (a) $body^+(r') \not\subseteq X$ or
   (b) $body^-(r') \cap \{head(r_j) \mid j < i\} \neq \emptyset$ or
   (c) $head(r') \in X$.

This criterion differs from the one in Definition 1 in the two (aforementioned) respects: First, it drops the requirement of a *grounded* enumeration and, second, it adds the second condition from Definition 7. Note that any $<^{\mathrm{BE}}$-preserving answer set is still generated by some grounded enumeration of rules; it is just that this property is now separated from the preference handling process.

Although the only grounded sequence given in (9) is still not $<^{\mathrm{BE}}$-preserving, there are now other $<^{\mathrm{BE}}$-preserving enumerations supporting the second answer set X_2, e.g. $\langle r_5, r_4, r_3, r_2 \rangle$. None of these enumerations enjoys groundedness. Rather, all of them share the appearance of r_3 before r_2 although the application of r_3 relies on that of r_2. The reversal of the groundedness relation between r_2 and r_3 is however essential for defeating r_1 (before r_2 is considered for application). Otherwise, there is no way to satisfy Condition 2b.

The discussion of Example (7) has already shed light on the difference between the strictly prescriptive strategy discussed in the previous section and Brewka and Eiter's more descriptive approach. The more descriptive attitude of their approach is nicely

[10] An alternative characterisation of BE-preferred answer sets for fully ordered logic programs, originally given in Proposition 5.1 in Brewka and Eiter (1999), is repeated in Theorem 9.

illustrated by the example in (17). Without dropping the requirement of deriving the prerequisite of the most preferred rule r_1, there is no way of justifying an answer set containing b. Such an approach becomes reasonable if we abandon the strictly prescriptive view of section 4 and adopt a more descriptive one by decomposing the construction of a preferred answer set into a respective guess and check step (as actually reflected by Definition 9). That is, once we know that there is an answer set containing certain formulas, we may rely on it when dealing with preferences. The descriptive flavour of this approach stems from the availability of an existing answer set of the underlying program when dealing with preference information. For instance, given the regular answer set $X_1 = \{a, b\}$ we know that a and b are derivable in a grounded fashion. We may then rely on this information when taking preferences into account, although the order underlying the construction of the regular answer set may be incompatible with the one imposed by the preference information.

Observe that program Π_{17} has no $<$-preserving answer set. Intuitively, this can be explained by the observation that for the highest ranked default r_1, neither applicability nor blockage can be asserted: Either of these properties relies on the applicability of less ranked defaults, effectively resulting in a circular situation destroying any possible $<$-preserving answer set.

5.3 Encoding

Although Brewka and Eiter's approach was originally defined only for the static case, we provide in this section an encoding for the more general dynamic case.

We have seen above that the underlying preference handling strategy drops groundedness, an indispensable property of standard answer sets. As a consequence, our encoding cannot keep the strictly prescriptive nature, as the one given in section 4. Rather, we have to decouple the generation of standard answer sets from the actual preference handling process. Given a statically ordered program $(\Pi, <)$, the idea is thus to take the original program Π and to extend it by a set of rules $\mathscr{T}'(\Pi, <)$ that ensures that the preferences in $<$ are preserved (without necessitating groundedness). In fact, $\mathscr{T}'(\Pi, <)$ provides an image of Π, similar to that of section 4, except that the rules in $\mathscr{T}'(\Pi, <)$ are partially disconnected from those in Π by means of an extended language. All this is made precise in the sequel.

Given a program Π over language $\mathscr{L}$, we assume a disjoint language $\mathscr{L}'$ containing literals L' for each L in $\mathscr{L}$. Likewise, rule r' results from r by replacing each literal L in r by L'. Then, in analogy to section 4.1, we map ordered programs over some language $\mathscr{L}$ onto standard programs in the language $\mathscr{L}^\circ$ obtained by extending $\mathscr{L} \cup \mathscr{L}'$ by new atoms $(n_r \prec n_s)$, $\mathsf{ok}(n_r)$, $\mathsf{rdy}(n_r, n_s)$, $\mathsf{bl}(n_r)$, and $\mathsf{ap}(n_r)$, for each r, s in Π.

Definition 12
Let $\Pi = \{r_1, \dots, r_k\}$ be an ordered logic program over $\mathscr{L}$.

Then, the logic program $\mathscr{U}(\Pi)$ over $\mathscr{L}^\circ$ is defined as

$$\mathscr{U}(\Pi) = \Pi \cup \textstyle\bigcup_{r \in \Pi} \tau(r) \ ,$$

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

$$\begin{array}{rrcl}
a_1(r): & head(r') & \leftarrow & \mathsf{ap}(n_r) \\
a_2(r): & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body(r), not\ body^-(r') \\
b_1(r,L): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), not\ L, not\ L' \\
b_2(r,K): & \mathsf{bl}(n_r) & \leftarrow & \mathsf{ok}(n_r), K, K' \\
\\
c_1(r): & \mathsf{ok}(n_r) & \leftarrow & \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \\
c_2(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & not\ (n_r \prec n_s) \\
c_3(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & (n_r \prec n_s), \mathsf{ap}(n_s) \\
c_4(r,s): & \mathsf{rdy}(n_r, n_s) & \leftarrow & (n_r \prec n_s), \mathsf{bl}(n_s) \\
c_5(r,s,J): & \mathsf{rdy}(n_r, n_s) & \leftarrow & head(s), J \\
\\
d(r): & & \leftarrow & not\ \mathsf{ok}(n_r) \\
\\
t(r,s,t): & n_r \prec n_t & \leftarrow & n_r \prec n_s, n_s \prec n_t \\
as(r,s): & \neg(n_s \prec n_r) & \leftarrow & n_r \prec n_s\ .
\end{array}$$

First, it is important to note that the original program Π is contained in the image $\mathscr{U}(\Pi)$ of the translation. As we show in Proposition 4, this allows us to construct (standard) answer sets of Π within answer sets of $\mathscr{U}(\Pi)$. Such an answer set can be seen as the *guess* in a guess-and-check approach; it corresponds to Condition 1 in Definition 10.

The corresponding *check*, viz. Condition 2 in Definition 10, is accomplished by the remaining rules in $\tau(r)$. Before entering into details, let us stress the principal differences among these rules and those used in translation $\mathscr{T}$ in section 4:

1. The first group of rules is 'synchronised' with the original rules in Π. That is, except for the literals in $body^+(a_2(r))$, all primed body literals are doubled in the original language.
2. As with the encoding in section 4.3, the second group of rules accommodates the modified strategy by adding a fifth rule.
3. Integrity constraint $d(r)$ is added for ruling out unsuccessful candidate answer sets of Π.

As before, the first group of rules of $\tau(r)$ expresses applicability and blocking conditions. The rules of form $a_i(r), i = 1, 2$, aim at rebuilding the guessed answer set within $\mathscr{L}'$. They form in $\mathscr{L}'$ the prerequisite-free counterpart of the original program. In fact, the prerequisite of $a_2(r)$ refers via $body^+(r) \subseteq body(r)$ to the guessed extension in $\mathscr{L}$; no formula in $\mathscr{L}'$ must be derived for applying $a_2(r)$. This accounts for the elimination of prerequisites in Condition 1 of Definition 9. The elimination of rules whose prerequisites are not derivable is accomplished by rules of form $b_1(r, L)$. Rules of form $b_2(r, L)$ guarantee that defaults are only defeatable by rules with higher priority. In fact, it is K' that must be derivable in such a way only.

The application of rules according to the given preference information is enforced by the second group of rules. In addition to the four rules used in Definition 2, an atom like $\mathsf{rdy}(n_r, n_s)$ is now also derivable if the head of s is true although s itself is defeated (both relative to the candidate answer set of Π in $\mathscr{L}$). This treatment of rules like s through $c_5(r,s,J)$ amounts to their elimination from the preference handling process, as originated by the second condition in Definition 7. As before, the whole group of 'rdy'-rules allows us to derive $\mathsf{ok}(n_r)$, indicating that r may potentially be applied, whenever we have for all s with $r < s$ that s has been applied or cannot be applied, or s has already been eliminated from the preference handling process.

Lastly, $d(r)$ rules out unsuccessful attempts in rebuilding the answer set from $\mathscr{L}$ within $\mathscr{L}'$ according to the given preference information. In this way, we eliminate all answer sets that do not respect preferences.

For illustration, reconsider program Π_{17}:

$$\begin{array}{rcrcl} r_1 & = & b & \leftarrow & a, not\ \neg b \\ r_2 & = & \neg b & \leftarrow & not\ b \\ r_3 & = & a & \leftarrow & not\ \neg a \end{array}$$

with r_1 being most preferred, then r_2, and finally r_3. We get, among others:

$$\begin{array}{lcl lcl lcl}
b' & \leftarrow & \mathsf{ap}(n_1) & \neg b' & \leftarrow & \mathsf{ap}(n_2) & a' & \leftarrow & \mathsf{ap}(n_3) \\
\mathsf{ap}(n_1) & \leftarrow & \mathsf{ok}(n_1), & \mathsf{ap}(n_2) & \leftarrow & \mathsf{ok}(n_2), & \mathsf{ap}(n_3) & \leftarrow & \mathsf{ok}(n_3), \\
 & & a, & & & & & & \\
 & & not\ \neg b, & & & not\ b, & & & not\ \neg a, \\
 & & not\ \neg b' & & & not\ b' & & & not\ \neg a' \\
\mathsf{bl}(n_1) & \leftarrow & \mathsf{ok}(n_1), & & & & & & \\
 & & not\ a, & & & & & & \\
 & & not\ a' & & & & & & \\
\mathsf{bl}(n_1) & \leftarrow & \mathsf{ok}(n_1), & \mathsf{bl}(n_2) & \leftarrow & \mathsf{ok}(n_2), & \mathsf{bl}(n_3) & \leftarrow & \mathsf{ok}(n_3), \\
 & & \neg b, \neg b' & & & b, b' & & & \neg a, \neg a'.
\end{array}$$

First, suppose there is an answer set of $\mathscr{U}(\Pi_{17})$ containing the standard answer set $X_2 = \{a, \neg b\}$ of Π_{17}. Clearly, r_2 and r_3 would contribute to this answer set, since they also belong to $\mathscr{U}(\Pi_{17})$. 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 *not* a. Therefore, we do not obtain $\mathsf{ok}(n_2)$. This satisfies integrity constraint $d(r_2)$, which destroys the putative answer set at hand. Hence, as desired, $\mathscr{U}(\Pi_{17})$ has no answer set including X_2.

For a complement, consider the $<^{\mathrm{BE}}$-preserving answer set $X_1 = \{a, b\}$ of Π_{17}. In this case, r_3 and r_1 apply. Given $\mathsf{ok}(n_1)$ and a, we derive b' and $\mathsf{ap}(n_1)$. This gives $\mathsf{ok}(n_2)$, b, and b', from which we get $\mathsf{bl}(n_2)$. Finally, we derive $\mathsf{ok}(n_3)$ and a' and $\mathsf{ap}(n_3)$. Unlike above, no integrity constraint is invoked and we obtain an answer set containing a and b.

For another example, consider the program given in (16). While Π_{16} has two standard answer sets, it has no BE-preferred ones under the ordering imposed in (16). Suppose $\mathscr{U}(\Pi_{16})$ had an answer set containing a and b. This yields $\mathsf{rdy}(n_2, n_1)$ (via $c_5(n_2, n_1, b)$) from which we get $\mathsf{ok}(n_2)$. Having a excludes

$$a_2(r_2) = \mathsf{ap}(n_2) \leftarrow \mathsf{ok}(n_2), not\ a, not\ a'\ .$$

Moreover,

$$b_2(r_2, a) = \mathsf{bl}(n_2) \leftarrow \mathsf{ok}(n_2), a'$$

is inapplicable since a' is not derivable (by higher ranked rules). We thus cannot derive $\mathsf{ok}(n_3)$, which makes $d(r_3)$ destroy the putative answer set.

5.4 *Formal elaboration*

The next theorem gives the major result of this section.

Theorem 7
Let $(\Pi, <)$ be a statically ordered logic program and let X be a consistent set of literals.

Then, X is a BE-preferred answer set of $(\Pi, <)$ iff $X = Y \cap \mathscr{L}_{\mathscr{A}}$ for some answer set Y of $\mathscr{U}(\Pi) \cup \{(n_1 \prec n_2) \leftarrow \mid (r_1, r_2) \in <\}$.

In what follows, we elaborate upon the structure of the encoded logic programs:

Proposition 4
Let Π be an ordered logic program over $\mathscr{L}$ and let X be a consistent answer set of $\mathscr{U}(\Pi)$.

Then, we have the following results:

1. $X \cap \mathscr{L}$ is a consistent (standard) answer set of Π;
2. $(X \cap \mathscr{L})' = X \cap \mathscr{L}'$ (or $L \in X$ iff $L' \in X$ for $L \in \mathscr{L}$);
3. $r \in \Pi \cap \Gamma^X_{\mathscr{U}(\Pi)}$ iff $a_2(r) \in \Gamma^X_{\mathscr{U}(\Pi)}$;
4. $r \in \Pi \setminus \Gamma^X_{\mathscr{U}(\Pi)}$ iff $b_1(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ or $b_2(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ for some $L \in body^+(r) \cup body^-(r)$;
5. if $c_5(r, s, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$, then $b_2(s, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ for $L \in body^-(s)$.

The last property shows that eliminated rules are eventually found to be inapplicable. Thus, it is sufficient to remove r' from the preference handling process in $\mathscr{L}'$; the rule is found to be blocked anyway.

In analogy to Proposition 2, the following result shows that our encoding treats all rules and it gathers complete knowledge on their applicability status.

Proposition 5
Let X be a consistent answer set of $\mathscr{U}(\Pi)$ for ordered logic program Π.

We have for all $r \in \Pi$ that

1. $\mathsf{ok}(n_r) \in X$; and
2. $\mathsf{ap}(n_r) \in X$ iff $\mathsf{bl}(n_r) \notin X$.

The last result reveals an alternative choice for integrity constraint $d(r)$, namely

$$\leftarrow not\ \mathsf{ap}(n_r), not\ \mathsf{bl}(n_r)\ .$$

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 subsequent derivation of ok-literals within the grounded enumerations of the generating default rules.

Theorem 8
Let X be a consistent answer set of

$$\mathscr{U}'(\Pi) = \mathscr{U}(\Pi) \cup \{(n_1 \prec n_2) \leftarrow \mid (r_1, r_2) \in <\}$$

for some statically ordered logic program $(\Pi, <)$. Let $\langle s_i \rangle_{i \in I}$ be a grounded enumeration of $\Gamma^X_{\mathscr{U}'(\Pi)}$.

Define

$$\hat{\Pi} = \Pi \setminus \{r \in \Pi \mid head(r) \in X, body^-(r) \cap X \neq \emptyset\}$$

and, for all $r_1, r_2 \in \hat{\Pi}$, define $r_1 \ll r_2$ iff $k_2 < k_1$ where $s_{k_j} = c_1(r_j)$ for $j = 1, 2$.

Then, $\ll$ is a total ordering on $\hat{\Pi}$ such that $(< \cap (\hat{\Pi} \times \hat{\Pi})) \subseteq \ll$.

That is, whenever $\Pi = \hat{\Pi}$, we have that $\ll$ is a total ordering on Π such that $< \subseteq \ll$.

Finally, one may ask why we do not need to account for the 'inherited' ordering in Condition 2 of Definition 9. In fact, this is taken care of through the tags $\mathsf{ap}(n_r)$ in the consequents of rules $a_2(r)$ that guarantee an isomorphism between Π and $\{a_2(r) \mid r \in \Pi\}$ in Definition 9. 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 answer set at hand.

5.5 *Extensions to the dynamic case*

The original elaboration of BE-preferred answer sets in Brewka and Eiter (1998; 1999) deals with the static case only.[11] In fact, our framework leaves room for two alternative extensions to the dynamic case, depending on how (or: 'where') preference information is formed. The more prescriptive option is to form preference information 'on the fly' within language $\mathscr{L}'$. This is close to the dynamic setting explored in section 4. The more descriptive approach is to gather all preference information inside the standard answer set formed in language $\mathscr{L}$ and then to rely on this when verifying preferences within $\mathscr{L}'$.

While the latter approach is already realised through $\mathscr{U}$ in Definition 12,[12] the former is obtained by replacing all occurrences of $\prec$ in $c_2(r,s)$, $c_3(r,s)$, and $c_4(r,s)$ by $\prec'$ and by adding $(t(r,s,t))'$ and $(as(r,s))'$ so that $\prec'$ becomes a strict partial order. Let us refer to the variant of $\mathscr{U}$ obtained in this way by $\mathscr{V}$. The difference between both strategies can be explained by the following example. Consider ordered program $\Pi_{18} = \{r_1, \ldots, r_4\}$, where

$$\begin{array}{rcrcl} r_1 & = & a & \leftarrow & not\ \neg a\ , \\ r_2 & = & b & \leftarrow & not\ \neg b\ , \\ r_3 & = & n_1 \prec n_2 & \leftarrow & , \\ r_4 & = & n_3 \prec n_1 & \leftarrow & . \end{array} \tag{18}$$

[11] An extension to the dynamic case is briefly described for default theories in Brewka and Eiter (2000).
[12] A closer look at the encoding of static preferences in Theorem 7 reveals that there are no 'primed' occurrences of preference atoms.

Clearly, Π_{18} has a single answer set, $\{a, b, n_3 \prec n_1, n_1 \prec n_2\}$. While this answer set is accepted by the strategy underlying $\mathscr{U}(\Pi_{18})$, it is denied by that of $\mathscr{V}(\Pi_{18})$. To see this, observe that $\mathscr{V}$ necessitates that all preference information concerning higher ranked rules has been derived before a rule can be considered for application (as formalised in Condition 2 in Definition 3). This is however impossible for rule r_1 because it dominates r_3 which contains preference information about r_1. Unlike this, $\mathscr{U}$ accepts each rule application order tolerating $n_3 \prec n_1$ and $n_1 \prec n_2$.

In fact, we can characterise the strategy underlying transformation $\mathscr{U}$ in the general (dynamic) case by adapting its static concept of order preservation, given in Definition 11. For this, it is sufficient to substitute Π and $<$ by $\Pi^\star$ and $<_X$ in Definition 11. In contrast to Definition 3, this extension to the dynamic case avoids an additional condition assuring that all preference information about a rule is available before it is considered for application. This works because $\mathscr{U}$ gathers all preference information in language $\mathscr{L}$ and then relies on it when verifying preferences within $\mathscr{L}'$. Similar to the static setting, this dynamic strategy thus separates preference formation from preference verification. This is different from the prescriptive strategy of $\mathscr{V}$ that integrates both processes and thus necessitates a concept of order preservation including Condition 2 in Definition 3.

Both strategies $\mathscr{U}$ as well as $\mathscr{V}$ differ from the strictly prescriptive strategies discussed in section 4. To see this, consider the program $\Pi_{19} = \{r_1, r_2, r_3\}$ with

$$\begin{array}{rclcl} r_1 & = & a & \leftarrow & not\ \neg a\ , \\ r_2 & = & b & \leftarrow & not\ \neg b\ , \\ r_3 & = & n_1 \prec n_2 & \leftarrow & a, b\ . \end{array} \tag{19}$$

Π_{19} has a single answer set, $\{a, b, n_1 \prec n_2\}$. Both $\mathscr{U}$ and $\mathscr{V}$ accept this set as a preferred one. This is due to the above discussed elimination of groundedness from the preference test. Unlike this, $\mathscr{T}$ yields no answer set on Π_{19} because there is no way to apply r_3 before r_1, as stipulated by Condition 2 in Definition 3.

While it is easy to drop Condition 2 in Definition 3, it takes more effort to realise such a concept of order preservation. First, one has to resort to a generate and test construction similar to the one used above. That is, apart from the original program in some language $\mathscr{L}$, we need a constrained reconstruction process in a mirror language $\mathscr{L}'$. A corresponding translation can be obtained from the one in Definition 12 as follows.

Given an ordered logic program Π, let $\tau'(r)$ be the collection of rules obtained from $\tau(r)$ in Definition 12 by

1. replacing $a_2(r)$ by

$$a_2(r):\quad \mathsf{ap}(n_r) \;\leftarrow\; \mathsf{ok}(n_r), body(r), body(r');\quad \text{and}$$

2. deleting $c_5(r, s, J)$.

(Condition 1 leaves the prerequisites of rules intact, and Condition 2 eliminates the filter expressed by Condition 2c in Definition 11.)

Then, define the logic program $\mathscr{S}(\Pi)$ over $\mathscr{L}^\circ$ as $\mathscr{S}(\Pi) = \Pi \cup \bigcup_{r\in\Pi} \tau'(r)$. One can then show that the answer sets of $\mathscr{S}(\Pi)$ are order preserving in the sense of

Table 1. *Summary of dynamic translations*

Integrated features	$\mathscr{T}$	$\mathscr{S}$	$\mathscr{U}$	$\mathscr{V}$
preference verification and groundedness	×	×		
preference verification and preference formation	×			×

Definition 1, once Π and $<$ are replaced by $\Pi^\star$ and $<_X$. This leaves us with an enumeration of the generating rules, as opposed to the entire program and there is no need for stipulating Condition 2 in Definition 3 anymore.

Note that the strategy underlying $\mathscr{S}$ amounts to a synergy between that of $\mathscr{T}$ and $\mathscr{U}$: While borrowing the guessing of preferences from $\mathscr{U}$, it sticks to the integration of groundedness into the associated concept of order preservation. The above translations along with their properties are summarised in Table 1.

6 Implementation

Our approach has been implemented and serves as a front-end to the logic programming systems dlv (Eiter *et al.*, 1997) and smodels (Niemelä and Simons, 1997). These systems represent state-of-the-art implementations for logic programming within the family of stable model semantics (Gelfond and Lifschitz, 1988). For instance, dlv also admits strong negation and disjunctive rules. The resulting compiler, called plp (for preference logic programming), is implemented in Prolog and is publicly available at

`http://www.cs.uni-potsdam.de/~torsten/plp/` .

This URL contains also diverse examples taken from the literature, including those given in the preceding sections. The current implementation runs under the Prolog systems ECLiPSe and SICStus and comprises roughly 1400 lines of code. It currently implements the three major strategies elaborated upon in this paper, namely $\mathscr{T}$, $\mathscr{U}$, and $\mathscr{W}$, given in sections 4, 5 and 4.3, respectively.

The plp compiler differs from the approach described above in two minor respects: first, the translation applies to named rules only, i.e. it leaves unnamed rules unaffected; and second, it admits the specification of rules containing variables, whereby rules of this form are processed by applying an additional instantiation step.

The syntax of plp is summarised in Table 2. For illustration, we give in figure 1 the file comprising Example (10). Once this file is read into plp it is subject to multiple transformations. Most of these transformations are rule-centered in the sense that they apply in turn to each single rule. The first phase of the compilation is system independent and corresponds more or less to the transformations given in the preceding sections. While the original file is supposed to have the extension `lp`, the result of the system-independent compilation phase is kept in an intermediate file with extension `pl`. Applying the implementation of transformation $\mathscr{T}$ to the

Table 2. *The syntax of* plp *input files*

Meaning	Symbols	Internal
$\bot, \top$	`false/0`, `true/0`	
$\neg$	`neg/1`, `-/1` (prefix)	`neg_`L, $L \in \mathscr{L}$
not	`not/1`, `~/1` (prefix)	
$\wedge$	`,/1` (infix; in body)	
$\vee$	`;/1`, `v/2`, `|/2` (infix; in head)	
$\leftarrow$	`:-/1` (infix; in rule)	
$\prec$	`</2` (infix)	`prec/2`
$n_r : \langle head(r)\rangle \leftarrow \langle body(r)\rangle$	$\langle head(r)\rangle$ `:-` `name`$(n_r), \langle body(r)\rangle$ $\langle head(r)\rangle$ `:-` $[n_r], \langle body(r)\rangle$	

ok, rdy
ap, bl
		`ok/1`, `rdy/2`
		`ap/1`, `bl/1`

source file in figure 1 yields at first the file in figure 2. We see that merely rules r_2 and r_3 are translated, while r_1 and r_4 are unaffected. This is reasonable since only r_2 and r_3 are subject to preference information. This is similar to the Prolog rule representing $c_1(r)$, insofar as its body refers to r_2 and r_3 only. Classical negation is implemented in the usual way by appeal to integrity constraints, given at the end of figure 2.

While this compilation phase can be engaged explicitly by the command `lp2pl/1`, for instance in SICStus Prolog by typing

```
  | ?- lp2pl('Examples/example'). ,
```

one is usually interested in producing system-specific code that is directly usable by either dlv or smodels. This can be done by means of the commands `lp2dlv/2` and `lp2sm/2`,[13] which then produce system-specific code resulting in files having extensions `dlv` and `sm`, respectively. These files can then be fed into the respective system by a standard command interpreter, such as a UNIX shell, or from within the Prolog system through commands `dlv/1` or `smodels/1`. For example, after compiling our example by `lp2dlv`, we may proceed as follows:

```
| ?- dlv('Examples/example').
Calling :dlv  Examples/example.dlv
dlv [build BEN/Jun 11 2001   gcc 2.95.2 19991024 (release)]

{true, name(n2), name(n3), neg_a, ok(n2), rdy(n2,n2), rdy(n2,n3),
 rdy(n3,n3),prec(n3,n2), neg_prec(n2,n3), ap(n2), b, rdy(n3,n2),
 ok(n3), bl(n3)}
```

[13] These files are themselves obtainable from the intermediate `pl`-files via commands `pl2dlv/1` and `pl2sm/1`, respectively.

```
    neg a .
        b :- name(n2), neg a, not c.
        c :- name(n3), not b.
(n3 < n2) :- not d.
```

Fig. 1. The source code of Example (10), given in file `example.lp`.

```
neg_a.
b :- ap(n2).
ap(n2) :- ok(n2), neg_a, not c.
bl(n2) :- ok(n2), not neg_a.
bl(n2) :- ok(n2), c.
c :- ap(n3).
ap(n3) :- ok(n3), not b.
bl(n3) :- ok(n3), b.
prec(n3, n2) :- not d.
ok(X) :- name(X), rdy(X, n2), rdy(X, n3).
rdy(X,Y) :- name(X), name(Y), not prec(X,Y).
rdy(X,Y) :- name(X), name(Y), prec(X,Y), ap(Y).
rdy(X,Y) :- name(X), name(Y), prec(X,Y), bl(Y).
neg_prec(Y, X) :- name(X), name(Y), prec(X,Y).
prec(X,Z) :- name(X), name(Z), name(Y), prec(X,Y), prec(Y,Z).
name(n3).                   name(n2).
false :- a, neg_a.          false :- b, neg_b.
false :- c, neg_c.          false :- d, neg_d.
false :- name(X), name(Y), prec(X,Y), neg_prec(X,Y).
```

Fig. 2. The (pretty-printed) intermediate code resulting from Example (10), given in file `example.pl`.

Both commands can be furnished with the option `nice` (as an additional argument) in order to strip off auxiliary predicates:

```
| ?- dlv('Examples/example',nice).
Calling :dlv  -filter=a [...] -filter=neg_d Examples/example.dlv
dlv [build BEN/Jun 11 2001   gcc 2.95.2 19991024 (release)]

{neg_a, b}
```

The above series of commands can be engaged within a single one by means of `lp4dlv/2` and `lp4sm/2`, respectively. The overall (external) comportment of plp is illustrated in figure 3.

For treating variables, some more preprocessing is necessary for instantiating the rules prior to their compilation. The presence of variables is indicated by file extension `vlp`. The contents of such a file is first instantiated by systematically replacing variables by constants and then freed from function symbols by replacing terms by constants, e.g. `f(a)` is replaced by `f_a`. This is clearly a rather pragmatic approach. A more elaborated compilation would be obtained by proceeding right from the start in a system-specific way.

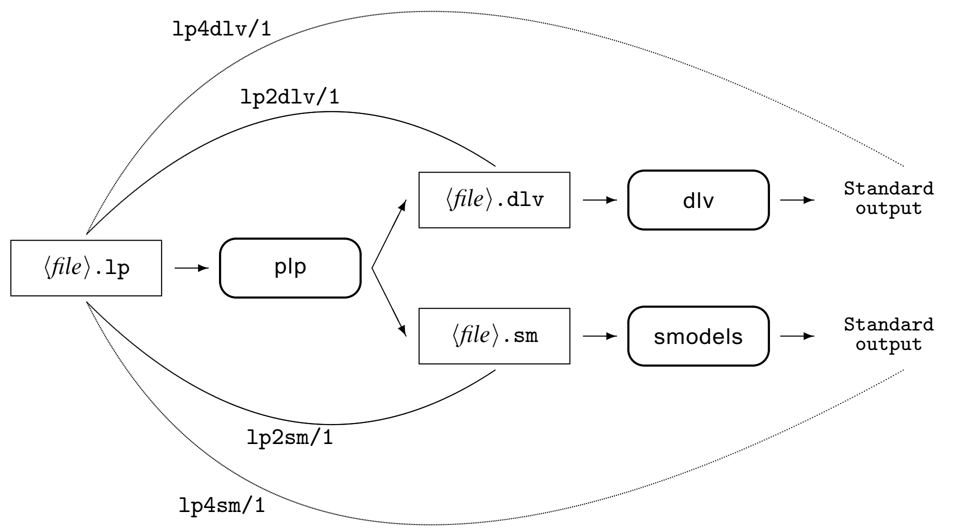

Fig. 3. Compilation with plp: external view.

```
    perfected :- [ucc], possession,             not neg perfected.
neg perfected :- [sma], ship, neg finstatement, not     perfected.

(Y < X) :- [lex_posterior(X,Y)],
           newer(X,Y), not neg (Y < X).
(X < Y) :- [lex_superior(X,Y)],
           state_law(X), federal_law(Y), not neg (X < Y).

possession.              newer(ucc,sma).
ship.                    federal_law(sma).
neg finstatement.        state_law(ucc).

(lex_posterior(X,Y) < lex_superior(X,Y)).
```

Fig. 4. A formalisation of Gordon's legal reasoning example, given in file `legal.vlp`.

For illustration, we give in figure 4 a formalisation of the popular legal reasoning example due to Gordon (1993):

"A person wants to find out if her security interest in a certain ship is 'perfected', or legally valid. This person has possession of the ship, but has not filed a financing statement. According to the code UCC, a security interest can be perfected by taking possession of the ship. However, the federal Ship Mortgage Act (SMA) states that a security interest in a ship may only be perfected by filing a financing statement. Both UCC and SMA are applicable; the question is which takes precedence here. There are two legal principles for resolving such conflicts. Lex Posterior *gives precedence to newer laws; here we have that UCC is more recent than SMA. But* Lex Superior *gives precedence to laws supported by the higher authority; here SMA has higher authority since it is federal law."*

Compiling file legal.vlp yields the following result:

```
| ?- vlp4dlv('Examples/Variables/legal',nice).
Calling :dlv  -filter=[...] Examples/Variables/legal.dlv
dlv [build BEN/Jun 11 2001   gcc 2.95.2 19991024 (release)]

{possession, ship, neg_finstatement, newer(ucc,sma),
 state_law(ucc), federal_law(sma), neg_perfected}
```

More details on the plp compiler can be found at the above cited URL or in Delgrande *et al.* (2000b).

7 Other approaches to preference

The present preference framework has its roots in an approach proposed in Delgrande and Schaub (1997; 2000a) for handling preference information in default logic. There, preference information is expressed using an ordered *default theory*, consisting of default rules, world knowledge, and a preference relation between default rules. Such ordered default theories are then translated into standard default theories, determining extensions of the given theory in which the preferences are respected. Both static and dynamic orderings are discussed, albeit in a less uniform manner than realised in the present case. More specifically, Delgrande and Schaub (1997; 2000a) define the notion of an order preserving extension for static preference orderings only. As well, the encoding of dynamic preferences relies on an additional predicate $\not<$, expressing non-preference between two (names of) rules. In contrast, the current framework is specified right from the beginning for the dynamic case, and static preferences are just a special instance of dynamic ones. Furthermore, Delgrande and Schaub (1997; 2000a) are primarily concerned with a *specific* preference strategy, similar to that expressed by Definition 1, whilst here the emphasis is to demonstrate the flexibility of the framework by providing encodings for *differing preference strategies*. Finally, in contradistinction to other preference approaches requiring dedicated algorithms, the existence of readily available solvers for the answer set semantics, like dlv and smodels, makes a realisation of the current approach straightforward, as documented by the discussion in the previous section.

The problem of dealing with preferences in the context of nonmonotonic rule systems has attracted extensive interest in the past decades. In fact, for almost every nonmonotonic approach there exist prioritised versions designed to handle preference information of some form (Gelfond *et al.*, 1989; Konolige, 1988; Rintanen, 1994; Benferhat *et al.*, 1993; Nebel, 1998; Eiter and Gottlob, 1995; Brewka, 1989; Brewka, 1996). Prioritised versions of default logic and logic programming under the answer set semantics include Baader and Hollunder (1993) and Brewka (1993), Rintanen (1998b), Sakama and Inoue (1996) Gelfond and Son (1997) and Zhang and Foo (1997), as well as those approaches discussed earlier. As argued in Delgrande and Schaub (2000a), these approaches can be divided into *descriptive* and *prescriptive* approaches. In the former case, one has a 'wish list' where the intent is that one way or another the highest-ranked rules be applied. In the latter case the ordering reflects the order in which rules should be applied. The relationship of Doyle and

Wellman's work (Doyle and Wellman, 1991) to such preferences is also discussed in Delgrande and Schaub (2000a).

The approach of Rintanen (1998b) addresses descriptive preference orders in default logic. This notion of preference differs conceptually from the preference orderings dealt with here. In particular, Rintanen's method is based on a meta-level filtering process eliminating extensions which are not considered to be preferred according to certain criteria. This method yields a higher worst-case complexity than standard default logic (providing the polynomial hierarchy does not collapse), and a similar method applied to logic programs under the answer set semantics results likewise in an increase of complexity. Consequently, Rintanen's approach is not polynomial-time translatable into standard logic programs (given the same proviso as before) and can thus not be efficiently represented within our framework. Furthermore, as argued in Brewka and Eiter (1999), Rintanen's approach does not obey Principle II, and exhibits counter-intuitive results in some cases.

For prescriptive approaches, Baader and Hollunder (1993) and Brewka (1994) present prioritised variants of default logic in which the iterative specification of an extension is modified. Preference information is given at the meta-level (thus, only static preferences are considered), and a default is only applicable at an iteration step if no $<$-greater default is applicable.[14] The primary difference between these approaches rests on the number of defaults applicable at each step: while Brewka allows only for applying a single default that is maximal with respect to a total extension of $<$, Baader and Hollunder allow for applying all $<$-maximal defaults at each step. Both approaches violate Principle I, but as shown by Rintanen (1998a), neither of them result in an increase of worst-complexity compared to standard default logic. Thus, the relevant reasoning tasks associated with these preference approaches can be translated in polynomial time into equivalent problems of standard default logic, which makes the two preference methods amenable for the framework of Delgrande and Schaub (1997; 2000a). However, neither Brewka nor Baader and Hollunder deal with context-sensitive preferences.

The previously discussed methods rely on a two-level approach, treating preferences at the meta-level by defining an alternative semantics. A similar methodology is inherent to most approaches to preference handling in extended logic programming. One of the few exceptions is given by the approach in Gelfond and Son (1997), which avoids defining a new logic programming formalism in order to cope with preference information, although it still utilises a two-layered approach in reifying rules and preferences. For example, a rule such as $p \leftarrow r, \neg s, not\ q$ is expressed by the formula $default(n, p, [r, \neg s], [q])$ (or, after reification, by the corresponding *term* inside a *holds*-predicate, respectively) where n is the name of the rule. The semantics of a domain description is given in terms of a set of domain-independent rules for predicates like *holds*. These rules can be regarded as a meta-interpreter for the domain description. Interestingly, the approach is based on the notion of 'defeat' (of justifications) in contrast to an order-preserving consideration of rules, as found in

[14] These authors use $<$ in the reverse order from us.

our approach. Also, the specific strategy elaborated upon in Gelfond and Son (1997) differs from the ones considered here in that the preference $d_1 < d_2$ *"stops the application of default d_2 if defaults d_1 and d_2 are in conflict with each other and the default d_1 is applicable"* (Gelfond and Son, 1997). (Such strategies are also studied in Delgrande and Schaub (2000b).) For detecting such conflicts, however, the approach necessitates an extra conflict-indicating predicate. That is, one must state explicitly $\mathit{conflict}(d_1, d_2)$ to indicate that d_1 and d_2 are conflicting. In principle, as with our framework, the approach of Gelfond and Son offers a variety of different preference handling instances.

Zhang and Foo (1997) present an operational semantics for ordered logic programs based on an iterative reduction to standard programs. The approach admits both static and dynamic preferences, in which the dynamic case is reduced to the static one. Interestingly, the semantics has a nondeterministic flavour in the sense that an ordered program may be reduced in more than one way to a standard program. As shown by Zhang (2000), the somewhat elaborate semantics results in a worst-case complexity which lies at the second level of the polynomial hierarchy. Thus, it is intrinsically harder than standard answer set semantics, providing the polynomial hierarchy does not collapse. Consequently, under this proviso, it is not possible to express Zhang and Foo's method within our framework in a polynomial way, unless our language is extended. Their approach leads also to new answer sets, once preferences are added, as illustrated by the following program:

$$\begin{array}{rcl} r_1 & = & p \leftarrow \mathit{not}\ q_1 \\ r_2 & = & \neg p \leftarrow \mathit{not}\ q_2 \end{array} \qquad \text{and} \qquad r_1 < r_2\ .$$

Zhang and Foo's approach yields two answer sets (one with p and one with $\neg p$). In contrast, no preferred answer set is obtained under DST-, WZL-, or BE-preference. Finally, their approach violates Principle II of Brewka and Eiter (1999).

The method of Sakama and Inoue (1996) has the interesting feature that the given preference order is not a relation between (names of) rules, as in the previous frameworks, but represents a relation between *literals*.[15] This relation is used to determine a preference relation on the answer sets of a disjunctive logic program.[16] Intuitively, an answer set X_1 is at least as preferable as an answer set X_2 iff there are literals $L_1 \in X_1 \setminus X_2$ and $L_2 \in X_2 \setminus X_1$ such that L_1 has at least the priority of L_2, and there is no literal $L_2' \in X_2 \setminus X_1$ which has strictly higher priority than L_1. An answer set X is preferred iff there is no other answer set which is strictly preferred over X. The minimality criterion on answer sets makes this approach (presumably) harder than standard answer set semantics for disjunctive logic programs, yielding a worst-case complexity which lies at the third level of the polynomial hierarchy. Thus, even by restricting rules to the non-disjunctive case, Sakama and Inoue's approach does not admit a polynomial representation within our framework (providing the polynomial hierarchy does not collapse). As well, their approach does not satisfy Principle II of Brewka and Eiter (1999).

[15] Cf. Geffner and Pearl (1992) for another approach to preferences on literals.
[16] Disjunctive logic programs are characterised by permitting disjunctions in rule heads.

Finally, Wang *et al.* (2000) present an approach intermediate between the prescriptive approach described in section 4 and the more descriptive approach of Brewka and Eiter (1999), expressed in our framework in section 5. An encoding of Wang *et al.*'s approach in our framework was given in section 4.3. We obtain the following relations among these three approaches. For differentiating the answer sets obtained in these approaches, let us refer to the answer sets obtained in the approach of Wang *et al.* (2000) as WZL-*preferred*, in analogy to the term BE-*preferred answer set* introduced earlier, referring to an answer set obtained in the approach of Brewka and Eiter (1999).

Then, for a given statically ordered logic program $(\Pi, <)$, the following results can be shown (Schaub and Wang, 2001a):

(i) Every $<$-preserving answer set (in the sense of section 4) is WZL-preferred.
(ii) Every WZL-preferred answer set is BE-preferred.
(iii) Every BE-preferred answer set is an answer set.

In no instance do we obtain the converse. For example, (13) is a counterexample in the first case, and (19) in the second. Thus, the framework illustrates that the preference approaches of section 4, Wang *et al.* (2000), and Brewka and Eiter (1999) form a hierarchy of successively-weaker notions of preference. In fact, this hierarchy is of practical value since it allows us to apply the concept of a 'back-up'-semantics (van der Hoek and Witteveen, 2000) for validating strategies on specific ordered logic programs. The idea is to first interpret a program with respect to the strongest semantics, and then to gradually weaken the strategy until a desirable set of conclusions is forthcoming.

A feature common to the latter three approaches is that, for statically ordered programs, the addition of preferences never increases the number of preferred answer sets.[17] In fact, we have seen above that preferences may sometimes be too strong and deny the existence of preferred answer sets although standard answer sets exist. This is because preferences impose additional dependencies among rules that must be respected by the standard answer sets. In a related paper (Schaub and Wang, 2001b), it is shown that programs whose order amounts to a stratification guarantee a unique answer set. Furthermore, Delgrande and Schaub (2000a) show how graph structures, as introduced in Papadimitriou and Sideri (1994), give sufficient conditions for guaranteeing preferred extensions of ordered default theories; these results obviously carry over to extended logic programs due to the results in Gelfond and Lifschitz (1991).

8 Discussion

8.1 Further issues and refinements

We briefly sketch the range of further applicability and point out distinguishing features of our framework here.

[17] Cf. Proposition 5.4 in Brewka and Eiter (1999).

First, we draw the reader's attention to the expressive power offered by dynamic preferences in connection with variables in the input language. Consider the rule

$$n_1(x) \prec n_2(y) \leftarrow p(y), not\ (x = c), \tag{20}$$

where $n_1(x), n_2(y)$ are names of rules containing the variables x and y, respectively. Although such a rule represents only its set of ground instances, it nonetheless constitutes a much more concise specification. Since most other approaches employ static preferences of the form $n_1(x) \prec n_2(y) \leftarrow$, these approaches would necessarily have to express (20) as an enumeration of static ground preferences rather than a single rule.

Second, we note that our methodology is also applicable to disjunctive logic programs (where rule heads are disjunctions of literals). To see this, observe that the transformed rules unfold the conditions expressed in the body of the rules, while a rule's head remain untouched, as manifested by rule $a_1(r)$. In addition to handling preferences about disjunctive rules, our approach allows us to express disjunctive preferences, such as

$$(n_3 \prec n_2) \vee (n_3 \prec n_4) \leftarrow \neg a\ ,$$

saying that if $\neg a$ holds, then either r_2 or r_4 are preferred to r_3. Thus, informally, given $\neg a$ along with three mutually exclusive rules r_1, r_2, r_3, the above preference results in two rather than three answer sets.

Finally, there is a straightforward way of accommodating preferences on literals, as for instance put forward in Sakama and Inoue (2000). A preference $p \lhd q$ among two literals $p, q \in \mathscr{L}$ can then be realised by stipulating that $r_1 < r_2$ whenever $head(r_1) = p$ and $head(r_2) = q$.

8.2 Concluding remarks

We have described a general *methodology* in which logic programs containing preferences on the set of rules can be compiled into logic programs under the answer set semantics. An ordered logic program, in which preference atoms appear in the program rules, is transformed into a second, extended logic program. In this second program, no explicit preference information appears, yet preferences are respected, in that the answer sets obtained in the transformed theory correspond to the preferred answer sets of the original theory.

In our main approach, we provide an encoding for capturing a strongly prescriptive notion of preference, in which rules are to be applied in accordance with the prescribed order. Since this approach admits a dynamic (as well as, trivially, static) notion of preference, one is able to specify preferences holding in a given context, by 'default', or where one preference depends on another. We also show how the approach of Brewka and Eiter (1999) can be expressed in our framework for preferences. Further, we extend their approach to handle the dynamic case as well.

These translations illustrate the generality of our framework. As well, they shed light specifically on Brewka and Eiter's approaches, since we can provide a translation and encoding of their approaches into extended logic programs. In a sense then, our overall methodology provides a means of *axiomatising* different preference orderings, in that the image of a translation shows how an approach can be encoded in an extended logic program. The contrast between Definitions 2 and 12 for example illustrate succinctly how our main approach to preferences and that of Brewka and Eiter (1999) relate. In general then our approach provides a uniform basis within which diverse approaches and encodings can be compared. This is illustrated by the hierarchy of approaches discussed at the end of section 7. Similarly, the framework allows us to potentially formalise the interaction of several orderings inside the same framework, and to specify in the theory how they interact.

In all of these approaches we can prove that the resulting transformations accomplish what is claimed: that the appropriate rules in the image of the transformation are treated in the correct order, that all rules are considered, that we do indeed obtain preferred answer sets, and so on. In addition to the formal results, we illustrate the generality of the approach by formalising an example due to Gordon (1993), as well as giving examples of context-based preference, canceling preferences, preferences among preferences, and preferences by default.

Our approach of translating preferences into extended logic programs has several other advantages over previous work. First, we avoid the two-level structure of such work. While the previous 'meta-level' approaches must commit themselves to a semantics and a fixed strategy, our approach (as well as that of Gelfond and Son (1997)) is very flexible with respect to changing strategies, and is open for adaptation to different semantics and different concepts of preference handling (as illustrated by our encoding and extension of the method of Brewka and Eiter (1999)). Second, for a translated program in our approach, any answer set is 'preferred', in the sense that *only* 'preferred' answer sets (as specified by the ordering on rules) are produced. In contrast, many previous approaches, in one fashion or another, select among answer sets for the most preferred. Hence one could expect the present approach to be (pragmatically) more efficient, since it avoids the generation of unnecessary answer sets. As well, as section 7 notes, some of these other approaches are at a higher level of the complexity hierarchy than standard extended logic programs. Lastly, in 'compiling' preferences into extended logic programs, some light is shed on the foundations of extended logic programs. In particular, we show implicitly that for several notions of explicit, dynamic, preference expressed among rules, such preference information does not increase the expressibility of an extended logic program.

Finally, this paper demonstrates that these approaches are easily implementable. We describe here a compiler (described more fully in Delgrande *et al.* (2000b)) as a front-end for `dlv` and `smodels`. The current prototype is available at

`http://www.cs.uni-potsdam.de/~torsten/plp/` .

This implementation is used as the core reasoning engine in a knowledge-based information system for querying Internet movie databases, currently developed at the Vienna University of Technology (Eiter *et al.*, 2002). Also, it serves for modeling linguistic phenomena occurring in phonology and syntax (Besnard *et al.*, 2002), which are treated in linguistics within Optimality Theory (Prince and Smolensky, 1993; Kager, 1999).

Acknowledgements

The authors would like to thank the anonymous referees for their constructive comments which helped to improve the paper. The first author was partially supported by a Canadian NSERC Research Grant; the second author was partially supported by the German Science Foundation (DFG) under grant FOR 375/1-1, TP C; and the third author was partially supported by the Austrian Science Fund (FWF) under grants N Z29-INF and P13871-INF.

A Proofs

For later usage, we provide the following lemmata without proof.

Lemma 1
Let X be an answer set of logic program Π. Then, we have for $r \in \Pi$

1. if $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$, then $head(r) \in X$;
2. if $r \in \Gamma_\Pi^X$, then $head(r) \in X$.

The inverse of each of these propositions holds, whenever $head(r) = head(r')$ implies $r = r'$.

Lemma 2
Let X be an answer set of logic program Π.
Then, there is a grounded enumeration $\langle r_i \rangle_{i \in I}$ of Γ_Π^X.

That is, we have for consistent X that $body^+(r_i) \subseteq \{head(r_j) \mid j < i\}$.
The following result is taken from Proposition 5.1 in Brewka and Eiter (1999).

Theorem 9
Let $(\Pi, \ll)$ be a fully ordered program and let X be an answer set of Π.
Then, X is a BE-preferred answer set of $(\Pi, \ll)$ iff for each rule $r \in \Pi$ with

1. $body^+(r) \subseteq X$ and
2. $head(r) \notin X$

there is some rule $r' \in \Gamma_\Pi^X$ such that

1. $r \ll r'$ and
2. $head(r') \in body^-(r)$.

In the sequel, we give the proofs for the theorems in the order the latter appear in the body of the paper.

A.1 Proofs of section 4

Proof of Proposition 1
Let X be a consistent answer set of $\mathscr{T}(\Pi)$ for an ordered program Π.

1. Since rules $t(r, r', r'')$ and $as(r, r')$ are members of $\mathscr{T}(\Pi)$, for any $r, r', r'' \in \Pi$, the relation $<_X$ is clearly transitive and antisymmetric. Thus, $<_X$ is a strict partial order.
2. Suppose Π contains static preferences only, i.e., $\Pi = \Pi' \cup \Pi''$, where $\Pi'' \subseteq \{(n_r \prec n_{r'}) \leftarrow\mid r, r' \in \Pi'\}$ and Π' contains no occurrences of preference atoms of form $n \prec m$ for some names n, m. Hence, for each consistent answer set Z, Z' of Π and any $(n_r \prec n_{r'}) \leftarrow \in \Pi''$, we have that $(n_r \prec n_{r'}) \in Z$ iff $(n_r \prec n_{r'}) \in Z'$. Since X is assumed to be consistent, well-known properties of answer sets imply that any other answer set Y of Π is also consistent. It follows that $(n_r \prec n_{r'}) \in X$ iff $(n_r \prec n_{r'}) \in Y$, for any answer set Y of Π. From this, $<_X = <_Y$ for any answer set Y is an immediate consequence. □

Proof of Proposition 2
Let X be a consistent answer set of $\mathscr{T}(\Pi)$ for an ordered program Π.

1+2. We prove both propositions by parallel induction on $<_X$.

Base. Let r be a maximal element of $<_X$.

1. By assumption, $n_r \prec n_{r'} \notin X$ for all $r' \in \Pi$. This implies that
$$\mathsf{rdy}(n_r, n_{r'}) \leftarrow \in \mathscr{T}(\Pi)^X \qquad \text{for all } r' \in \Pi\ ,$$
which results in $\mathsf{ok}(n_r) \in Cn(\mathscr{T}(\Pi)^X)$ because $Cn(\mathscr{T}(\Pi)^X)$ is closed under $\mathscr{T}(\Pi)^X$. Hence, $\mathsf{ok}(n_r) \in X$, since $X = Cn(\mathscr{T}(\Pi)^X)$.
2. We distinguish the following two cases:
   - If $a_2(r) \in \Gamma^X_{\mathscr{T}(\Pi)}$, then $\mathsf{ap}(n_r) \in X$ by Lemma 1.
   - If $a_2(r) \notin \Gamma^X_{\mathscr{T}(\Pi)}$, then we have in view of Property 1, namely $\mathsf{ok}(n_r) \in X$, that one of the following two cases is true:
     - If $body^+(r) \nsubseteq X$, then there is some $L^+ \in body^+(r)$ with $L^+ \notin X$. Hence, $(b_1(r, L^+))^+ \in \mathscr{T}(\Pi)^X$, yielding $\mathsf{bl}(n_r) \in X$ in analogy to 1.
     - If $body^-(r) \cap X \neq \emptyset$, then there is some $L^- \in body^-(r)$ with $L^- \in X$. Hence, $(b_2(r, L^-))^+ \in \mathscr{T}(\Pi)^X$, yielding $\mathsf{bl}(n_r) \in X$ in analogy to 1.

     This shows that either $\mathsf{ap}(n_r) \in X$ or $\mathsf{bl}(n_r) \in X$. In other words, $\mathsf{ap}(n_r) \in X$ iff $\mathsf{bl}(n_r) \notin X$.

Step. Consider $r \in \Pi$. Assume that $\mathsf{ok}(n_{r'}) \in X$ and either $\mathsf{ap}(n_{r'}) \in X$ or $\mathsf{bl}(n_{r'}) \in X$ for all r' with $r <_X r'$.

1. In analogy to the base case, we have $\mathsf{rdy}(n_r, n_{r''}) \in X$ for all $r'' \in \Pi$ with $r \not<_X r''$.
Consider r' with $r <_X r'$. We thus have $n_r \prec n_{r'} \in X$. By the induction hypothesis, we have either $\mathsf{ap}(n_{r'}) \in X$ or $\mathsf{bl}(n_{r'}) \in X$. Hence we have either
$$body((c_3(r, r'))^+) \subseteq X \text{ or } body((c_4(r, r'))^+) \subseteq X\ .$$

This implies $\mathsf{rdy}(n_r, n_{r'}) \in X$ for all $r' \in \Pi$ with $r <_X r'$.
We have thus shown that $\mathsf{rdy}(n_r, n_{r'''}) \in X$ for all $r''' \in \Pi$.
By the fact that X is closed, we get $\mathsf{ok}(n_r) \in X$.

2. Analogous to the base case. $\square$

Proof of Proposition 3
Let X be a consistent answer set of $\mathcal{T}(\Pi)$ for an ordered program Π, let $\Omega = \mathcal{T}(\Pi)^X$, and consider some $r \in \Pi$.

1. Given $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$ and $body^+(r) \subseteq T^j_\Omega\emptyset$, we get $body^+(a_2(r)) \subseteq T^{\max(i,j)}_\Omega\emptyset$. With $body^-(a_2(r)) \cap X = \emptyset$, this implies $\mathsf{ap}(n_r) \in T^{\max(i,j)+1}_\Omega\emptyset$.
2. Given $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$ and $body^+(r) \nsubseteq X$, we get $body^+(b_1(r, L^+)) \subseteq T^i_\Omega\emptyset$ and $body^-(b_1(r, L^+)) \cap X = \emptyset$, for some $L^+ \in body^+(r)$. Thus, $\mathsf{bl}(n_r) \in T^{i+1}_\Omega\emptyset$.
3. We have $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$ and $L^- \in X$ for some $L^- \in body^-(r)$. Assume $L^- \in T^k_\Omega\emptyset$ for some minimal k. Then, we get $body(b_2(r, L^-)) \subseteq T^{\max(i,k)}_\Omega\emptyset$. This implies $\mathsf{bl}(n_r) \in T^{\max(i,k)+1}_\Omega\emptyset$. That is, $\mathsf{bl}(n_r) \in T^j_\Omega\emptyset$ for some $j > i$.
4. Assume $\mathsf{ap}(n_r) \in T^j_\Omega\emptyset$ for some $j < i+2$. Since $\mathsf{ap}(n_r)$ can only be derived by means of rule $a_2(r) \in \mathcal{T}(\Pi)$, we obtain that $\mathsf{ok}(n_r) \in T^{j-1}_\Omega\emptyset$. But $T^{j-1}_\Omega\emptyset \subseteq T^i_\Omega\emptyset$, so $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$. Similar arguments show that $\mathsf{bl}(n_r) \in T^k_\Omega\emptyset$ for some $k < i+2$ implies $\mathsf{ok}(n_r) \in T^i_\Omega\emptyset$. $\square$

Proof of Theorem 1
Let Π be an ordered logic program, X a consistent answer set of $\mathcal{T}(\Pi)$, and $\langle r_i \rangle_{i \in I}$ a grounded enumeration of $\Gamma^X_{\mathcal{T}(\Pi)}$.

Consider $r_1, r_2 \in \Pi$ with $n_{r_1} <_X n_{r_2}$. Let $r_i = c_1(r_1)$ and $r_j = c_1(r_2)$. We show that $j < i$.

Given that $r_i = c_1(r_1)$, we have $\{\mathsf{rdy}(r_1, r) \mid r \in \Pi\} \subseteq \{head(r_k) \mid k < i\}$. In particular, we then have $\mathsf{rdy}(r_1, r_2) \in \{head(r_k) \mid k < i\}$. Because of $n_{r_1} <_X n_{r_2}$, we have $(n_{r_1} \prec n_{r_2}) \in X$. Therefore, $c_2(r_1, r_2) \notin \Gamma^X_{\mathcal{T}(\Pi)}$. Hence, we must have either

$$c_3(r_1, r_2) \in \{r_k \mid k < i\} \quad \text{or} \quad c_4(r_1, r_2) \in \{r_k \mid k < i\}$$

and then furthermore that either

$$a_2(r_2) \in \{r_k \mid k < i\} \text{ or } b_1(r_2, L^+) \in \{r_k \mid k < i\} \text{ or } b_2(r_2, L^-) \in \{r_k \mid k < i\}$$

for some $L^+ \in body^+(r_2)$ or some $L^- \in body^-(r_2)$. In either case, we must have $\mathsf{ok}(r_2) \in \{head(r_k) \mid k < i\}$, that is, $c_1(r_2) \in \{r_k \mid k < i\}$, and thus $j < i$. $\square$

Proof of Theorem 2
Let $(\Pi, <)$ be a statically ordered logic program over $\mathcal{L}_\mathcal{A}$ and X a set of literals.

Only-if part. Let X be a $<$-preserving answer set of Π.
Define $\overline{\Pi} = (\Pi \cup \{(n_r \prec n_{r'}) \leftarrow \ \mid r < r'\})^\star$ and $\overline{X} = X \cup \{\neg(n_{r'} \prec n_r) \mid r < r'\}$.
We draw on the following propositions in the sequel.

Lemma 3
Given the above prerequisites, we have

(i) $X = \overline{X} \cap \mathcal{L}_\mathcal{A}$;

(ii) $< = <_X = <_{\overline{X}}$;
(iii) $\overline{X}$ is a $<$-preserving answer set of $\overline{\Pi}$.

Hence there is an enumeration $\langle r_i \rangle_{i \in I}$ of $\Gamma_{\overline{\Pi}}^{\overline{X}}$ satisfying Conditions 1 to 3 in Definition 1. Without loss of generality, assume that there is some $m \in I$ such that

$$\begin{array}{rrcl} & \{r_i \mid 0 \leqslant i \leqslant m, i \in I\} & = & \{r \mid head(r) \in \mathscr{L}_{\mathscr{A}_<}\} \\ \text{and} & \{head(r_i) \mid i > m, i \in I\} \cap \mathscr{L}_{\mathscr{A}_<} & = & \emptyset\ . \end{array} \quad \text{(A 1)}$$

From $\langle r_i \rangle_{i \in I}$, we construct an enumeration $\langle r_j \rangle_{j \in J}$ of $\overline{\Pi}$ with $I \subseteq J$ and $\ll_I \subseteq \ll_J$ (where $\ll_I, \ll_J$ are the total orders on $\Gamma_{\overline{\Pi}}^{\overline{X}}$ and $\overline{\Pi}$ induced by $\langle r_i \rangle_{i \in I}$ and $\langle r_j \rangle_{j \in J}$, respectively) such that for every $i, j \in J$ we have that:

1. If $r_i < r_j$, then $j < i$.
   This is obtainable by letting $\ll_J$ be a total extension of $<$, that is, $< \subseteq \ll_J$.
2. If $r_i < r_j$, then there is some $r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}$ such that

   2a. $k < i$ and
   2b. $head(r_k) = (n_{r_i} \prec n_{r_j})$.

   This is trivially satisfiable because of (A 1).
3. If $r_i \in \Gamma_{\overline{\Pi}}^{\overline{X}}$, then $body^+(r_i) \subseteq \{head(r_k) \mid r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}, k < i\}$.
   This is a direct consequence of Condition 1 in Definition 1.
4. If $r_i \in \overline{\Pi} \setminus \Gamma_{\overline{\Pi}}^{\overline{X}}$, then

   4a. $body^+(r_i) \not\subseteq X$ or
   4b. $body^-(r_i) \cap \{head(r_k) \mid r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}, k < i\} \neq \emptyset$.

   Suppose $r_j < r_i$ for some $r_j \in \Gamma_{\overline{\Pi}}^{\overline{X}}$. Then, 4a is a direct consequence of its counterpart in Definition 1, expressed there as Condition 3a.
   Now, consider the $\ll_I$-smallest rule $r_l \in \Gamma_{\overline{\Pi}}^{\overline{X}}$ with $r_l < r_i$. Then, the counterpart of 4b, Condition 3b in Definition 1, implies that $body^-(r_i) \cap \{head(r_k) \mid r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}, k < l\} \neq \emptyset$. Let $r_h \in \Gamma_{\overline{\Pi}}^{\overline{X}}$ be the predecessor of r_l in $\langle r_i \rangle_{i \in I}$. Then, we have $body^-(r_i) \cap \{head(r_k) \mid r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}, k \leqslant h\} \neq \emptyset$. Without violating any other constraints, we can then position r_i in $\langle r_j \rangle_{i \in J}$ such that $h < i < l$. Consequently, we obtain $body^-(r_i) \cap \{head(r_k) \mid r_k \in \Gamma_{\overline{\Pi}}^{\overline{X}}, k < i\} \neq \emptyset$.
   Otherwise, that is, whenever $r_j \not< r_i$ for every $r_j \in \Gamma_{\overline{\Pi}}^{\overline{X}}$, rule r_i can be positioned after the $\ll_I$-maximal rule in $\langle r_j \rangle_{i \in J}$; this satisfies all constraints.

Given that $< = <_{\overline{X}}$, we get that $\overline{X}$ is a $<_{\overline{X}}$-preserving answer set of $\overline{\Pi}$. According to Theorem 3, there is then some answer set Y of $\mathscr{T}(\Pi, <)$ such that $\overline{X} = Y \cap \mathscr{L}$. Consequently, we obtain that $X = \overline{X} \cap \mathscr{L}_{\mathscr{A}} = (Y \cap \mathscr{L}) \cap \mathscr{L}_{\mathscr{A}} = Y \cap \mathscr{L}_{\mathscr{A}}$.

If part. Let Y be an answer set of $\mathscr{T}(\Pi, <)$, i.e., of

$$\mathscr{T}(\Pi \cup \{(n_r \prec n_{r'}) \leftarrow \ \mid r < r'\}).$$

According to Theorem 3, there is then an enumeration $\langle s_k \rangle_{k \in K}$ of

$$(\Pi \cup \{(n_r \prec n_{r'}) \leftarrow \mid r < r'\})^\star$$

satisfying Conditions 1, 2, 3, and 4 in Definition 3.

Define $\langle r_i \rangle_{i \in I}$ as the enumeration obtained from $\langle s_k \rangle_{k \in K}$ by

1. deleting all rules apart from those of form $a_2(r)$ for $r \in \Pi$; and
2. replacing each rule of form $a_2(r)$ by r.

Define $X = \{head(r_i) \mid i \in I\}$. Then, we have $X = Y \cap \mathscr{L}_{\mathscr{A}}$ and $\Gamma^X_\Pi = \{head(r_i) \mid i \in I\}$. Hence $\langle r_i \rangle_{i \in I}$ is an enumeration of Γ^X_Π. Moreover, $\langle r_i \rangle_{i \in I}$ satisfies Conditions 1, 2, and 3 in Definition 1 by virtue of $\langle s_k \rangle_{k \in K}$ satisfying Conditions 1, 2, 3, and 4 in Definition 3. More specifically, Condition 1 in Definition 1 is a direct consequence of Condition 3 in Definition 3. As well, Condition 2 in Definition 1 follows immediately from Condition 1 in Definition 3, while Condition 3 in Definition 1 is a consequence of Conditions 4 and 1 in Definition 3. Hence, X is a $<$-preserving answer set of Π. □

Proof of Theorem 3
Let Π be an ordered logic program over language $\mathscr{L}$.

If part. Let Y be a consistent answer set of $\mathscr{T}(\Pi)$. Define

$$\begin{aligned} X \quad = \quad & \{head(r) \mid \mathsf{ap}(n_r) \in Y\} \cup \{n_r \prec n_t \mid r <_Y s, s <_Y t\} \\ & \cup \{\neg(n_s \prec n_r) \mid r <_Y s\} . \end{aligned}$$

Observe that $X = Y \cap \mathscr{L}$. Furthermore, we note the following useful relationships for X and Y.

Lemma 4
For any $L \in \mathscr{L}$, we have that $L \in Y$ iff $L \in X$.

Given that Y is consistent, this implies that X is consistent, too.

Lemma 5
We have that $<_Y = <_X$.

If part, I. First, we show that X is a (standard) answer set of $\Pi^\star$. That is, $X = Cn((\Pi^\star)^X)$.
Observe that $(\Pi^\star)^X = \Pi^X \cup \{t(r,s,t), as(r,s) \mid r,s,t \in \Pi\}$.

'$\supseteq$' part. We start by showing that X is closed under $(\Pi^\star)^X$.
Consider $r^+ \in \Pi^X$ with $body^+(r) \subseteq X$. We then have $body^-(r) \cap X = \emptyset$. Hence, we also have $body^-(r) \cap Y = \emptyset$ by Lemma 4. This implies that $a_2(r)^+ \in \mathscr{T}(\Pi)^Y$. Note that $body^+(r) \subseteq X$ implies $body^+(r) \subseteq Y$. Also, we have $\mathsf{ok}(n_r) \in Y$ by Condition 1 of Proposition 2. The fact that Y is closed under $\mathscr{T}(\Pi)^Y$ implies $\mathsf{ap}(n_r) \in Y$. We thus get $head(r) \in X$ by definition of X.
Consider $t(r,s,t)$ with $\{n_r \prec n_s, n_s \prec n_t\} \subseteq X$. By Lemma 5, we get $r <_Y s$ and $s <_Y t$. We thus get $head(t(r,s,t)) = n_r \prec n_t \in X$ by definition of X.
Consider $as(r,s)$ with $n_r \prec n_s \in X$. By Lemma 5, we get $r <_Y s$. We thus get $head(as(r,s)) = \neg(n_s \prec n_r) \in X$ by definition of X.
Note that the closure of X under $(\Pi^\star)^X$ shows that $Cn((\Pi^\star)^X) \subseteq X$.

'$\subseteq$' part. We now show that X is the smallest set being closed under $(\Pi^\star)^X$, or equivalently that $X \subseteq Cn((\Pi^\star)^X)$. We observe that

$$X = Y \cap \mathscr{L} = \left(\bigcup_{i\geqslant 0} T^i_{\mathscr{T}(\Pi)^Y}\emptyset\right) \cap \mathscr{L}\ .$$

We then show by induction that $(T^i_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L} \subseteq Cn((\Pi^\star)^X)$ for $i \geqslant 0$.

Base. Trivial since $T^0_{\mathscr{T}(\Pi)^Y}\emptyset = \emptyset$.

Step. Assume $(T^j_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L} \subseteq Cn((\Pi^\star)^X)$ for $0 \leqslant j \leqslant i$.

- If $(T^{i+1}_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L} = \emptyset$, our claim obviously holds.
- If there is some $r \in \Pi$ such that $head(r) \in (T^{i+1}_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L}$, that is, $head(r) = head(a_1(r)^+)$, we have

$$\left\{\begin{array}{lrcl} a_1(r)^+ : & head(r) & \leftarrow & \mathsf{ap}(n_r) \\ a_2(r)^+ : & \mathsf{ap}(n_r) & \leftarrow & \mathsf{ok}(n_r), body^+(r) \end{array}\right\} \subseteq \mathscr{T}(\Pi)^Y\ .$$

  Consequently, we must have $\{\mathsf{ap}(n_r), \mathsf{ok}(n_r)\} \cup body^+(r) \subseteq T^i_{\mathscr{T}(\Pi)^Y}\emptyset$.
  According to the induction hypothesis, $body^+(r) \subseteq (T^i_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L}$ implies $body^+(r) \subseteq Cn((\Pi^\star)^X)$.
  Also, $a_2(r)^+ \in \mathscr{T}(\Pi)^Y$ implies that $body^-(r) \cap Y = \emptyset$. By Lemma 4, we thus get $body^-(r) \cap X = \emptyset$, which implies $r^+ \in \Pi^X$.
  Given that $Cn((\Pi^\star)^X)$ is closed under Π^X, we get that $head(r) \in Cn((\Pi^\star)^X)$.
- If $head(t(r,s,t)) \in (T^{i+1}_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L}$, then we must have $\{n_r \prec n_s, n_s \prec n_t\} \subseteq T^i_{\mathscr{T}(\Pi)^Y}\emptyset$. And by the induction hypothesis, we further get $\{n_r \prec n_s, n_s \prec n_t\} \subseteq Cn((\Pi^\star)^X)$. Given that $Cn((\Pi^\star)^X)$ is closed under $t(r,s,t) \in (\Pi^\star)^X$, we thus obtain that $head(t(r,s,t)) \in Cn((\Pi^\star)^X)$.
- If $head(as(r,s)) \in (T^{i+1}_{\mathscr{T}(\Pi)^Y}\emptyset) \cap \mathscr{L}$, then we must have $n_r \prec n_s \in T^i_{\mathscr{T}(\Pi)^Y}\emptyset$. And by the induction hypothesis, we further get $n_r \prec n_s \in Cn((\Pi^\star)^X)$. Given that $Cn((\Pi^\star)^X)$ is closed under $as(r,s) \in (\Pi^\star)^X$, we thus obtain that $head(as(r,s)) \in Cn((\Pi^\star)^X)$.

In all, we have now shown that $X = Cn((\Pi^\star)^X)$. That is, X is a (standard) answer set of $\Pi^\star$.

If part, II. Next, we show that X is a $<_X$-preserving answer set of $\Pi^\star$. Since Y is a (standard) answer set of $\mathscr{T}(\Pi)$, according to Lemma 2, there is a grounded enumeration $\langle s_k\rangle_{k\in K}$ of $\Gamma^Y_{\mathscr{T}(\Pi)}$.

Define $\langle r_i\rangle_{i\in I}$ as the enumeration obtained from $\langle s_k\rangle_{k\in K}$ by

1. deleting all rules apart from those of form $a_2(r)$, $b_1(r,L^+)$, $b_2(r,L^-)$;
2. replacing each rule of form $a_2(r)$, $b_1(r,L^+)$, $b_2(r,L^-)$ by r; and
3. removing duplicates[18] by increasing i

for $r \in \Pi^\star$ and $L^+ \in body^+(r)$, $L^- \in body^-(r)$.

First of all, we note that $\langle r_i\rangle_{i\in I}$ satisfies Condition 3. This is a direct consequence of the fact that $\langle s_k\rangle_{k\in K}$ enjoys groundedness.

[18] Duplicates can only occur if a rule is blocked in multiple ways.

For establishing Condition 4, consider $r_i \notin \Gamma^X_{\Pi^\star}$. If we have $body^+(r_i) \not\subseteq X$, then this establishes Condition 4a and we are done. Otherwise, if we have $body^-(r) \cap X \neq \emptyset$, then there is some $L^- \in body^-(r)$ such that $L^- \in X$. This implies $L^- \in Y$ by Lemma 4. Given this and that $\mathsf{ok}(r) \in Y$ by Condition 1 of Proposition 2, we obtain $b_2(r, L^-) \in \Gamma^Y_{\mathscr{T}(\Pi)}$. More precisely, let $b_2(r, L^-) = s_k$ for some $k \in K$. Then, we have that $L^- \in \{head(s_l) \mid l \in K, l < k\}$. Given the way $\langle r_i \rangle_{i \in I}$ is obtained from $\langle s_k \rangle_{k \in K}$, we thus obtain that $L^- \in \{head(r_l) \mid r_l \in \Gamma^X_{\Pi^\star}, l < i\}$. That is, we obtain Condition 4b.

Next, we show that $\langle r_i \rangle_{i \in I}$ satisfies Condition 1. Consider $r_i, r_j \in \Pi$ for some $i, j \in I$. Then, there are $k_i, k_j \in K$ such that

$$\begin{array}{lllll}
 & s_{k_i} = a_2(r_i) & & & \\
\text{or} & s_{k_i} = b_1(r_i, L^+) & \text{for some } L^+ \in body^+(r_i) & & \\
\text{or} & s_{k_i} = b_2(r_i, L^-) & \text{for some } L^- \in body^-(r_i)\ , & & \text{and} \\
 & s_{k_j} = a_2(r_j) & & & \\
\text{or} & s_{k_j} = b_1(r_j, L^+) & \text{for some } L^+ \in body^+(r_j) & & \\
\text{or} & s_{k_j} = b_2(r_j, L^-) & \text{for some } L^- \in body^-(r_j)\ . & &
\end{array}$$

Assume $r_i <_X r_j$. Then, we also have $r_i <_Y r_j$ by Lemma 5. Given this, by the properties of transformation $\mathscr{T}$, it is easy to see that $k_j < k_i$ must hold. Given the way $\langle r_i \rangle_{i \in I}$ is obtained from $\langle s_k \rangle_{k \in K}$, the relation $k_j < k_i$ implies that $j < i$. This establishes Condition 1.

For addressing Condition 2, assume $r_i <_X r_j$ for $i, j \in I$. By definition, $r_i <_X r_j$ implies $(n_{r_i} \prec n_{r_j}) \in X$. That is, there is some smallest $h \in I$ such that $r_h \in \Gamma^X_{\Pi^\star}$ and $head(r_h) = (n_{r_i} \prec n_{r_j})$. Furthermore, we obtain $(n_{r_i} \prec n_{r_j}) \in Y$ by Lemma 5. We thus have $c_2(r_i, r_j)^+ \notin \mathscr{T}(\Pi)^Y$, while we have

$$\left\{ \begin{array}{llll} c_3(r_i, r_j)^+ : & \mathsf{rdy}(n_{r_i}, n_{r_j}) & \leftarrow & (n_{r_i} \prec n_{r_j}), \mathsf{ap}(n_{r_j}) \\ c_4(r_i, r_j)^+ : & \mathsf{rdy}(n_{r_i}, n_{r_j}) & \leftarrow & (n_{r_i} \prec n_{r_j}), \mathsf{bl}(n_{r_j}) \end{array} \right\} \subseteq \mathscr{T}(\Pi)^Y\ .$$

Since $\mathsf{rdy}(n_{r_i}, n_{r_j}) \subseteq Y$ (by Condition 1 in Proposition 2), we have either $c_3(r_i, r_j) \in \Gamma^Y_{\mathscr{T}(\Pi)}$ or $c_4(r_i, r_j) \in \Gamma^Y_{\mathscr{T}(\Pi)}$. Analogously, we have $c_1(r_i) \in \Gamma^Y_{\mathscr{T}(\Pi)}$.

Now, consider the enumeration $\langle s_k \rangle_{k \in K}$ of $\Gamma^Y_{\mathscr{T}(\Pi)}$, and, in particular, rules s_{k_i} (as defined above) and $s_{k_h} = a_2(r_h)$ for some $k_h \in K$. Since $\langle s_k \rangle_{k \in K}$ is grounded, the minimality of k_h (due to that of h) implies that there are some $l_1, l_2, l_3 > 0$ such that

- either $s_{k_h + l_1} = c_3(r_i, r_j)$ or $s_{k_h + l_1} = c_4(r_i, r_j)$,
- $s_{k_h + l_1 + l_2} = c_1(r_i)$, and
- $s_{k_h + l_1 + l_2 + l_3} = s_{k_i}$.

Consequently, we have $k_h < k_i$. The construction of $\langle r_i \rangle_{i \in I}$ from $\langle s_k \rangle_{k \in K}$ implies that $h < i$, which establishes Condition 2.

Only-if part. Let X be a consistent $<_X$-preserving answer set of $\Pi^\star$.

Define

$$\begin{array}{rcl} Y & = & \{head(r) \mid r \in \Gamma^X_{\Pi^\star}\} \\ & & \cup \{\mathsf{ap}(n_r) \mid r \in \Gamma^X_{\Pi^\star}\} \cup \{\mathsf{bl}(n_r) \mid r \notin \Gamma^X_{\Pi^\star}\} \\ & & \cup \{\mathsf{ok}(n_r) \mid r \in \Pi^\star\} \cup \{\mathsf{rdy}(n_r, n_{r'}) \mid r, r' \in \Pi^\star\}\ . \end{array}$$

We note the following useful relationship for X and Y.

Lemma 6
For any $L \in \mathscr{L}$, we have that $L \in X$ iff $L \in Y$.

Given that X is consistent, this implies that Y is consistent, too.

Lemma 7
We have that $<_Y = <_X$.

We must show that Y is an answer set of $\mathscr{T}(\Pi)$, that is, $Y = Cn(\mathscr{T}(\Pi)^Y)$.

'$\supseteq$' *part.* We start by showing that Y is closed under $\mathscr{T}(\Pi)^Y$. For this, we show for $r \in \mathscr{T}(\Pi)$ that $head(r^+) \in Y$ whenever $r^+ \in \mathscr{T}(\Pi)^Y$ with $body^+(r) \subseteq Y$.

1. Consider $a_1(r) : head(r) \leftarrow \mathsf{ap}(n_r) \in \mathscr{T}(\Pi)$. We have $(a_1(r))^+ = a_1(r)$. Clearly, $a_1(r) \in \mathscr{T}(\Pi)^Y$. By definition of Y, we have $head(r) \in Y$ whenever $\mathsf{ap}(n_r) \in Y$.
2. Consider $a_2(r) : \mathsf{ap}(n_r) \leftarrow \mathsf{ok}(n_r), body(r) \in \mathscr{T}(\Pi)$. Suppose $(a_2(r))^+ = \mathsf{ap}(n_r) \leftarrow \mathsf{ok}(n_r), body^+(r) \in \mathscr{T}(\Pi)^Y$ with $body^-(r) \cap Y = \emptyset$. Assume $\{\mathsf{ok}(n_r)\} \cup body^+(r) \subseteq Y$. From Lemma 6, we get $body^+(r) \subseteq X$. Accordingly, $body^-(r) \cap Y = \emptyset$ implies $body^-(r) \cap X = \emptyset$. Hence, we have $r \in \Gamma^X_{\Pi^\star}$, which implies $\mathsf{ap}(n_r) \in Y$ by definition of Y.
3. Consider $b_1(r, L^+) : \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), not\ L^+ \in \mathscr{T}(\Pi)$, and assume $(b_1(r, L^+))^+ = \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r) \in \mathscr{T}(\Pi)^Y$ with $L^+ \cap Y = \emptyset$. The latter and Lemma 6 imply that $L^+ \notin X$ for $L^+ \in body^+(r)$. Hence, $r \notin \Gamma^X_{\Pi^\star}$, implying that $\mathsf{bl}(n_r) \in Y$ by definition of Y.
4. Consider $b_2(r, L^-) : \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), L^- \in \mathscr{T}(\Pi)$. We have $(b_2(r, L^-))^+ = b_2(r, L^-)$. Clearly, $b_2(r, L^-) \in \mathscr{T}(\Pi)^Y$. Suppose $\{\mathsf{ok}(n_r), L^-\} \subseteq Y$. Lemma 6 implies that $L^- \in X$ for $L^- \in body^-(r)$. Hence, $r \notin \Gamma^X_{\Pi^\star}$, implying that $\mathsf{bl}(n_r) \in Y$ by definition of Y.
5. Consider $c_1(r) : \mathsf{ok}(n_r) \leftarrow \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \in \mathscr{T}(\Pi)$. We have $(c_1(r))^+ = c_1(r)$. Clearly, $c_1(r) \in \mathscr{T}(\Pi)^Y$. We trivially have $\mathsf{ok}(n_r) \in Y$ by definition of Y.
6. Consider $\{c_2(r, r'), c_3(r, r'), c_4(r, r')\} \subseteq \mathscr{T}(\Pi)$. Clearly, $head(c_i(r, r')) = \mathsf{rdy}(n_r, n_{r'})$ for $i = 2, 3, 4$. We trivially have $\mathsf{rdy}(n_r, n_{r'}) \in Y$ by definition of Y.
7. Consider $t(r, r', r'') : n_r \prec n_{r''} \leftarrow n_r \prec n_{r'}, \prec n_{r''} \in \mathscr{T}(\Pi)$. We have $(t(r, r', r''))^+ = t(r, r', r'')$. Clearly, $t(r, r', r'') \in \mathscr{T}(\pi)^Y$. Assume $\{n_r \prec n_{r'},\ n_{r'} \prec n_{r''}\} \subseteq Y$. From Lemma 6, we get $\{n_r \prec n_{r'}, n_{r'} \prec n_{r''}\} \subseteq X$. That is, $body^+(t(r, r', r''))^+ \subseteq X$ for $t(r, r', r'') \in \Pi^\star$. In analogy to what we have shown above in 2 and 1, we then obtain $\mathsf{ap}(n_{t(r,r',r'')}) \in Y$ and $head(t(r, r', r'')) \in Y$. We thus get $n_r \prec n_{r''} \in Y$.
8. Consider $as(r, r') : \neg(n_{r'} \prec n_r) \leftarrow n_r \prec n_{r'} \in \mathscr{T}(\Pi)$. We have $(as(r, r'))^+ = as(r, r')$. Clearly, $as(r, r') \in \mathscr{T}(\Pi)^Y$. Assume $(n_r \prec n_{r'}) \in Y$. Then, in analogy to 7, we get $\neg(n_{r'} \prec n_r) \in Y$.

Note that the closure of Y under $\mathscr{T}(\Pi)^Y$ shows that $Cn(\mathscr{T}(\Pi)^Y) \subseteq Y$.

'$\subseteq$' *part.* We must now show that Y is the smallest set being closed under $\mathscr{T}(\Pi)^Y$, or equivalently that $Y \subseteq Cn(\mathscr{T}(\Pi)^Y)$.

Since X is a $<_X$-preserving answer set of $\Pi^\star$, there is an enumeration $\langle r_i \rangle_{i \in I}$ of

$\Pi^\star$ satisfying all conditions given in Definition 3. We proceed by induction on $\langle r_i\rangle_{i\in I}$ and show that

$$\begin{aligned} & \{head(r_i), \mathsf{ap}(n_{r_i}) \mid r_i \in \Gamma^X_{\Pi^\star}, i\in I\} \\ \cup\ & \{\mathsf{bl}(n_{r_i}) \mid r_i \notin \Gamma^X_{\Pi^\star}, i\in I\} \\ \cup\ & \{\mathsf{ok}(n_{r_i}) \mid i\in I\} \cup \{\mathsf{rdy}(n_{r_i}, n_{r_j}) \mid i,j\in I\} \quad\subseteq\quad Cn(\mathscr{T}(\Pi)^Y)\ . \end{aligned}$$

Base. Consider $r_0 \in \Pi^\star$. Given that X is consistent, we have $(r_0, r) \notin <_X$ for all $r\in\Pi^\star$ by Condition 1. By Lemma 7, we thus have $(n_{r_0} \prec n_r) \notin Y$ for all $r\in\Pi^\star$. Consequently,

$$c_2(r_0,r)^+ = \mathsf{rdy}(n_{r_0}, n_r) \leftarrow\ \in \mathscr{T}(\Pi)^Y \text{ for all } r\in\Pi^\star.$$

Since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we get $\mathsf{rdy}(n_{r_0}, n_r) \in Cn(\mathscr{T}(\Pi)^Y)$ for all $r\in\Pi^\star$. Given that $\Pi^\star = \{r_1,\ldots,r_k\}$ and that

$$c_1(r_0) = c_1(r_0)^+ = \mathsf{ok}(n_{r_0}) \ \leftarrow\ \mathsf{rdy}(n_{r_0}, n_{r_1}),\ldots,\mathsf{rdy}(n_{r_0}, n_{r_k}) \in \mathscr{T}(\Pi)^Y,$$

the closure of $Cn(\mathscr{T}(\Pi)^Y)$ under $\mathscr{T}(\Pi)^Y$ implies furthermore that $\mathsf{ok}(n_{r_0}) \in Cn(\mathscr{T}(\Pi)^Y)$.

We distinguish two cases.

- Suppose $r_0 \in \Gamma^X_{\Pi^\star}$. Since $\langle r_i\rangle_{i\in I}$ satisfies Condition 3, we have $body^+(r_0) = \emptyset$. Also, $r_0 \in \Gamma^X_{\Pi^\star}$ implies that $body^-(r_0) \cap X = \emptyset$, from which we obtain by Lemma 6 that $body^-(r_0)\cap Y = \emptyset$. We thus obtain that

$$a_2(r_0) = a_2(r_0)^+ = \mathsf{ap}(n_{r_0}) \leftarrow \mathsf{ok}(n_{r_0}) \in \mathscr{T}(\Pi)^Y\ . \tag{A 2}$$

  We have shown above that $\mathsf{ok}(n_{r_0}) \in Cn(\mathscr{T}(\Pi)^Y)$. Accordingly, since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we obtain $\mathsf{ap}(n_{r_0}) \in Cn(\mathscr{T}(\Pi)^Y)$ because of (A 2).

  This, the closure of $Cn(\mathscr{T}(\Pi)^Y)$ under $\mathscr{T}(\Pi)^Y$, and the fact that

$$a_1(r_0) = a_1(r_0)^+ = head(r_0) \leftarrow \mathsf{ap}(n_{r_0}) \in \mathscr{T}(\Pi)^Y$$

  imply furthermore that $head(r_0) \in Cn(\mathscr{T}(\Pi)^Y)$.
- Otherwise, we have $r_0 \in \Pi^\star \setminus \Gamma^X_{\Pi^\star}$. Because r_0 cannot satisfy Condition 4b, since $body^-(r) \cap \{head(r_j) \mid j < 0\} = \emptyset$, we must have $body^+(r_0) \nsubseteq X$. Then, there is some $L^+ \in body^+(r_0)$ with $L^+ \notin X$. By Lemma 6, we also have $L^+ \notin Y$. Therefore,

$$b_1(r_0, L^+) = b_1(r_0, L^+)^+ = \mathsf{bl}(n_{r_0}) \leftarrow \mathsf{ok}(n_{r_0}) \in \mathscr{T}(\Pi)^Y\ . \tag{A 3}$$

  We have shown above that $\mathsf{ok}(n_{r_0}) \in Cn(\mathscr{T}(\Pi)^Y)$. Given this along with (A 3) and the fact that $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we obtain $\mathsf{bl}(n_{r_0}) \in Cn(\mathscr{T}(\Pi)^Y)$.

Step. Consider $r_i \in \Pi^\star$ and assume that our claim holds for all $r_j \in \Pi^\star$ with $j < i$. We start by providing the following lemma.

Lemma 8

Given the induction hypothesis, we have

(i) $\mathsf{ok}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$, and

(ii) $\{\mathsf{rdy}(n_{r_i}, n_{r_j}) \mid j\in I\} \subseteq Cn(\mathscr{T}(\Pi)^Y)$.

Proof of Lemma 8

We first prove that $\mathsf{rdy}(n_{r_i}, n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$ for all $j \in I$.

Suppose $r_i <_X r_j$. Since $\langle r_i \rangle_{i \in I}$ satisfies Condition 1, we have that $j < i$. Then, the induction hypothesis implies that either $\mathsf{ap}(n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$ or $\mathsf{bl}(n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$. According to Condition 2, there is some $r_k \in \Gamma^X_{\Pi^\star}$ with $k \in I$ such that $head(r_k) = (n_{r_i} \prec n_{r_j})$ and $k < i$. By the induction hypothesis, we get that $head(r_k) \in Cn(\mathscr{T}(\Pi)^Y)$. Hence, we obtain $(n_{r_i} \prec n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$. Clearly, we have that

$$\left\{ \begin{array}{lccc} c_3(r_i, r_j)^+ : & \mathsf{rdy}(n_{r_i}, n_{r_j}) & \leftarrow & (n_{r_i} \prec n_{r_j}), \mathsf{ap}(n_{r_j}) \\ c_4(r_i, r_j)^+ : & \mathsf{rdy}(n_{r_i}, n_{r_j}) & \leftarrow & (n_{r_i} \prec n_{r_j}), \mathsf{bl}(n_{r_j}) \end{array} \right\} \subseteq \mathscr{T}(\Pi)^Y .$$

Since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we obtain $\mathsf{rdy}(n_{r_i}, n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$. Therefore, we have shown that $\mathsf{rdy}(n_{r_i}, n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$ whenever $r_i <_X r_j$.

Contrariwise, suppose $(r_i, r_j) \notin <_X$. We get by Lemma 6 that $(n_{r_i} \prec n_{r_j}) \notin Y$. This implies that $c_2(r_i, r_j)^+ = \mathsf{rdy}(n_{r_i}, n_{r_j}) \leftarrow\ \in \mathscr{T}(\Pi)^Y$. Since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we obtain $\mathsf{rdy}(n_{r_i}, n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$.

Hence, we have $\mathsf{rdy}(n_{r_i}, n_{r_j}) \in Cn(\mathscr{T}(\Pi)^Y)$ for all $j \in I$.

Since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $c_1(r_i)^+ \in \mathscr{T}(\Pi)^Y$, we obtain $\mathsf{ok}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$. □

We now distinguish the following two cases.

- If $r_i \in \Gamma^X_{\Pi^\star}$, then $body^+(r_i) \subseteq \{head(r_j) \mid r_j \in \Gamma^X_{\Pi^\star}, j < i\}$ according to Condition 3. By the induction hypothesis, we hence obtain $body^+(r_i) \subseteq Cn(\mathscr{T}(\Pi)^Y)$. By Lemma 8, we have $\mathsf{ok}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$.
  Also, $r_i \in \Gamma^X_{\Pi^\star}$ implies $body^-(r_i) \cap X = \emptyset$. By Lemma 6, we thus have $body^-(r_i) \cap Y = \emptyset$. This implies

  $$a_2(r_i) = a_2(r_i)^+ = \mathsf{ap}(n_{r_i}) \leftarrow \mathsf{ok}(n_{r_i}), body^+(r_i) \in \mathscr{T}(\Pi)^Y . \qquad \text{(A 4)}$$

  As shown above, we have $body(a_2(r_i)^+) \subseteq Cn(\mathscr{T}(\Pi)^Y)$. Given (A 4) and the fact that $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we therefore get $\mathsf{ap}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$. Accordingly, we obtain $head(r_i) \in Cn(\mathscr{T}(\Pi)^Y)$ because of $a_1(r_i)^+ \in \mathscr{T}(\Pi)^Y$.
- Otherwise, we have $r_i \in \Pi^\star \setminus \Gamma^X_{\Pi^\star}$. According to Condition 4, we may distinguish the following two cases.
  - If $body^+(r_i) \nsubseteq X$, then there is some $L^+ \in body^+(r_i)$ with $L^+ \notin X$. By Lemma 6, we also have $L^+ \notin Y$. Therefore,

    $$b_1(r_i, L^+) = b_1(r_i, L^+)^+ = \mathsf{bl}(n_{r_i}) \leftarrow \mathsf{ok}(n_{r_i}) \in \mathscr{T}(\Pi)^Y . \qquad \text{(A 5)}$$

    By Lemma 8, we have $\mathsf{ok}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$. Given this along with (A 5) and the fact that $Cn(\mathscr{T}(\Pi)^Y)$ is closed under $\mathscr{T}(\Pi)^Y$, we obtain $\mathsf{bl}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$.
  - If $body^-(r) \cap \{head(r_j) \mid r_j \in \Gamma^X_{\Pi^\star}, j < i\} \neq \emptyset$, then there is some $L^- \in body^-(r_i)$ with $L^- \in \{head(r_j) \mid r_j \in \Gamma^X_{\Pi^\star}, j < i\}$. That is, $L^- = head(r_j)$ for some $r_j \in \Gamma^X_{\Pi^\star}$ with $j < i$. With the induction hypothesis, we then obtain $L^- \in Cn(\mathscr{T}(\Pi)^Y)$. By Lemma 8, we have $\mathsf{ok}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$.

Since $Cn(\mathscr{T}(\Pi)^Y)$ is closed under

$$b_2(r_i, L^-) = b_2(r_i, L^-)^+ = \mathsf{bl}(n_{r_i}) \leftarrow \mathsf{ok}(n_{r_i}), L^- \in \mathscr{T}(\Pi)^Y \ ,$$

we obtain $\mathsf{bl}(n_{r_i}) \in Cn(\mathscr{T}(\Pi)^Y)$. □

Proof of Theorem 4

Suppose Π contains no preference information. Obviously, Π can then be regarded both as a dynamically ordered program as well as a statically ordered program $(\Pi, <)$ with $< = \emptyset$. Hence, Theorems 2 and 3 imply that X is $<$-preserving iff X is $<_X$-preserving. This proves the equivalence of 2 and 1. Inspecting the conditions of Definition 1, it is easily seen that any answer set is trivially $<$-preserving for $< = \emptyset$. It follows that 2 is equivalent to 3. □

Proof of Theorem 5

Principle I. Let $(\Pi, <)$ be a statically ordered logic program, let X_1 and X_2 be two answer sets of Π generated by $\Gamma_\Pi^{X_1} = R \cup \{r_1\}$ and $\Gamma_\Pi^{X_2} = R \cup \{r_2\}$, respectively, where $r_1, r_2 \notin R$, and assume $r_1 < r_2$.

Suppose X_1 is a $<$-preserving answer set of Π. Clearly, both X_1 and X_2 must be consistent. Furthermore, we have that $r_2 \notin \Gamma_\Pi^{X_1}$, i.e., r_2 is defeated by X_1. On the other hand, given that $\Gamma_\Pi^{X_2} = R \cup \{r_2\}$ and since $r_2 \notin R$, it follows that $body^+(r_2) \subseteq \{head(r') \mid r' \in R\}$, which in turn implies that $body^+(r_2) \subseteq X_1$. Since $r_1 < r_2$ and X_1 is assumed to be $<$-preserving, we obtain that $body^-(r_2) \cap \{head(r'_k) \mid k < j_0\} \neq \emptyset$ for some enumeration $\langle r'_i \rangle_{i \in I}$ of $\Gamma_\Pi^{X_1}$ and some j_0 such that $r_1 = r'_{j_0}$. Hence, there is some $k_0 < j_0$ such that $head(r'_{k_0}) \in body^-(r_2)$. But $r'_{k_0} \in R$, so r_2 is defeated by X_2, which is a contradiction. It follows that X_1 is not $<$-preserving.

Principle II-S. Let $(\Pi, <)$ be a statically ordered logic program and let X be a $<$-preserving answer set of Π. Furthermore, let r be a rule where $body^+(r) \nsubseteq X$, and let $(\Pi \cup \{r\}, <')$ be a statically ordered logic program where $<'$ is a strict partial order which agrees with $<$ on rules from Π. We show that X is a $<'$-preserving answer set of $\Pi \cup \{r\}$.

First, it is rather obvious that X is an answer set of $\Pi \cup \{r\}$. Now consider an enumeration $\langle r_i \rangle_{i \in I}$ of Γ_Π^X satisfying Items 1–3 of Definition 1 with respect to $<$. Since $\Gamma_\Pi^X = \Gamma_\Pi^{X \cup \{r\}}$, $\langle r_i \rangle_{i \in I}$ is also an enumeration of $\Gamma_\Pi^{X \cup \{r\}}$. It remains to show that $\langle r_i \rangle_{i \in I}$ satisfies Items 1–3 of Definition 1 with respect to $<'$. Clearly, Condition 1 is unaffected by $<'$ and is thus still satisfied. Moreover, $<'$ agrees with $<$ on the rules of Π, so Condition 2 holds for $<'$ as well. Finally, Condition 3 holds in virtue of $r \notin \Gamma_\Pi^{X \cup \{r\}}$ and $body^+(r) \nsubseteq X$.

Principle II-D. Let Π be a (dynamically) ordered logic program, let X be a $<_X$-preserving answer set of $\Pi^\star$, and consider a rule r such that $body^+(r) \nsubseteq X$. We must show that X is an $<_X$-preserving answer set of $\Pi^\star \cup \{r\}$.

If $r \in \Pi^\star$, then Principle II-D holds trivially, so let us assume that $r \notin \Pi^\star$. Similar to the above, X is clearly an answer set of $\Pi^\star \cup \{r\}$. Let $\langle r_i \rangle_{i \in I}$ be an enumeration of $\Pi^\star$ satisfying Items 1–4 of Definition 3. Define $\langle r'_j \rangle_{j \in J}$ as the sequence starting with r and continuing with the sequence $\langle r_i \rangle_{i \in I}$. Clearly, $\langle r'_j \rangle_{j \in J}$ is an enumeration of $\Pi^\star \cup \{r\}$. Moreover, it satisfies Items 1–4 of Definition 3. To see this, we first note that, according to the definition of an ordered program, the term n_r does not occur in the rules of $\Pi^\star$ because r is assumed not to be a member of $\Pi^\star$. Hence, the relation $r'_i <_X r'_j$ holds at most for rules $r'_i, r'_j \in \Pi^\star$. Consequently, $\langle r'_j \rangle_{j \in J}$ obeys Items 1 and 2, because $\langle r_i \rangle_{i \in I}$ does. Furthermore, Item 3 holds, because $\Gamma^{X \cup \{r\}}_{\Pi^\star} = \Gamma^X_{\Pi^\star}$. Concerning the final condition of Definition 3, observe that $(\Pi^\star \cup \{r\}) \setminus \Gamma^{X \cup \{r\}}_{\Pi^\star} = (\Pi^\star \setminus \Gamma^X_{\Pi^\star}) \cup \{r\}$. Now, each $r' \in \Pi^\star \setminus \Gamma^X_{\Pi^\star}$ satisfies Condition 4, and r satisfies it as well, because $body^+(r) \not\subseteq X$. Therefore, Condition 4 is met. It follows that X is $<_X$-preserving. □

A.2 Proofs of section 5

Proof of Theorem 7

Let $(\Pi, <)$ be a statically ordered logic program over language $\mathscr{L}$.

We abbreviate $\mathscr{U}(\Pi) \cup \{(n_1 \prec n_2) \leftarrow \mid (r_1, r_2) \in <\}$ by $\mathscr{U}'(\Pi)$.

Only-if part. Let X be a consistent BE-preferred answer set of $(\Pi, <)$. By definition, X is then a standard answer set of Π. Define

$$\begin{aligned} Y \quad = \quad & \{head(r) \mid r \in \Gamma^X_\Pi\} \cup \{head(r') \mid r \in \Gamma^X_\Pi\} \\ & \cup \{\mathsf{ok}(n_r) \mid r \in \Pi\} \cup \{\mathsf{rdy}(n_r, n_{r'}) \mid r, r' \in \Pi\} \\ & \cup \{\mathsf{ap}(n_r) \mid r \in \Gamma^X_\Pi\} \cup \{\mathsf{bl}(n_r) \mid r \notin \Gamma^X_\Pi\} \\ & \cup \{(n_r \prec n_{r'}) \mid (r, r') \in <\} \cup \{\neg(n_r \prec n_{r'}) \mid (r', r) \in <\}. \end{aligned}$$

We note the following useful relationships for X and Y.

Lemma 9

For any $L \in \mathscr{L}$, we have

1. $L \in X$ iff $L \in Y$;
2. $L' \in X'$ iff $L' \in Y$; and
3. $L \in Y$ iff $L' \in Y$.

Given that X is consistent, this implies that Y is consistent, too.

Lemma 10

For $r, s \in \Pi$, we have $n_r \prec n_s \in Y$ iff $r < s$.

We must show that Y is an answer set of $\mathscr{U}'(\Pi)$, that is, $Y = Cn(\mathscr{U}'(\Pi)^Y)$.

'$\supseteq$' *part.* We start by showing that Y is closed under $\mathscr{U}'(\Pi)^Y$. For this, we show for $r \in \mathscr{U}'(\Pi)$ that $head(r^+) \in Y$ whenever $r^+ \in \mathscr{U}'(\Pi)^Y$ with $body^+(r) \subseteq Y$.

1. Consider $a_1(r) : head(r') \leftarrow \mathsf{ap}(n_r) \in \mathscr{U}'(\Pi)$. We have $(a_1(r))^+ = a_1(r)$. Clearly, $a_1(r) \in \mathscr{U}'(\Pi)^Y$. By definition of Y, we have $head(r') \in Y$ whenever $\mathsf{ap}(n_r) \in Y$.

2. Consider $a_2(r) : \mathsf{ap}(n_r) \leftarrow \mathsf{ok}(n_r), body(r), not\ body^-(r') \in \mathscr{U}'(\Pi)$, and suppose that $(a_2(r))^+ \in \mathscr{U}'(\Pi)^Y$, i.e., $(body^-(r) \cup body^-(r')) \cap Y = \emptyset$. Assume $\{\mathsf{ok}(n_r)\} \cup body^+(r) \subseteq Y$. From Condition 1 of Lemma 9 we get $body^+(r) \subseteq X$. Accordingly, $(body^-(r) \cup body^-(r')) \cap Y = \emptyset$ implies $body^-(r) \cap X = \emptyset$. Hence, we have $r \in \Gamma_\Pi^X$, which implies $\mathsf{ap}(n_r) \in Y$ by definition of Y.
3. Consider $b_1(r, L) : \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), not\ L, not\ L' \in \mathscr{U}'(\Pi)$, and suppose that $(b_1(r, L))^+ = \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r) \in \mathscr{U}'(\Pi)^Y$ with $\{L, L'\} \cap Y = \emptyset$. The latter and Condition 1 of Lemma 9 imply that $L \notin X$ for $L \in body^+(r)$. Hence, $r \notin \Gamma_\Pi^X$, implying that $\mathsf{bl}(n_r) \in Y$ by definition of Y.
4. Consider $b_2(r, K) : \mathsf{bl}(n_r) \leftarrow \mathsf{ok}(n_r), K, K' \in \mathscr{U}'(\Pi)$. We have $(b_2(r, K))^+ = b_2(r, K)$. Clearly, $b_2(r, K) \in \mathscr{U}'(\Pi)^Y$. Suppose $\{\mathsf{ok}(n_r), K, K'\} \subseteq Y$. Condition 1 of Lemma 9 implies that $K \in X$ for $K \in body^-(r)$. Hence, $r \notin \Gamma_\Pi^X$, implying that $\mathsf{bl}(n_r) \in Y$ by definition of Y.
5. Consider $c_1(r) : \mathsf{ok}(n_r) \leftarrow \mathsf{rdy}(n_r, n_{r_1}), \ldots, \mathsf{rdy}(n_r, n_{r_k}) \in \mathscr{U}'(\Pi)$. We have $(c_1(r))^+ = c_1(r)$. Clearly, $c_1(r) \in \mathscr{U}'(\Pi)^Y$. We trivially have $\mathsf{ok}(n_r) \in Y$ by definition of Y.
6. Consider the case that $\{c_2(r, s), c_3(r, s), c_4(r, s), c_5(r, s, J)\} \subseteq \mathscr{U}'(\Pi)$. Clearly, $head(c_i(r, s)) = \mathsf{rdy}(n_r, n_s)$ for $i = 2, \ldots, 5$. We trivially have $\mathsf{rdy}(n_r, n_s) \in Y$ by definition of Y.
7. Consider $d(r) : \leftarrow not\ \mathsf{ok}(n_r) \in \mathscr{U}'(\Pi)$. Suppose $(d(r))^+ \in \mathscr{U}'(\Pi)^Y$, i.e., $\mathsf{ok}(n_r) \cap Y = \emptyset$. This is impossible by definition of Y. Hence, $d(r)^+ \notin \mathscr{U}'(\Pi)^Y$.
8. Consider $t(r, s, t) : n_r \prec n_t \leftarrow n_r \prec n_s, n_s \prec n_t \in \mathscr{U}'(\Pi)$. Then, we have that $(t(r, s, t))^+ = t(r, s, t)$. Clearly, $t(r, s, t) \in \mathscr{U}'(\Pi)^Y$. Assume $\{n_r \prec n_s, n_s \prec n_t\} \subseteq Y$. By definition of Y, this implies $r < s$ and $s < t$. By transitivity of $<$, we then obtain $r < t$. Again, by definition of Y, we get $n_r \prec n_t \in Y$.
9. Consider $as(r, s) : \neg(n_s \prec n_r) \leftarrow n_r \prec n_s \in \mathscr{U}'(\Pi)$. We have $(as(r, s))^+ = as(r, s)$. Clearly, $as(r, s) \in \mathscr{U}'(\Pi)^Y$. Assume $n_r \prec n_s \in Y$. By definition of Y, this implies $r < s$. Again, by definition of Y, we then get $\neg(n_s \prec n_r) \in Y$.

Note that the closure of Y under $\mathscr{U}'(\Pi)^Y$ shows that $Y \supseteq Cn(\mathscr{U}'(\Pi)^Y)$.

'$\subseteq$' *part*. We must now show that Y is the smallest set being closed under $\mathscr{U}'(\Pi)^Y$, or equivalently that $Y \subseteq Cn(\mathscr{U}'(\Pi)^Y)$.

First, we show that

$$\{head(r) \mid r \in \Gamma_\Pi^X\} \subseteq Cn(\mathscr{U}'(\Pi)^Y) \ . \tag{A 6}$$

Observe that $\{head(r) \mid r \in \Gamma_\Pi^X\} = \bigcup_{i \geqslant 0} T^i_{\Pi^X} \emptyset$. Given this, we proceed by induction and show that $T^i_{\Pi^X} X \subseteq Cn(\mathscr{U}'(\Pi)^Y)$ for $i \geqslant 0$.

Base. Trivial, since $T^0_{\Pi^X} \emptyset = \emptyset$.

Step. Assume $T^j_{\Pi^X} \emptyset \subseteq Cn(\mathscr{U}'(\Pi)^Y)$ for $0 \leqslant j \leqslant i$, and consider $head(r^+) \in T^{i+1}_{\Pi^X} \emptyset$ for some $r^+ \in \Pi^X$.

Given that $r \in \mathscr{U}'(\Pi)$ and that $X = Y \cap \mathscr{L}$ by Condition 1 of Lemma 9, we also have $r^+ \in \mathscr{U}'(\Pi)^Y$.

Now, $head(r^+) \in T^{i+1}_{\Pi^X} \emptyset$ implies that $body(r^+) \subseteq T^i_{\Pi^X} \emptyset$. Then, the induction hypothesis provides us with $body(r^+) \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. The fact that $Cn(\mathscr{U}'(\Pi)^Y)$ is closed under $\mathscr{U}'(\Pi)^Y$ implies that $head(r^+) \in Cn(\mathscr{U}'(\Pi)^Y)$.

Second, we show that

$$\{(n_r \prec n_{r'}) \mid (r,r') \in <\} \cup \{\neg(n_r \prec n_{r'}) \mid (r,r') \notin <\} \subseteq Cn(\mathscr{U}'(\Pi)^Y). \tag{A 7}$$

By definition, we have $\{(n_r \prec n_{r'}) \leftarrow \mid (r,r') \in <\} \subseteq \mathscr{U}'(\Pi)$ as well as $\{(n_r \prec n_{r'}) \leftarrow \mid (r,r') \in <\} \subseteq \mathscr{U}'(\Pi)^Y$. From this, we obviously get $\{(n_r \prec n_{r'}) \mid (r,r') \in <\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. On the other hand, we have $as(r,r') \in \mathscr{U}'(\Pi)$ and clearly also $as(r,r')^+ \in \mathscr{U}'(\Pi)^Y$. The fact that $Cn(\mathscr{U}'(\Pi)^Y)$ is closed under $\mathscr{U}'(\Pi)^Y$ implies that $head(as(r,r')^+) \in Cn(\mathscr{U}'(\Pi)^Y)$ whenever $body(as(r,r')^+) \in Cn(\mathscr{U}'(\Pi)^Y)$. That is, $\neg(n_{r'} \prec n_r) \in Cn(\mathscr{U}'(\Pi)^Y)$ whenever $n_r \prec n_{r'} \in Cn(\mathscr{U}'(\Pi)^Y)$. Given that we have just shown that $\{(n_r \prec n_{r'}) \mid (r,r') \in <\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$ holds, we get $\{\neg(n_{r'} \prec n_r) \mid (r,r') \in <\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$.

Third, we show that

$$\begin{array}{ll} & \{head(r'), \mathsf{ap}(n_r) \mid r \in \Gamma_\Pi^X\} \\ \cup & \{\mathsf{bl}(n_r) \mid r \notin \Gamma_\Pi^X\} \\ \cup & \{\mathsf{ok}(n_r) \mid r \in \Pi\} \cup \{\mathsf{rdy}(n_r, n_{r'}) \mid r, r' \in \Pi\} \quad \subseteq \quad Cn(\mathscr{U}'(\Pi)^Y)\,. \end{array}$$

We do so by induction on $\ll$, the total order on Π inducing X according to Definition 6.

Base. Let $r_0 \in \Pi$ be the $\ll$-greatest rule. In analogy to the base case in the '$\subseteq$'-part of the proof of the only-if direction of Theorem 3, we can prove that $\mathsf{ok}(n_{r_0}) \in Cn(\mathscr{U}'(\Pi)^Y)$. Given this, we can show that $\mathsf{ap}(n_{r_0}) \in Cn(\mathscr{U}'(\Pi)^Y)$ or $\mathsf{bl}(n_{r_0}) \in Cn(\mathscr{U}'(\Pi)^Y)$ holds in analogy to the (more general) proof carried out in the induction step below.

Step. Consider $r \in \Pi$ and assume that our claim holds for all $s \in \Pi$ such that $r \ll s$. We start by providing the following lemma.

Lemma 11

Given the induction hypothesis, we have

(i) $\mathsf{ok}(n_r) \in Cn(\mathscr{U}'(\Pi)^Y)$ and
(ii) $\{\mathsf{rdy}(n_r, n_s) \mid s \in \Pi\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$.

Proof of Lemma 11

First, we prove $\{\mathsf{rdy}(n_r, n_s) \mid s \in \Pi\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$.
Consider $\mathsf{rdy}(n_r, n_s) \in Y$ for some $s \in \Pi$. We distinguish the following cases.

- Assume that $head(s) \in X$ and $body^-(s) \cap X \neq \emptyset$. That is, $J \in X$ for some $J \in body^-(s)$. We have shown in (A 6) that $X \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. Hence, $\{head(s), J\} \subseteq X$ implies $\{head(s), J\} \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. That is, $body(c_5(r,s,J)^+) \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. We clearly have $c_5(r,s,J)^+ \in \mathscr{U}'(\Pi)^Y$. Since $Cn(\mathscr{U}'(\Pi)^Y)$ is closed under $\mathscr{U}'(\Pi)^Y$, we have $head(c_5(r,s,J)^+) \in Cn(\mathscr{U}'(\Pi)^Y)$. That is, $\mathsf{rdy}(n_r, n_s) \in Cn(\mathscr{U}'(\Pi)^Y)$.
- Assume $r \not\ll s$. Since Y is consistent, we thus get $(n_r \prec n_s) \notin Y$. Hence, we have $c_2(r,s)^+ \in \mathscr{U}'(\Pi)^Y$. Trivially, we have $body(c_2(r,s)^+) = \emptyset \subseteq Cn(\mathscr{U}'(\Pi)^Y)$. Since $Cn(\mathscr{U}'(\Pi)^Y)$ is closed under $\mathscr{U}'(\Pi)^Y$, we have $head(c_2(r,s)^+) \in Cn(\mathscr{U}'(\Pi)^Y)$. That is, $\mathsf{rdy}(n_r, n_s) \in Cn(\mathscr{U}'(\Pi)^Y)$.

- Otherwise, $r \ll s$ holds. By the induction hypothesis, we have

  $$\{\mathsf{ap}(n_s) \mid r \ll s, s \in \Gamma_\Pi^X\} \cup \{\mathsf{bl}(n_s) \mid r \ll s, s \notin \Gamma_\Pi^X\} \subseteq Cn(\mathcal{U}'(\Pi)^Y)\ .$$

  Moreover, we get $(n_r \prec n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$ by what we have shown in (A 7). We distinguish the following two cases.

  — Assume $body^-(s) \cap X = \emptyset$. If $body^+(s) \subseteq X$, then $s \in \Gamma_\Pi^X$. By the induction hypothesis, we get $\mathsf{ap}(n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$. Therefore, we have $body(c_3(r,s)^+) = \{(n_r \prec n_s), \mathsf{ap}(n_s)\} \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. Clearly, we have $c_3(r,s)^+ \in \mathcal{U}'(\Pi)^Y$. Since $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we have $head(c_3(r,s)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $\mathsf{rdy}(n_r,n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$. If $body^+(s) \not\subseteq X$, then $s \notin \Gamma_\Pi^X$, as dealt with next.

  — Assume $head(s) \notin X$. That is, $s \notin \Gamma_\Pi^X$. By the induction hypothesis, we get $\mathsf{bl}(n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$. Therefore, we have $body(c_4(r,s)^+) = \{(n_r \prec n_s), \mathsf{bl}(n_s)\} \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. Clearly, we have $c_4(r,s)^+ \in \mathcal{U}'(\Pi)^Y$. Since $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we have $head(c_4(r,s)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $\mathsf{rdy}(n_r,n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$.

We have shown that $\mathsf{rdy}(n_r,n_s) \in Cn(\mathcal{U}'(\Pi)^Y)$ for all $s \in \Pi$. This implies that $body(c_1(r)^+) \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. Clearly, we have $c_1(r)^+ \in \mathcal{U}'(\Pi)^Y$. Since $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we have $head(c_1(r)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $\mathsf{ok}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$. □

We distinguish the following three cases.

1. Assume $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. That is, $r \in \Gamma_\Pi^X$.
   From $body^-(r) \cap X = \emptyset$, we deduce $body^-(r) \cap Y = \emptyset$, by Condition 1 of Lemma 9. Moreover, Condition 3 of Lemma 9 gives $body^-(r') \cap Y = \emptyset$. Therefore, $a_2(r)^+ \in \mathcal{U}'(\Pi)^Y$. We have just shown in (A 6) that $X \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. This and $body^+(r) \subseteq X$ imply that $body^+(r) \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. From Lemma 11, we get $\mathsf{ok}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$. We thus have $body(a_2(r)^+) \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. Since $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we have $head(a_2(r)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $\mathsf{ap}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$.
   Clearly, $a_1(r)^+ \in \mathcal{U}'(\Pi)^Y$. Given $\mathsf{ap}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$ and the fact that $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we have $head(a_1(r)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $head(r') \in Cn(\mathcal{U}'(\Pi)^Y)$.
2. Assume $body^+(r) \subseteq X$ and $body^-(r) \cap X \neq \emptyset$.
   By Theorem 9, there is then some rule $r^* \in \Gamma_\Pi^X$ such that $r \ll r^*$ and $head(r^*) \in body^-(r)$. Clearly, $r^* \in \Gamma_\Pi^X$ implies that $head(r^*) \in X$. We have shown in (A 6) that $X \subseteq Cn(\mathcal{U}'(\Pi)^Y)$. Accordingly, $head(r^*) \in Cn(\mathcal{U}'(\Pi)^Y)$. By the induction hypothesis, we also have $head(r^*)' \in Cn(\mathcal{U}'(\Pi)^Y)$. From Lemma 11, we obtain $\mathsf{ok}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$. We thus have $body(b_2(r,K)^+) = \{\mathsf{ok}(n_r), K, K'\} \subseteq Cn(\mathcal{U}'(\Pi)^Y)$ for $K = head(r^*)$. Clearly, we have $b_2(r,K)^+ \in \mathcal{U}'(\Pi)^Y$. Given that $Cn(\mathcal{U}'(\Pi)^Y)$ is closed under $\mathcal{U}'(\Pi)^Y$, we obtain that $head(b_2(r,K)^+) \in Cn(\mathcal{U}'(\Pi)^Y)$. That is, $\mathsf{bl}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$.
3. Assume $body^+(r) \not\subseteq X$. That is, there is some $L \in body^+(r)$ such that $L \notin X$.
   By Lemma 9, we then also have $L \notin Y$ and $L' \notin Y$. We then have $b_1(r,L)^+ \in \mathcal{U}'(\Pi)^Y$. Given that $\mathsf{ok}(n_r) \in Cn(\mathcal{U}'(\Pi)^Y)$ by Lemma 11, and since $Cn(\mathcal{U}'(\Pi)^Y)$

is closed under $\mathscr{U}'(\Pi)^Y$, we obtain that $head(b_1(r,L)^+) \in Cn(\mathscr{U}'(\Pi)^Y)$. That is, $\mathsf{bl}(n_r) \in Cn(\mathscr{U}'(\Pi)^Y)$.

This completes our proof of $Y \subseteq Cn(\mathscr{U}'(\Pi)^Y)$.

In all, we have thus shown that $Y = Cn(\mathscr{U}'(\Pi)^Y)$. That is, Y is an answer set of $\mathscr{U}'(\Pi)$.

If part. Let Y be a consistent answer set of $\mathscr{U}'(\Pi)$. We show that $X = Y \cap \mathscr{L}_{\mathscr{A}}$ is a BE-preferred answer set of $(\Pi, <)$.

The definition of $\mathscr{U}'(\Pi)$ induces the following properties for X and Y.

Lemma 12
For any $L \in \mathscr{L}$, we have that $L \in X$ iff $L \in Y$.

Given that Y is consistent, this implies that X is consistent, too.

Lemma 13
For $r, s \in \Pi$, we have $n_r \prec n_s \in Y$ iff $r < s$.

Consider $r \in \Pi$ such that $body^+(r) \subseteq X$ and $head(r) \notin X$. To show that X is BE-preferred, we must prove, according to Theorem 9, that there is some rule $r^* \in \Pi$ such that (i) $r^* \in \Gamma_\Pi^X$, (ii) $head(r^*) \in body^-(r)$, and (iii) $r \ll r^*$.

Since X is a standard answer set of Π (by Property 1 of Proposition 4), our choice of r implies that $body^-(r) \cap X \neq \emptyset$. By Lemma 12, the latter condition implies, in turn, $body^-(r) \cap Y \neq \emptyset$. Hence, $a_2(r)^+ \notin \mathscr{U}'(\Pi)^Y$. And so, $\mathsf{ap}(n_r) \notin Y$. Consequently, we obtain $\mathsf{bl}(n_r) \in Y$ from Condition 2 of Proposition 5 and the fact that Y is consistent. Moreover, $body^+(r) \subseteq X \subseteq Y$ implies $b_1(r,L)^+ \notin \mathscr{U}'(\Pi)^Y$ for all $L \in body^+(r)$. Since $\mathsf{bl}(n_r) \in Y$, we must therefore have $b_2(r,K) \in \Gamma^Y_{\mathscr{U}'(\Pi)}$ for some $K \in body^-(r)$. That is, $body^+(b_2(r,K)) = \{\mathsf{ok}(n_r), K, K'\} \subseteq Y$. Hence, there is some $r^* \in \Pi$ with $head(r^*) = K$, establishing Condition (ii).

More precisely, we have $head(r^*)' = K'$ such that $\mathsf{ap}(n_{r^*}) \in Y$. That is, $\{a_1(r^*), a_2(r^*)\} \subseteq \Gamma^Y_{\mathscr{U}'(\Pi)}$. In fact, $a_2(r^*) \in \Gamma^Y_{\mathscr{U}'(\Pi)}$ implies that $body^+(r^*) \subseteq Y$ and $body^-(r^*) \cap Y \neq \emptyset$ holds. With Lemma 12, we get furthermore that $body^+(r^*) \subseteq X$ and $body^-(r^*) \cap X \neq \emptyset$. That is, $r^* \in \Gamma_\Pi^X$, establishing Condition (i).

Because Y is an answer set of $\mathscr{U}'(\Pi)$ there is some minimal k such that $\{\mathsf{ok}(n_r), K, K'\} \subseteq T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$. By minimality of k, we have $\mathsf{bl}(n_r) \notin T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$. Obviously, we also have $\mathsf{ap}(n_r) \notin T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$ because $\mathsf{ap}(n_r) \notin Y$. On the other hand, for deriving K', we must have $\mathsf{ap}(n_{r^*}) \in T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$. This implies that $\mathsf{ok}(n_{r^*}) \in T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$ and, in particular, that $\mathsf{rdy}(n_{r^*}, n_r) \in T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$. Now, observe that $c_5(n_{r^*}, n_r, J) \notin \Gamma^Y_{\mathscr{U}'(\Pi)}$ because $body^-(r) \cap Y \neq \emptyset$, as shown above. Also, we have just shown that $\mathsf{ap}(n_r) \notin T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$ and $\mathsf{bl}(n_r) \notin T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$. Therefore, we must have $\mathsf{rdy}(n_{r^*}, n_r) \in T^k_{\mathscr{U}'(\Pi)^Y}\emptyset$ because of $c_2(n_{r^*}, n_r) \in \Gamma^Y_{\mathscr{U}'(\Pi)}$. That is, $c_2(n_{r^*}, n_r)^+ \in \mathscr{U}'(\Pi)^Y$ which implies that $(n_{r^*} \prec n_r) \notin Y$. This and Lemma 13 imply that $r^* \not< r$. That is, either $r < r^*$ holds or neither $r^* \not< r$ nor $r \not< r^*$ is true. In either case, there is some total order $\ll$ extending $<$ with $r \ll r^*$. This establishes Condition (iii).

In all, we therefore obtain with Theorem 9 that X is a BE-preferred answer set of Π. $\square$

Proof of Proposition 4

Let Π be an ordered logic program over $\mathscr{L}$ and let X be a consistent answer set of $\mathscr{U}(\Pi)$.

1. The consistency of $X \cap \mathscr{L}$ follows from that of X.
   Since X is an answer set of $\mathscr{U}(\Pi)$, we have $X = Cn(\mathscr{U}(\Pi)^X)$. That is,
   $$X = Cn(\Pi^X \cup (\mathscr{U}(\Pi) \setminus \Pi)^X)\ .$$
   Observe that $\Pi^X = \Pi^{X \cap \mathscr{L}}$ and $head((\mathscr{U}(\Pi) \setminus \Pi)^X) \cap \mathscr{L} = \emptyset$. Consequently, we have $X \cap \mathscr{L} = Cn(\Pi^{X \cap \mathscr{L}})$.
2. Consider $L \in \mathscr{L}$. We show $L \in X$ iff $L' \in X$.

   If part. Suppose $L' \in X$. Then, there is some $r \in \Pi$ such that $L = head(r)$ and $\mathsf{ap}(n_r) \in X$. From the latter, we get that $a_2(r) \in \Gamma^X_{\mathscr{U}(\Pi)}$ must hold, which in turn implies $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. Hence, $L \in X$.

   Only-if part. Assume $L \in X$. Since $L \in \mathscr{L}$, there is some $r \in \Pi$ such that $r \in \Gamma^X_{\mathscr{U}(\Pi)}$. Suppose $L' \notin X$. Then, $\mathsf{ap}(n_r) \notin X$. Since $\mathsf{ok}(n_r) \in X$ (by Condition 1 of Proposition 5) and both $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$ holds (by condition $r \in \Gamma^X_{\mathscr{U}(\Pi)}$), it follows that there is some $K' \in body^-(r') \cap X$. This in turn implies that there exists some $s \in \Pi$ such that $K = head(s)$ (and thus $K' = head(s')$) and $a_2(s) \in \Gamma^X_{\mathscr{U}(\Pi)}$. From the last condition we get $body^+(s) \subseteq X$ and $body^-(s) \cap X = \emptyset$, and therefore $K = head(s) \in X$. Thus, $K \in X \cap body^-(r)$, contradicting $r \in \Gamma^X_{\mathscr{U}(\Pi)}$.
3. This property is a simple consequence of the observation that $body(a_2(r)) = body(r) \cup body^-(r')$, together with Item 2. The details follow.

   If part. Since $body(r) \subseteq body(a_2(r))$, the conditions $body^+(a_2(r)) \subseteq X$ and $body^-(a_2(r)) \cap X = \emptyset$ yield $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$. So, $a_2(r) \in \Gamma^X_{\mathscr{U}(\Pi)}$ implies $r \in \Pi \cap \Gamma^X_{\mathscr{U}(\Pi)}$.

   Only-if part. From Item 2, it holds that $body^-(r) \cap X = \emptyset$ implies $body^-(r') \cap X = \emptyset$. Thus, using the relation $body(a_2(r)) = body(r) \cup body^-(r')$, we get that the two conditions $body^+(r) \subseteq X$ and $body^-(r) \cap X = \emptyset$ jointly imply $body^+(a_2(r)) \subseteq X$ and $body^-(a_2(r)) \cap X = \emptyset$. Therefore, $r \in \Pi \cap \Gamma^X_{\mathscr{U}(\Pi)}$ only if $a_2(r) \in \Gamma^X_{\mathscr{U}(\Pi)}$.
4. Consider $r \in \Pi$.

   If part. Assume $b_1(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ or $b_2(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ for some $L \in body^+(r) \cup body^-(r)$.

   — Suppose $b_1(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$. Then, $L \notin X$, for $L \in body^+(r)$. Hence, $body^+(r) \not\subseteq X$ and therefore $r \notin \Gamma^X_{\mathscr{U}(\Pi)}$.
   — Suppose $b_2(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$. Then, $L \in X$, for $L \in body^-(r)$. Hence, $body^-(r) \cap X \neq \emptyset$ and therefore $r \notin \Gamma^X_{\mathscr{U}(\Pi)}$.

Only-if part. Suppose $r \notin \Gamma^X_{\mathscr{U}(\Pi)}$. There are two cases to distinguish.

— $body^+(r) \nsubseteq X$: Then, there is some $L \in body^+(r)$ such that $L \notin X$. According to Item 2, we also have $L' \notin X$. Furthermore, by Condition 1 of Proposition 5, we have $\mathsf{ok}(n_r) \in X$. So, $body^+(b_1(r, L)) \subseteq X$ and $body^-(b_1(r, L)) \cap X = \emptyset$. This means that $b_1(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ holds.

— $body^-(r) \cap X \neq \emptyset$: Then, there is some $L \in body^-(r)$ such that $L \in X$. Invoking Item 2 again, we get $L' \in X$. Thus, $b_2(r, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$.

5. Suppose $c_5(r, s, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$. Then, $L \in X$ for $L \in body^-(s)$. By Item 2, we also have $L' \in X$. Furthermore, $\mathsf{ok}(n_r) \in X$, by Condition 1 of Proposition 5. It follows that $b_2(s, L) \in \Gamma^X_{\mathscr{U}(\Pi)}$ holds. □

Proof of Proposition 5
Analogous to the proof of Proposition 2. □

Proof of Theorem 8
Let $(\Pi, <)$ be a statically ordered logic program and let X be a consistent answer set of

$$\mathscr{U}'(\Pi) \;=\; \mathscr{U}(\Pi) \cup \{(n_1 \prec n_2) \leftarrow \mid (r_1, r_2) \in <\}\;.$$

Furthermore, let

$$\hat{\Pi} \;=\; \Pi \setminus \{r \in \Pi \mid head(r) \in X, body^-(r) \cap X \neq \emptyset\}$$

and let $\langle s_i \rangle_{i \in I}$ be some grounded enumeration of $\Gamma^X_{\mathscr{U}'(\Pi)}$.

Lemma 14
Given the above prerequisites, we have for all $r, r' \in \hat{\Pi}$:
If $r < r'$, then $j < i$ for $s_i = c_1(r)$ and $s_j = c_1(r')$.

Proof of Lemma 14
Analogous to the proof of Theorem 1. □

By Proposition 5, we have $\mathsf{ok}(r) \in X$ for all $r \in \Pi$. Therefore, we also have $c_1(r) \in \Gamma^X_{\mathscr{U}'(\Pi)}$ for all $r \in \Pi$. Now, define for all $r_1, r_2 \in \hat{\Pi}$ that $r_1 \ll r_2$ if $j < i$ for $s_i = c_1(r_1)$ and $s_j = c_1(r_2)$. By definition, $\ll$ is a total ordering on $\hat{\Pi}$. Furthermore, $r_1 < r_2$ implies $r_1 \ll r_2$. That is, $(< \cap (\hat{\Pi} \times \hat{\Pi})) \subseteq \ll$. □

References

Baader, F. and Hollunder, B. (1993) How to prefer more specific defaults in terminological default logic. In: Bajcsy, R. (ed.), *Proceedings 13th International Joint Conference on Artificial Intelligence* (*IJCAI'93*), pp. 669–674. Morgan Kaufmann.

Benferhat, S., Cayrol, C., Dubois, D., Lang, J. and Prade, H. (1993) Inconsistency management and prioritized syntax-based entailment. In: Bajcsy, R. (ed.), *Proceedings 13th International Joint Conference on Artificial Intelligence* (*IJCAI'93*), pp. 640–647. Morgan Kaufmann.

Besnard, Ph., Mercer, R. and Schaub, T. (2002) Optimality theory via default logic. In: Benferhat, S. and Giunchiglia, E. (eds.), *Proceedings 9th International Workshop on Non-Monotonic Reasoning*, pp. 77–83.

Brewka, G. (1989) Preferred subtheories: An extended logical framework for default reasoning. *Proceedings 11th International Joint Conference on Artificial Intelligence* (*IJCAI'89*), pp. 1043–1048. Morgan Kaufmann.

Brewka, G. (1994) Adding priorities and specificity to default logic. In: Pereira, L. and Pearce, D. (eds.), *European Workshop on Logics in Artificial Intelligence* (*JELIA'94*)*: Lecture Notes in Artificial Intelligence 838*, pp. 247–260. Springer-Verlag.

Brewka, G. (1996) Well-founded semantics for extended logic programs with dynamic preferences. *J. Artif. Intell. Res.* **4**, 19–36.

Brewka, G. and Eiter, T. (1998) Preferred answer sets for extended logic programs. In: Cohn, A., Schubert, L. and Shapiro, S. (eds.), *Proceedings 6th International Conference on the Principles of Knowledge Representation and Reasoning* (*KR'98*), pp. 86–97. Morgan Kaufmann.

Brewka, G. and Eiter, T. (1999) Preferred answer sets for extended logic programs. *Artif. Intell.* **109**(1–2), 297–356.

Brewka, G. and Eiter, T. (2000) Prioritizing default logic. In: Hölldobler, St. (ed.), *Intellectics and Computational Logic – Papers in Honour of Wolfgang Bibel*, pp. 27–45. Kluwer Academic.

Delgrande, J. and Schaub, T. (1997) Compiling reasoning with and about preferences into default logic. In: Pollack, M. (ed.), *Proceedings 15th International Joint Conference on Artificial Intelligence* (*IJCAI'97*), pp. 168–174. Morgan Kaufmann.

Delgrande, J. and Schaub, T. (2000a) Expressing preferences in default logic. *Artif. Intell.* **123**(1-2), 41–87.

Delgrande, J. and Schaub, T. (2000b) The role of default logic in knowledge representation. In: Minker, J. (ed.), *Logic-Based Artificial Intelligence*, pp. 107–126. Kluwer Academic.

Delgrande, J., Schaub, T. and Tompits, H. (2000a) A compilation of Brewka and Eiter's approach to prioritization. In: Ojeda-Aciego, M., Guzmán, I., Brewka, G. and Pereira, L. (eds.), *Proceedings 8th European Workshop on Logics in Artificial Intelligence* (*JELIA 2000*)*: Lecture Notes in Artificial Intelligence 1919*, pp. 376–390. Springer-Verlag.

Delgrande, J., Schaub, T. and Tompits, H. (2000b) A compiler for ordered logic programs. In: Baral, C. and Truszczyński, M. (eds.), *Proceedings 8th International Workshop on Non-Monotonic Reasoning*. `arXiv.org` e-Print archive. System Abstract.

Delgrande, J., Schaub, T. and Tompits, H. (2000c) Logic programs with compiled preferences. In: Baral, C. and Truszczyński, M. (eds.), *Proceedings 8th International Workshop on Non-Monotonic Reasoning*. `arXiv.org` e-Print archive.

Delgrande, J., Schaub, T. and Tompits, H. (2000d) Logic programs with compiled preferences. In: Horn, W. (ed.), *Proceedings 14th European Conference on Artificial Intelligence* (*ECAI 2000*), pp. 392–398. IOS Press.

Delgrande, J., Schaub, T. and Tompits, H. (2001) plp: A generic compiler for ordered logic programs. In: Eiter, T., Faber, W. and Truszczyński, M. (eds.), *Proceedings 6th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2001*)*: Lecture Notes in Artificial Intelligence 2173*, pp. 411–415. Springer-Verlag.

Doyle, J. and Wellman, M. (1991) Impediments to universal preference-based default theories. *Artif. Intell.* **49**(1-3), 97–128.

Eiter, T. and Gottlob, G. (1995) The complexity of logic-based abduction. *J. ACM*, **42**, 3–42.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G. and Scarcello, F. (1997) A deductive system for nonmonotonic reasoning. In: Dix, J., Furbach, U. and Nerode, A. (eds.), *Proceedings 4th International Conference on Logic Programming and Non-Monotonic Reasoning* (*LPNMR'97*)*: Lecture Notes in Artificial Intelligence 1265*, pp. 363–374. Springer-Verlag.

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. (2002) A generic approach for knowledge-based information-site selection. *Proceedings 8th International Conference on the Principles of Knowledge Representation and Reasoning* (*KR 2002*), pp. 459–469. Morgan Kaufmann.

Geffner, H. and Pearl, J. (1992) Conditional entailment: Bridging two approaches to default reasoning. *Artif. Intell.* **53**(2-3), 209–244.

Gelfond, M. and Lifschitz, V. (1988) The stable model semantics for logic programming. *Proceedings 5th International Conference on Logic Programming* (*ICLP'88*), pp. 1070–1080. The MIT Press.

Gelfond, M. and Lifschitz, V. (1991) Classical negation in logic programs and deductive databases. *New Generat. Comput.* **9**, 365–385.

Gelfond, M. and Son, T. (1997) Reasoning with prioritized defaults. In: Dix, J., Pereira, L. and Przymusinski, T. (eds.), *Third International Workshop on Logic Programming and Knowledge Representation: Lecture Notes in Computer Science 1471*, pp. 164–223. Springer-Verlag.

Gelfond, M., Przymusinska, H. and Przymusinski, T. (1989) On the relationship between circumscription and negation as failure. *Artif. Intell.*, **38**(1), 75–94.

Gordon, T. (1993) *The pleading game: An artificial intelligence model of procedural justice*. Dissertation, Technische Hochschule Darmstadt, Germany.

Kager, R. (1999) *Optimality Theory: A textbook*. Cambridge University Press.

Konolige, K. (1988) Hierarchic autoepistemic theories for nonmonotonic reasoning. *Proceedings 7th National Conference on Artificial Intelligence* (*AAAI'88*), pp. 439–443. Morgan Kaufmann.

Lifschitz, V. (1996) Foundations of logic programming. In: Brewka, G. (ed.), *Principles of Knowledge Representation*, pp. 69–127. CSLI Publications.

McCarthy, J. (1986) Applications of circumscription to formalizing common-sense knowledge. *Artif. Intell.* **28**, 89–116.

Nebel, B. (1998) How hard is it to revise a belief base? In: Dubois, D. and Prade, H. (eds.), *Handbook of Defeasible Reasoning and Uncertainty Management Systems, Volume 3: Belief Change*. Kluwer Academic.

Niemelä, I. and Simons, P. (1997) Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In: Dix, J., Furbach, U. and Nerode, A. (eds.), *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'97*)*: Lecture Notes in Artificial Intelligence 1265*, pp. 420–429. Springer-Verlag.

Papadimitriou, C. and Sideri, M. (1994) Default theories that always have extensions. *Artif. Intell.* **69**, 347–357.

Prince, A. and Smolensky, P. (1993) *Optimality theory: Constraint interaction in generative grammar*. Technical report, University of Colorado.

Reiter, R. (1980) A logic for default reasoning. *Artif. Intell.* **13**(1-2), 81–132.

Rintanen, J. (1994) Prioritized autoepistemic logic. In: MacNish, C., Pearce, D. and Pereira, L. M. (eds.), *Proceedings of the European Workshop on Logics in Artificial Intelligence* (*JELIA'94*)*: Lecture Notes in Artificial Intelligence 838*, pp. 232–246. Springer-Verlag.

Rintanen, J. (1998a) Complexity of prioritized default logics. *J. Artif. Intell. Res.* **9**, 423–461.

Rintanen, J. (1998b) Lexicographic priorities in default logic. *Artif. Intell.* **106**, 221–265.

Sakama, C. and Inoue, K. (1996) Representing priorities in logic programs. In: Maher, M. (ed.), *Proceedings Joint International Conference and Symposium on Logic Programming*, pp. 82–96. Cambridge: The MIT Press.

Sakama, C. and Inoue, K. (2000) Prioritized logic programming and its application to commonsense reasoning. *Artif. Intell.* **123**(1-2), 185–222.

Schaub, T. and Wang, K. (2001a) A comparative study of logic programs with preference.

In: Nebel, B. (ed.), *Proceedings 17th International Joint Conference on Artificial Intelligence* (*IJCAI 2001*), pp. 597–602. Morgan Kaufmann.

Schaub, T. and Wang, K. (2001b) *Towards a semantic framework for preference handling in logic programming* (submitted).

van der Hoek, W. and Witteveen, C. (2000) Classical and general frameworks for recovery. In: Horn, W. (ed.), *Proceedings 14th European Conference on Artificial Intelligence* (*ECAI 2000*), pp. 33–37. IOS Press.

Wang, K., Zhou, L. and Lin, F. (2000) Alternating fixpoint theory for logic programs with priority. *Proceedings 1st International Conference on Computational Logic* (*CL 2000*)*: Lecture Notes in Computer Science 1861*, pp. 164–178. Springer-Verlag.

Zhang, Y. (2000) *Logic program based updates*. Draft available at http://www.cit.uws.edu.au/ yan/.

Zhang, Y. and Foo, N. (1997) Answer sets for prioritized logic programs. In: Maluszynski, J. (ed.), *Proceedings International Symposium on Logic Programming* (*ILPS'97*), pp. 69–84. The MIT Press.