RC 21472 (96900) 07 May 1999
Computer Science

IBM Research Report

Compiling Prioritized Default Rules into Ordinary Logic Programs

Benjamin N. Grosof

IBM Research Division
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598, USA
(914) 784-7783 direct -7455 fax -7100 main
Internet e-mail: grosof@us.ibm.com
(alt. grosof@cs.stanford.edu)
Web: http://www.research.ibm.com/people/g/grosof

Abstract:

Prioritized default rules offer a conveniently higher level of specification that facilitates revision and modularity. They handle conflicts, including arising during updating of rule sets, using partially-ordered prioritization information that is naturally available based on relative specificity, recency, and authority. Despite having received much study, however, they have had as yet little impact on on practical rule-based systems and software engineering generally, and had very few deployed serious applications.

We give a new approach to the semantics and implementation of prioritized default rules: to compile them into ordinary logic programs (LP's). (By "logic program" and "prioritized default rules", we mean in the sense of declarative knowledge representation (KR), including model-theoretic entailment and forward as well as backward inferencing. In particular, by "logic program", we do not simply mean Prolog.)

We use the compilation approach both to expressively generalize and to implement courteous LP's, a previous formalism featuring classical negation and prioritized conflict handling. We show that we preserve courteous LP's' attractive reasoning behaviors and polynomial-time computational cost. Our expressive generalization enables recursion, and also reasoning about the prioritization. Our implementation enables courteous LP's functionality to be added modularly to ordinary LP rule engines, via a pre-processor, with moderate, tractable computational overhead.

More generally, we show that the compilation approach is applicable to implementing, and to defining, numerous variants of prioritized default rule KR's, beyond the particular courteous LP variant given here.

This takes a long step towards actual deployment of prioritized default rules in commercially fielded technology and applications.

Related Papers and Material: can be found via the author's Web address and as IBM Research Reports at http://www.research.ibm.com .

Keywords: logic programming, default reasoning, priorities, non-monotonic reasoning, intelligent agents, rules, e-commerce, electronic commerce, compilation, knowledge compilation. More keywords: applications, modularity, tractability, negation-as-failure, Prolog, updating, belief revision, artificial intelligence, machine learning, deductive databases.

1 Introduction

The overall problem we address is: how to enable prioritized default rules to be used as a widely practical knowledge representation for specification and execution of rule-based software.

We are attracted by some virtues of prioritized default rules: they handle conflicts, including arising during updating of rule sets, using partially-ordered prioritization information that is naturally available based on relative specificity, recency, and authority.

Prioritized default rules are of long-standing interest in the knowledge representation (KR) community, and have received much study. Prioritized logic programs, a recent sub-family of prioritized default rules, extend ordinary logic programs (LP's) to feature classical negation and prioritized conflict handling. Compared to ordinary LP's, prioritized LP's enable a conveniently higher expressive level of specification that facilitates revision, updating, merging, and modularity.

However, prioritized default rules have as yet had little impact on practical rule-based systems and software engineering generally, and had very few deployed serious applications. One difficulty with prioritized default rules is getting the semantics right, including intuitively simple enough that non-experts in KR can feel comfortable specifying, and often repeatedly modifying, rule sets. Another difficulty is the complexity of implementing inferencing in a new KR. A third difficulty is facilitating a transition, which is best made incrementally, by builders and users of previous rule-based technology, to any new knowledge representation — including one for prioritized default rules.

We take a new overall approach to remedy these three difficulties, especially the third. We call this approach: **Prioritized Default Rules via Compilation to ordinary logic programs**, abbreviated **PDRC**. Compilation is a broad approach, in that multiple prioritized-default-rule KR's can be compiled into ordinary logic programs (OLP's). More precisely, the compilation is parametrized by a transform. For the sake of practicality and scaling, we require that the PDRC transform have worst-case polynomial-time complexity. Different semantics of the input KR can be defined and/or implemented via different transforms. Two given different transforms may correspond to the same semantics, or to different semantics.

By "knowledge representation", we mean a reasoning formalism that is specified in terms not only of syntax but also of semantics in the sense of what conclusions are entailed (a.k.a. sanctioned), i.e., a formalism that is declarative. By "logic program" and "prioritized default rules", we mean in the sense of declarative knowledge representation, including model-theoretic entailment (i.e., a set of premises entails a set of sanctioned conclusions) and forward as well as backward inferencing relative to such entailment. In particular, by "logic program", we do not simply mean Prolog, which is one particular kind of backward-inferencing LP engine.

In applying our approach, we begin by focusing on our favorite of the previous prioritized default rule KR's: courteous logic programs [9], a kind of prioritized logic programs. Courteous LP's have a number of attractive properties, detailed at more length in [8]. They have a unique consistent conclusion set, and are computationally tractable (under commonly-made assumptions) with relatively modest extra computational cost compared to OLP's. Their behavior captures many examples of prioritized default reasoning in a graceful, concise, and intuitive manner. They have a number of established well-behavior properties, including under merging.

As a target KR, OLP's are very attractive. They are computationally tractable (again, under commonly-made assumptions), unlike even the propositional case of classical logic, yet represent basic non-monotonicity via the negation-as-failure expressive feature. They are in widespread deployment and application, including by many programmers who know and care very little about KR generally. There are a number of highly efficient and sophisticated OLP rule systems / inferencing engines available, including for forward as well as backward inferencing. OLP's are also closely related to derived relations in SQL relational databases, and to several other varieties of rule-based systems.

Overview: see the Abstract.

2 Preliminaries

Background: We assume the reader is mainly familiar with: the standard concepts and results in the logic programming literature (e.g., as reviewed in [2]), including **ordinary** logic program (also known as a "normal" or "general" logic program; e.g., cf. pure Prolog), instantiation, unification, predicate / atom dependency graph and its cyclicity characteristics, the stratified semantics [1] and locally stratified semantics [14], the well-founded semantics, [16], the Clark predicate completion [5], and extended logic programs [6].

Well-Founded Semantics: There are multiple different semantics for OLP. In general, our compilation approach could be used with any of them. Our current favorite, however, is the well-founded semantics (**WFS**). We like both its behavior and its relative computational simplicity. It always has a unique model, i.e., set of conclusions. For ground OLP's (i.e., after instantiation of the OLP), inferencing (exhaustive or backward) in the WFS takes time $O(g^2)$ and has output size $O(g)$, where g is the size of the input OLP. For acyclic ground OLP's, inferencing takes time $O(g)$. Acyclic is a subclass of stratified, which is a subclass of locally stratified, which is a subclass of weakly (a.k.a. "dynamically" or "effectively") stratified; the stratified and locally stratified semantics are equivalent to special cases of the WFS. Acyclic means the atom dependency graph has no cycles; locally stratified means it has no cycles in which negation-as-failure appears. Formally, the WFS is defined in terms of a *model* $\langle T, F\rangle$, where T is the set of *true* atoms, and F is the set of $false$ atoms. p being *true* means that p is an entailed conclusion. p being $false$ means that p is not entailed, and also that $\sim p$ is satisfied. $\sim$ stands for negation-as-failure. When every atom is in $T \cup F$, then the model is said to be *total*, i.e., *2-valued*, otherwise it said to be *partial*, i.e., 3-valued. For weakly stratified OLP's, the WFS is 2-valued. More generally, sometimes p's truth value is assigned to a 3rd value: $undefined$.

3 Priorities + Classical Negation

We write **C1LP** to stand for the previous (i.e., "**version 1**") concept of courteous LP, defined in [9].

In the rest of this section, we define a new concept ("PCNLP" below): a less restricted version of C1LP syntax for which we give in this section a very partial semantics. Later on, we give for it a few different choices of complete semantics; more generally, it has a family of semantics, parametrized by the PDRC compilation transform. We closely follow the terminology and notation of [9], except where explicitly noted.

We are motivated to define the PCNLP syntax by the fact that two other prioritized LP KR's, in addition to C1LP, have a very similar syntax: [4] [17]. All three of these KR's were developed independently.

A **prioritized logic program with classical negation**, abbreviated as **PCNLP**, is defined syntactically as a set of *labelled rules*, whose language includes the *prioritization predicate Overrides*. Here, "labelled rule" and the "prioritization predicate" are as in C1LP.

PCNLP syntax thus differs from OLP as follows. Each atom in a rule may be classically negated. Each rule has an (optional) rule label. The rule labels are used as handles for specification of prioritization between rules, using a syntactically-reserved *prioritization predicate Overrides*. $Overrides(i, j)$ specifies that the label i has (strictly) higher priority than the label j.

A PCNLP rule, like a C1LP rule, has the form:

$\langle lab \rangle \quad L_0 \ \leftarrow \ L_1 \wedge \ \ldots \ \wedge L_m \wedge {\sim}L_{m+1} \wedge \ \ldots \ \wedge {\sim}L_n$

Here, $n \geq m \geq 0$, and each L_i is a classical literal. L_0 is called the *head* (i.e., consequent) of the rule; the rest is called the *body* (i.e., antecedent) of the rule. A *classical literal* is a formula of the form A or $\neg A$, where A is an atom. $\neg$ stands for the *classical negation* operator symbol, and is read in English as "not". $\sim$ stands for the *negation-as-failure* operator symbol, and is read in English as "fail". *lab* is the rule's label. The label is optional. If omitted, the label is said to be *empty*. The label is not required to be unique within the scope of the overall logic program; i.e., two rules may have the same label. The label is treated as a 0-ary function symbol. The label is preserved during instantiation; all the ground instances of the rule above have label *lab*. *Overrides* and the labels are treated as part of the language of the logic program, similarly to other predicate and function symbols appearing in the logic program.

A *literal* is a formula of the form L or ${\sim}L$, where L is a classical literal. An unnegated literal (i.e., an atom) is called *positive*. A ground rule with empty body is called a *fact*.

Syntactically, OLP is simply a special case of PCNLP. An OLP rule lacks a label and does not mention classical negation.

Intuitively, the semantic intent is that $\neg p$ means p is believed (or "known" or "proved" or "inferrable") to be false, while ${\sim}p$ means that p is not believed to be true; ${\sim}p$ thus can be satisfied if p is neither believed to be true nor believed to be false, i.e., if p's truth value is "unknown". Intuitively, $\neg p$ is stronger than ${\sim}p$; $\sim$ and $\neg$ are sometimes thus known as weak and strong negation, respectively.

Semantically, we interpret every rule with empty label as having the same catch-all label *emptyLabel*, which is treated as a new symbol (i.e., new with respect to the rest of the LP's language).

Semantically, we treat a PCNLP or OLP rule with variables as shorthand for the set of all its ground instances. This is as usual in the logic programming literature (including C1LP). We write $\mathcal{C}^{instd}$ to stand for the LP that results when each rule in $\mathcal{C}$ is replaced by the set of all its possible ground instantiations.

In the spirit of WFS, we define the **model**, (which we also call the **set of conclusions**) of a PCNLP most generally to be a truth value assignment that maps each each ground *classical literal* (rather than *atom* as in OLP WFS) to exactly one of $\langle true, false, undefined \rangle$, i.e., a model $\langle T, F \rangle$ for the ground classical literals. This generalization from atoms to classical literals is as usual in the logic programming literature when defining semantics for classical negation.

Relative to a PCNLP $\mathcal{C}$: the **rule locale** for a *predicate* p, written as $RuleLocale(p)$, is the (possibly empty) subset of rules within $\mathcal{C}$ in which p appears in the rule head (positively or negatively). Similarly, the rule locale for a ground *atom* p is the subset of rules within $\mathcal{C}^{instd}$ in which p appears in the head. (In [9] [8] this was called "definitional locale" and written as "$Defn(p)$".)

Within a (ground-)atom locale $RuleLocale(p)$, each rule has either head p or head $\neg p$. Intuitively, the rules having head p may *conflict* with the rules having head $\neg p$, with regard to whether p versus $\neg p$ should be concluded. The rules having head p "push" for p. The rules having head $\neg p$ "push" for $\neg p$". Prioritization information guides the resolution of this conflict.

Example 1 (Fred's Family Matters)

As an example of PCNLP, and as example of how we will generalize expressively beyond C1LP's syntax, we next give a modified version of [8]'s Example 10, a rule set about Fred's family and his mail's importance, that Fred specifies to his rule-based mail agent. Below, the "?" prefix indicates a logical variable.

$\langle Clo \rangle \quad Important(?msg) \;\leftarrow\; From(?msg, ?x) \wedge CloseFamily(?x, Fred)$
$\langle Dai \rangle \; \neg Important(?msg) \;\leftarrow\; From(?msg, AuntDaisy)$
$\langle Eme \rangle \quad Important(?msg) \;\leftarrow\; NotificationOf(?msg, ?es) \wedge PersonalEmergency(?es)$
$\langle CWA \rangle \; \neg Important(?msg) \;\leftarrow$

$$
\begin{array}{l}
Overrides(Dai, Clo) \;\leftarrow \\
Overrides(?x, CWA) \;\leftarrow\; (?x \neq CWA) \\
Overrides(?x, ?y) \;\leftarrow\; About(?x, MortalDanger) \wedge {\sim}About(?y, MortalDanger) \\
About(Eme, MortalDanger) \;\leftarrow \\
PersonalEmergency(?s) \;\leftarrow\; SevereIllnessOf(?s, ?x) \wedge CloseFamily(?x, Fred) \\
CloseFamily(?x, Fred) \;\leftarrow\; Ancestor(?x, Fred) \\
Ancestor(?x, ?z) \;\leftarrow\; Parent(?x, ?y) \wedge Ancestor(?y, ?z) \\
Parent(Mark, Fred) \;\leftarrow \\
Parent(Betty, Mark) \;\leftarrow \\
CloseFamily(AuntDaisy, Fred) \;\leftarrow \\
From(Item19, Betty) \;\leftarrow \\
From(Item20, AuntDaisy) \;\leftarrow \\
From(Item115, AuntDaisy) \;\leftarrow \\
NotificationOf(Item115, Sit79) \;\leftarrow \\
SevereIllnessOf(Sit79, AuntDaisy) \;\leftarrow
\end{array}
$$

4 Review: Courteous V1

In this section, we closely follow the terminology and notation of [9], except where explicitly noted.

Definition 2 (Courteous V1 Syntax)

Syntactically, a C1LP $\mathcal{C}$ is a PCNLP with three restrictions.
(1) $\mathcal{C}$ (i.e., its ground-*atom* dependency graph) is **acyclic**. [1]
(2) Prioritization is **"facts-only"**: Every rule in $\mathcal{C}$ mentioning *Overrides* has the form of a positive fact (about *Overrides*).
(3) The set of such **prioritization facts** in $\mathcal{C}$ specifies the prioritization relation to be a **strict partial order** on the labels. □

C1LP is called "courteous" because it is well-behaved and respects precedence (i.e., priority), in several regards. Every C1LP has a unique conclusion set that is consistent (inconsistent means both p and $\neg p$ are conclusions). Propositional C1LP inferencing is tractable: worst-case, quadratic-time.

In the definition of the semantics for C1LP in [9], the conclusion set is constructed incrementally (accumulatively) by iterating along a stratified sequence of (ground-)atom rule locales. The direction of the stratification, similar to locally-stratified semantics of OLP's, is from deeper (i.e., depended-upon in the sense of a head depending on a body) to shallower atoms; in addition, the *Overrides* atoms are treated as deepest. Every rule with empty label is interpreted as having the same catch-all label *empty_label*, which is treated as a new symbol (i.e., new with respect to the rest of the LP's language).

[1] Acyclicity (our terminology follows [2]) prevents recursion among ground atoms, hence is often also called **non-recursiveness** in the literature. This is a cause of some confusion, however, in that strictly speaking, acyclicity does *not* prevent recursion among *predicates*.

In each iteration (i.e., in each stratum), one atom rule locale p_i is "run" to add its conclusion, if any, to the conclusion set iterate. In this locale, p_i or $\neg p_i$ may be concluded; what happens is defined in terms of a **prioritized competition among candidate arguments**. A *candidate argument* c is generated iff the body of a rule r (in the locale) "fires": i.e., when r's body is satisfied in the previous ($(i-1)^{th}$) conclusion set iterate, i.e., is entailed by the conclusions from the deeper locales. Overall in the locale, there is thus a (possibly empty) team of candidates *for* p_i (i.e., having head p_i) and, likewise, a team of candidates for $\neg p_i$; these two teams are said to *oppose* each other. Each candidate has an associated label: c takes r's rule label. **A team wins iff: it has at least one unrefuted member, and every member of the opposing team is refuted. Refutation is based on prioritization**: one candidate having label j refutes another opposing candidate having label k iff j has higher priority than k, i.e., iff $Overrides(j, k)$ is satisfied in the previous conclusion set iterate. If the team for p_i wins, then p_i is added to the conclusion set; likewise for $\neg p_i$. It can happen that there are two non-empty teams, but neither team is refuted. In this case, then neither team wins; this corresponds to *mutual skeptical defeat*. It can also happen, of course, that neither team wins because both teams are empty, i.e., there are no candidates.

5 Compiling Generalized Courteous LP's

In this section, we define a new, generalized version of courteous LP's, using the compilation approach.

Central is a **conflict-resolution transform**, written `CR_Cour1`, which transforms any given PLPCN $\mathcal{C}$ into an OLP `CR_Cour1`($\mathcal{C}$). This transform in effect represents *within OLP* the C1LP semantical process of prioritized competition among teams of candidates.

In overview, the transform has four steps.

Step 1: Eliminate classical negation.
For each predicate P, each appearance of $\neg P$ is replaced by an appearance of the new predicate n_P.

Step 2: Analyze which pairs of rules are in opposition.
Opposition between two rules means that their heads represent unifiable complementary literals.

Step 3: For each predicate Q, create an associated output set of OLP rules. This is done by modifying the PCNLP rules whose heads mention Q, plus adding some more rules.

Step 4: Union the results of step 3 to form the overall output OLP.

Definition 3 (Eliminate Classical Negation)

The *ECN transform*, which eliminates (the appearance of) classical negation, takes an input PCNLP $\mathcal{C}$ and produces an output PCNLP `ECN`($\mathcal{C}$). This is done by a simple rewriting operation, essentially the same as that in [6]. Each appearance of $\neg P$ is replaced by n_P, where n_P is a newly introduced predicate symbol (with the same arity as P). We call n_P: P's *negation predicate*. n_P is only introduced if there is actually an appearance of $\neg P$ in the input PCNLP. Note that if the input PCNLP does not contain any appearances of classical negation, then the output PCNLP is simply the same as the input PCNLP.

We say that n_P is the *complement* or *opposer* of P, and vice versa. Given a predicate Q in `ECN`($\mathcal{C}$), we write $Q^{\neg}$ to stand for Q's complement.

It is useful to retain a trace, i.e., log record, of what rewriting was performed by any particular application of the ECN transform, i.e., to remember which predicates were original, which were newly introduced, and which pairs are complements. We accordingly define the ECN transform to additionally output $traceECN(\mathcal{C})$ which contains this information.

We further define the *inverse* ECN transform: $\texttt{ECN}^{-1}$, which maps $\texttt{ECN}(\mathcal{C})$ back into $\mathcal{C}$. This just rewrites n_P back to be $\neg P$. □

Definition 4 (Opposing Rules)

(a.) Relative to a PCNLP $\mathcal{C}$:
Let rule $rule_j$ have head $q(tj)$ and rule $rule_k$ have head $\neg q(tk)$, where q is a predicate, and tj and tk are term tuples of appropriate arity for q. We say that the rules $rule_j$ and $rule_k$ are *opposers* of each other when tj and tk are unifiable. We then also say that $\langle rule_j, rule_k \rangle$ is a *relevant opposition pair*, with associated maximum general unifier $\theta_{jk} = mgu(tj, tk)$. We also write θ_{jk} as $mgu(rule_j, rule_k)$. Note that θ_{jk} is required to be a non-empty substitution (i.e., a non-empty unifier / substitution). Opposition pairs are symmetric: we also say that $\langle rule_k, rule_j \rangle$ is a relevant opposition pair.

We write $RelOppPairs(rule_j)$ to stand for the set of all relevant opposition pairs in which rule $rule_j$ appears as the first member of the pair.

(b.) Relative to $\texttt{ECN}(\mathcal{C})$: We apply the same terminology and notation to post-ECN-transform rules where $\neg q$ has been rewritten as n_q.

Summary of (a.) & (b.): two rules are opposers, with associated mgu, if their heads represent unifiable complementary literals. □

Definition 5 (CR_Cour1 Per Locale)

Relative to a post-ECN-transform PCNLP $\texttt{ECN}(\mathcal{C})$: for each of its predicates q, we define the per-locale transform of its rules as follows.

We write this per-locale transform as $\texttt{CR\_Cour1}(\mathcal{C}, q)$ to indicate its association with the *original* (i.e., pre-ECN-transform) PCNLP $\mathcal{C}$.

Below, $q^{\neg}$ is to be read as: the complement of q. This is done *before* any subscripting is applied. E.g., if q is $Urgent$, then $q_u^{\neg}$ is n_Urgent_u. E.g., if q is $n\_Important$, then $q_{c4}^{\neg}$ is $Important_{c4}$.

If $RelOppPairs(q)$ is empty, then $\texttt{CR\_Cour1}(q)$ is: $RuleLocale(q)$ modified to remove the rule labels from each rule. I.e., in this case, the per-locale transform simply passes through the input's rule locale for predicate q, unchanged except to remove labels. In this case, we call the locale: **1-sided**. A special case of the 1-sided case is when $RuleLocale(q)$ or $RuleLocale(q^{\neg})$ are empty: then $RelOppPairs(q)$ is empty as well.

Otherwise, i.e., if $RelOppPairs(q)$ is non-empty, then $\texttt{CR\_Cour1}(q)$ is defined (more complexly) as follows. In this case, we call the locale: **2-sided**.

Initialize $\texttt{CR\_Cour1}(\mathcal{C}, q)$ to be empty.

(1.) For each $rule_j$ in $RuleLocale(q)$, add to $\texttt{CR\_Cour1}(\mathcal{C}, q)$ the rule:

$$q(tj) \;\leftarrow\; q_u(tj) \wedge \sim q_u^{\neg}(tj) \tag{1j}$$

Here, $rule_j$ has the form:

$$\langle lab_j \rangle \quad q(tj) \;\leftarrow\; Bj[yj] \tag{$rule_j$}$$

where $Bj[yj]$ stands for the body of $rule_j$; yj is the tuple of logical variables that appear in Bj; tj is the term tuple appearing (as argument tuple to q) in the head of $rule_j$. lab_j is $rule_j$'s rule label (which may be empty). q_u and $q_u^{\neg}$ are newly introduced predicates, each with the same arity as q. Intuitively, $q_u(t)$ stands for "q has an unrefuted candidate for instance t". Intuitively, $q_u^{\neg}(t)$ stands for "$q^{\neg}$ has an unrefuted candidate for instance t", or, alternatively, "q has an unrefuted opposer candidate for instance t".

(2.) For each $rule_j$ in $RuleLocale(q)$, add to $\texttt{CR\_Cour1}(\mathcal{C}, q)$ the rule:

$$q_{cj}(tj) \;\leftarrow\; Bj[yj] \tag{2j}$$

Here, $rule_j$ has the form:

$$\langle lab_j \rangle \quad q(tj) \;\leftarrow\; Bj[yj] \tag{$rule_j$}$$

where $Bj[yj]$ stands for the body of $rule_j$; yj is the tuple of logical variables that appear in Bj; tj is the term tuple appearing (as argument tuple to q) in the head of $rule_j$. lab_j is $rule_j$'s rule label (which may be empty). q_{cj} is a newly introduced predicate. Intuitively, $q_{cj}(t)$ stands for "q has a candidate for instance t, generated by rule $rule_j$"

(3.) For each $rule_j$ in $RuleLocale(q)$, add to $\mathtt{CR\_Cour1}(\mathcal{C}, q)$ the rule:

$$q_u(tj) \;\leftarrow\; q_{cj}(tj) \wedge \sim q_{rj}(tj) \tag{3j}$$

Here, q_{rj} is a newly introduced predicate. Intuitively, $q_{rj}(t)$ stands for "the candidate for q for instance t, generated by $rule_j$, is refuted". Intuitively, "refuted" means "refuted by some higher-priority conflicting rule's candidate".

(4.) For each $rule_j$ in $RuleLocale(q)$ and each $jkPair$ in $RelOppPairs(rule_j)$, add to $\mathtt{CR\_Cour1}(\mathcal{C}, q)$ the rule:

$$q_{rj}(tj \cdot \theta_{jk}) \;\leftarrow\; q^{\neg}_{ck}(tk \cdot \theta_{jk}) \wedge Overrides(lab_k, lab_j) \tag{4jk}$$

where $jkPair$ has the form:

$$\langle rule_j, rule_k \rangle \tag{$jkPair$}$$

Here, $rule_j$ is as in (3.), and similarly, $rule_k$ has the form:

$$\langle lab_k \rangle \quad q^{\neg}(tk) \;\leftarrow\; Bk[yk] \tag{$rule_k$}$$

$\cdot$ stands for the operation of applying a substitution, as usual with unifiers. θ_{jk} stands for $mgu(rule_j, rule_k)$. $q^{\neg}_{ck}$ bears the same relationship to $\langle q^{\neg}, rule_k \rangle$ as q_{cj} bears to $\langle q, rule_j \rangle$; it appears in aspect (3.) of $\mathtt{CR\_Cour1}(q^{\neg})$. *Overrides* stands, as usual, for the prioritization predicate. In the *Overrides* literal, if lab_j or lab_k is empty, then it is assigned to be *emptyLabel*. *emptyLabel* is a newly introduced 0-ary function (i.e., logical function symbol with 0 arity); intuitively, it stands for the empty rule label. (Recall the semantics of empty rule label in PCNLP, from section 3.) Note that *emptyLabel* is introduced at most once for a given $\mathcal{C}$, i.e., the same *emptyLabel* is shared by all the per-locale CR transforms. □

Definition 6 (CR_Cour1 Transform Overall)

Let $\mathcal{C}$ be a PCNLP. The overall output OLP $\mathtt{CR\_Cour1}(\mathcal{C})$ is defined as follows. Let $Preds(\mathtt{ECN}(\mathcal{C}))$ stand for the set of all predicates that appear in $\mathtt{ECN}(\mathcal{C})$.

$$\mathtt{CR\_Cour1}(\mathcal{C}) \;=\; \bigcup_{q \in Preds(\mathtt{ECN}(\mathcal{C}))} \mathtt{CR\_Cour1}(\mathcal{C}, q)$$

In other words, the output of the transform for the overall PCNLP is the result of first eliminating classical negation, via the ECN transform, then collecting (i.e., union'ing) all the per-locale transforms' outputs. □

One semantics (among many possible) for PCNLP as a knowledge representation is determined by compiling PCNLP to OLP via `CR_Cour1`, and adopting the well-founded semantics (WFS) for the resulting OLP. We call this formalism: version 1 of **unrestricted courteous-flavor** PCNLP, abbreviated **CU1LP**.

Let $\mathcal{C}$ be a given PCNLP. It has an *original* set of predicate and function symbols, i.e., ontology. $\texttt{CR\_Cour1}(\mathcal{C})$ typically has *extra* (i.e., newly introduced by the transform) predicate and function symbols. In general, these may include: the *original negation* predicates that represent classical negation of the original predicates (e.g., n_Urgent where $Urgent$ was an original predicate); the *adornment predicate* symbols that represent the intermediate stages of the process of prioritized argumentation (e.g., $Urgent_u$, $Urgent_{c4}$, $Urgent_{r4}$,n_Urgent_u); and the *adornment function* symbol $emptyLabel$.

We define the *OLP* conclusion set of $\mathcal{C}$ to be the WFS conclusion set of $\texttt{CR\_Cour1}(\mathcal{C})$. We also call this the *adorned* OLP conclusion set of $\mathcal{C}$, because it contains conclusions mentioning the adornment symbols. We define the *unadorned* OLP conclusion set to be the subset that does not mention any of the adornment symbols.

Corresponding to the OLP conclusion set is its *PCNLP version* which contains classical negation rather than negation predicates. We define the (unadorned) PCNLP-version conclusion set to be the result of applying the inverse ECN transform $\texttt{ECN}^{-1}$ to the unadorned OLP conclusion set, e.g., the negation predicate n_Urgent is rewritten instead as $\neg Urgent$.

When the context is clear, we will leave implicit the distinction between these different versions (adorned vs. unadorned, OLP versus PCNLP) of the conclusion set.

We summarize all this as follows.

Definition 7 (Compile PCNLP Via CR_Cour1)

Let $\mathcal{C}$ be a PCNLP. The CU1LP semantics for $\mathcal{C}$ is defined as the tuple
$\langle \mathcal{C}, \mathcal{O}, \langle T_O, F_O \rangle, \langle T_{PCN}, F_{PCN} \rangle \rangle$
Here, the post-transform (adorned) OLP $\mathcal{O}$ is $\texttt{CR\_Cour1}(\mathcal{C})$. $\langle T_O, F_O \rangle$ is the (WFS OLP) model for $\mathcal{O}$. $\langle T_{PCN}, F_{PCN} \rangle$ is the unadorned PCNLP version of $\langle T_O, F_O \rangle$. $\Box$

Next, we use the $\texttt{CR\_Cour1}$ transform and the compilation approach to define a generalized version of courteous LP's. To keep to the spirit of "courteous"-ness cf. C1LP, we wish to have some strong semantic guarantees about well-behavior that are similar to those in C1LP. Accordingly, we thus restrict CU1LP somewhat so as to ensure such well-behavior.

Definition 8 (Courteous LP's, Version 2)

Let $\mathcal{C}$ be a CU1LP. We say that $\mathcal{C}$ is a generalized, i.e., *version-2*, courteous LP, abbreviated as **C2LP**, when the following two restrictions are satisfied:
(1.) $\texttt{CR\_Cour1}(\mathcal{C})$ is locally stratified.
(2.) Prioritization is a strict partial order *"within"* every 2-sided rule locale.

By (2.) we mean the following. For every 2-sided predicate rule locale in Definition 5, the set of $Overrides$ tuples in T_O is a strict partial order when restricted to the set of rule labels appearing in $RuleLocale(p) \cup RuleLocale(p^\neg)$, where p is the locale predicate. $\Box$

Example 9 (Fred's C2LP)

Example 1 is a C2LP. Unlike a C1LP, it contains recursive (i.e., cyclic) dependencies, i.e., here about $Ancestor$. Also unlike a C1LP, it contains non-fact rules about $Overrides$; these result in reasoning about the prioritization. The $\texttt{CR\_Cour1}$ transform's output for the 2-sided $Important$ predicate locale is:
$Important(?msg) \;\leftarrow\; Important\_u(?msg) \wedge {\sim} n\_Important\_u(?msg)$
$Important\_c\_1(?msg) \;\leftarrow\; From(?msg, ?x) \wedge CloseFamily(?x, Fred)$
$Important\_u(?msg) \;\leftarrow\; Important\_c\_1(?msg) \wedge {\sim} Important\_r\_1(?msg)$

$Important\_r\_1(?msg) \leftarrow n\_Important\_c\_2(?msg) \wedge Overrides(Dai, Clo)$
$Important\_r\_1(?msg) \leftarrow n\_Important\_c\_4(?msg) \wedge Overrides(CWA, Clo)$
$Important(?msg) \leftarrow Important\_u(?msg) \wedge \sim n\_Important\_u(?msg)$
$Important\_c\_3(?msg) \leftarrow NotificationOf(?msg, ?es) \wedge PersonalEmergency(?es)$
$Important\_u(?msg) \leftarrow Important\_c\_3(?msg) \wedge \sim Important\_r\_3(?msg)$
$Important\_r\_3(?msg) \leftarrow n\_Important\_c\_2(?msg) \wedge Overrides(Dai, Eme)$
$Important\_r\_3(?msg) \leftarrow n\_Important\_c\_4(?msg) \wedge Overrides(CWA, Eme)$
$n\_Important(?msg) \leftarrow n\_Important\_u(?msg) \wedge \sim Important\_u(?msg)$
$n\_Important\_c\_2(?msg) \leftarrow From(?msg, AuntDaisy)$
$n\_Important\_u(?msg) \leftarrow n\_Important\_c\_2(?msg) \wedge \sim n\_Important\_r\_2(?msg)$
$n\_Important\_r\_2(?msg) \leftarrow Important\_c\_1(?msg) \wedge Overrides(Clo, Dai)$
$n\_Important\_r\_2(?msg) \leftarrow Important\_c\_3(?msg) \wedge Overrides(Eme, Dai)$
$n\_Important(?msg) \leftarrow n\_Important\_u(?msg) \wedge \sim Important\_u(?msg)$
$n\_Important\_c\_4(?msg) \leftarrow$
$n\_Important\_u(?msg) \leftarrow n\_Important\_c\_4(?msg) \wedge \sim n\_Important\_r\_4(?msg)$
$n\_Important\_r\_4(?msg) \leftarrow Important\_c\_1(?msg) \wedge Overrides(Clo, CWA)$
$n\_Important\_r\_4(?msg) \leftarrow Important\_c\_3(?msg) \wedge Overrides(Eme, CWA)$
`CR_Cour1` does not modify any of the other predicates' rule locales: they are all one-sided and do not mention classical negation.

The C2LP's entailed conclusions about $Important$ are: $Important(Item19)$, $\neg Important(Item20)$, and $Important(Item115)$. The C2LP's entailed conclusions about $Overrides$ include some resulting from reasoning through rules, e.g.: $Overrides(Eme, Dai)$. □

6 Well-Behavior

In this section, we show well-behavior properties for C2LP. We begin by showing that C2LP provides an equivalent alternative semantics, and thus an alternative means of implementation as well, for C1LP.

Theorem 10 (C2LP Equivalent on C1LP)

Let $\mathcal{C}$ be syntactically a C1LP.
If $\mathcal{C}$ is interpreted as a CU1LP, then:
(1.) $\mathcal{C}$ is a C2LP; and
(2.) $\mathcal{C}$'s unadorned PCNLP conclusion set is the same as the C1LP semantics' conclusion set, i.e., T_{PCN} is the same as the C1LP answer set.

Proof : (Detailed sketch)

(1.) Let $\mathcal{O}$ stand for the post-transform OLP `CR_Cour1`$(\mathcal{C})$. Examining the dependencies introduced by the `CR_Cour1` transform, it follows straightforwardly that pre-transform acyclicity of the dependency graph (on the original atoms) implies $\mathcal{O}$ is acyclic (i.e., on the post-transform atoms). Acyclicity implies local stratifiability. Prioritization premises being facts-only implies that the OLP model of the prioritization is simply the set of pairs specified by the premise prioritization facts. Prioritization premises being a strict partial order thus implies that the model of the prioritization is a strict partial order, including when projected onto any subset of the labels.

(2.) Because the $\mathcal{O}$ is locally stratified, its WFS coincides with the locally stratified semantics. In the rest of the proof, we adopt that as the semantics for $\mathcal{O}$.

Further examining the dependencies introduced by the `CR_Cour1` transform, it follows straightforwardly that we can choose $\mathcal{O}$'s (local-)stratification to be a sequence of strata from which there is a sequence-preserving isomorphism to the sequence of strata in the C1LP semantic construction for $\mathcal{C}$. Let p_i be the i^{th} (original) atom in the C1LP semantic construction. Let p_i have the form $q(a)$, where q is a predicate and a is a ground-term tuple. We choose the i^{th} $\mathcal{O}$-stratum to consist of the rule locales for the following set of atoms:

$$\{q(a), q_u(a), q_{rj}(a), q_{cj}(a),$$
$$q^{\neg}(a), q_u^{\neg}(a), q_{rj}^{\neg}(a), q_{cj}^{\neg}(a)\}$$

where j ranges over $RuleLocale(q)$ as in Definition 5. Note that some of these rule locales may be empty.

Each $\mathcal{O}$-stratum is acyclic. Thus in the locally-stratified semantics' iterated fixed point construction [14], each stratum's contribution to the conclusions is simply equivalent to (the ground literals derivable from) its Clark predicate completion [5]. (This equivalence property was shown for stratified semantics in [1], and straightforwardly generalizes to the locally-stratified case.)

The Clark predicate completion for each of the above-listed atoms straightforwardly implies that they equivalently represent the prioritized argumentation process for the $q(a)$ locale in the C1LP semantics (recall our earlier review of that in section 4). In particular, the unadorned conclusions about $q(a)$ and $q^{\neg}(a)$ in the $\mathcal{O}$-stratum are equivalent to those in the corresponding C1LP stratum.

Accumulating the conclusions while iterating in the stratification sequence thus (inductively) implies (2.). □

Theorem 11 (C2LP Behaves Courteously)

Let $\mathcal{C}$ be a C2LP. Then $\mathcal{C}$ *behaves courteously*, i.e., has the following four properties just as C1LP does.

(1.) Its conclusion set is unique and **2-valued**, i.e., its WFS model is total.

(2.) Its conclusion set is **consistent**, i.e., for every ground atom $q(a)$, $q(a)$ and $\neg q(a)$ are never both assigned to *true*.

(3.) Each unadorned conclusion can equivalently be described as resulting from the **same per-locale prioritized argumentation process as in the C1LP semantics** (recall review of that in section 4).

In more detail, this process is as follows. Consider the post-instantiation version of (PCNLP) $\mathcal{C}$. Let $q(a)$ be an original ground atom. If the body of $rule_j$ in with head $q(a)$ is satisfied, then we say $rule_j$ generates a candidate cj for $q(a)$, with associated label lab_j taken from $rule_j$. Likewise, if the body of $rule_k$ with head $q^{\neg}(a)$ is satisfied, we say $rule_k$ generates a candidate ck for $q^{\neg}(a)$, with associated label lab_k taken from $rule_k$. Let $Cands(q(a))$ consist of all the candidates for $q(a)$, and $Cands(q^{\neg}(a))$ consist of all the candidates for $q^{\neg}(a)$. Let cj be in $Cands(q(a))$, and ck be in $Cands(q^{\neg}(a))$. If $Overrides(lab_k, lab_j)$ is satisfied, then we say that cj is refuted. Likewise, if $Overrides(lab_j, lab_k)$ is satisfied, then we say that ck is refuted. Let $UnrefutedCands(q(a))$ consist of all the unrefuted candidates for $q(a)$, and $UnrefutedCands(q^{\neg}(a))$ consist of all the unrefuted candidates for $q^{\neg}(a)$. Then $q(a)$ is a conclusion iff $UnrefutedCands(q(a))$ is non-empty and $UnrefutedCands(q^{\neg}(a))$ is empty; and, likewise, $q^{\neg}(a)$ is a conclusion iff $UnrefutedCands(q^{\neg}(a))$ is non-empty and $UnrefutedCands(q(a))$ is empty.

(4.) The prioritized argumentation process in (3.) can alternatively be characterized as: a candidate is unrefuted exactly when it is maximal-priority; and a candidate wins exactly when it is maximal-priority and all other maximal-priority candidates agree with it.

In more detail, this process characterization is as follows. Let candidates be as in (3.). A candidate in $Cands(q(a)) \cup Cands(q^{\neg}(a))$
is **unrefuted iff**: it is **maximal-priority** in $Cands(q(a)) \cup Cands(q^{\neg}(a))$. Here, we say that an element is maximal-priority with respect to a set, when there is no other member of that set with

greater prioritization (in the sense of *Overrides* being satisfied). $q(a)$ (respectively, $q^{\neg}(a)$) is a conclusion iff the set of maximal-priority candidates (in $\mathcal{C}$'s atom rule locale, i.e., with head $q(a)$ or $q^{\neg}(a)$) is non-empty and all of them are for $q(a)$ (respectively, $q^{\neg}(a)$).

Proof : (Detailed sketch)

Let $\mathcal{O}$ stand for the post-transform OLP `CR_Cour1`$(\mathcal{C})$.

(1.) The WFS implies a unique model. This is 2-valued because $\mathcal{O}$ is locally stratifiable.

(2.) Consider the post-instantiation version of $\mathcal{O}$. Its local stratifiability implies the Clark predicate completion [5] (abbreviated *2PC*, where "2" stands for "2-valued")) for each of its atoms. It suffices to show that for any ground atom $q(a)$, that either $q(a)$ or $q^{\neg}(a)$ is assigned to $false$ in the model of $\mathcal{O}$. For each of these two atoms, the only rules in its locale arise from (1.) in Definition 5. Consider $q(a)$ in particular. It may have no rules in its locale, in which case its 2PC implies that $q(a)$ is $false$ in the model. Similarly, $q^{\neg}(a)$ may have no rules in its locale, in which case its 2PC implies that $q^{\neg}(a)$ is $false$ in the model. It thus suffices to show consistency for the case when both $q(a)$ and $q^{\neg}(a)$ have non-empty rule locales. If $q(a)$ does have one or more rules in its locale, then each of these rules is (a copy of):

$$q(a) \quad \leftarrow \quad q_u(a) \wedge {\sim} q_u^{\neg}(a)$$

Assume that $q(a)$ is $true$ in the model. The 2PC for $q(a)$ then implies that $q_u(a)$ is $true$ in the model and $q_u^{\neg}(a)$ is $false$ in the model. If $q^{\neg}(a)$ does have one or more rules in its locale, then each of these rules is (a copy of):

$$q^{\neg}(a) \quad \leftarrow \quad q_u^{\neg}(a) \wedge {\sim} q_u(a)$$

Since $q_u(a)$ is $true$ in the model and $q_u^{\neg}(a)$ is $false$ in the model, the 2PC for $q^{\neg}(a)$ implies that $q^{\neg}(a)$ must be $false$ in the model. Alternatively, instead of assuming that $q(a)$ is $true$ in the model, let us assume that $q^{\neg}(a)$ is $true$ in the model. The 2PC for $q^{\neg}(a)$ then implies that $q_u^{\neg}(a)$ is $true$ in the model and $q_u(a)$ is $false$ in the model. The 2PC for $q(a)$ then implies that $q(a)$ is $false$ in the model. In summary, the 2PC for $q(a)$ and $q^{\neg}(a)$ thus implies that $q^{\neg}(a)$ is not a conclusion if $q(a)$ is a conclusion, and vice versa.

(3.) This property follows from an argument that is very similar to the proof of part (2.) in Theorem 10. The main difference from the proof of part (2.) in Theorem 10 is that for general C2LP, unlike in C1LP, $\mathcal{O}$ may not be acyclic, and thus each $\mathcal{O}$-stratum may contain recursion.

$\mathcal{O}$'s local stratifiability implies the Clark predicate completion (2PC) for $\mathcal{O}$. (That stratifiability implies 2PC was shown in [1]. This straightforwardly generalizes to the locally-stratified case. Essentially, the Clark predicate completion follows from the fixed-point property of the model.)

The 2PC for $\mathcal{O}$ implies the 2PC for each of $\mathcal{O}$'s ground atoms. As in the proof of part (2.) in Theorem 10, consider the following set of atoms:

$$\{q(a), q_u(a), q_{rj}(a), q_{cj}(a),$$
$$q^{\neg}(a), q_u^{\neg}(a), q_{rj}^{\neg}(a), q_{cj}^{\neg}(a)\}$$

where j ranges over $RuleLocale(q)$ as in Definition 5. (Note that some of these rule locales may be empty.) The 2PC for each of the above-listed atoms straightforwardly implies that they equivalently represent the prioritized argumentation process for the $q(a)$ locale in the C1LP semantics (recall our earlier review of that in section 4).

(4.) The strict partial order property for prioritization (restriction (2.) in Definition 8) implies that unrefutedness corresponds to maximality in the prioritization ordering within the locale. □

7 Algorithms and Computational Complexity

Next, we analyze the computational complexity of CU1LP. We show it is worst-case polynomial-time for the propositional case and for the Datalog case with a bounded number of variables per rule. As part of this analysis, we give algorithms for CU1LP, including transforming and inferencing.

Let $\mathcal{C}$ stand for the input PCNLP. Let n stand for its size. Let $\mathcal{O}$ stand for the OLP `CR_Cour1`$(\mathcal{C})$.

Algorithm 12 (Transform `CR_Cour1`)

To perform the `CR_Cour1` transform:

1. Perform the ECN transform. To do this, linearly scan $\mathcal{C}$, rewriting each occurrence of classical negation, and building a table of the newly-introduced original negation predicates, with ancillary information constituting the $traceECN$.

2. Organize the post-ECN rules into predicate rule locales. To do this, linearly scan the rules, looking at their rule heads.

3. Build $RelOppPairs$. To do this, iterate through the original predicates. For each predicate, outer-loop through its rule locale, and inner-loop through its complementary predicate's rule locale, attempting to unify the inner-loop head atom with the outer-loop head atom. If successful in this attempt, store the relevant opposition pair with its unifier in $RelOppPairs$.

4. Initialize $\mathcal{O}$ to be empty. Then for each predicate:
perform the per-locale transform `CR_Cour1`$(\mathcal{C}, q)$, and append the result to $\mathcal{O}$. To do this, outer-loop through the rules in the locale; for step (4.) in Definition 5, also inner-loop through the rules in $RelOppPairs(outer - loop - rule)$. While doing all this, keep a record $traceAdorn$ of the newly-introduced predicates and functions, including which are adorning. □

Theorem 13 (Complexity of `CR_Cour1`)

`CR_Cour1`'s overall computational complexity is: $O(n^2)$ time and $O(n^2)$ output size.
Proof : (Sketch)
Step (1.) takes $O(n)$ time.
Step (2.) takes $O(n)$ time.
Step (3.) takes $O(n^2)$ time and its output has size $O(n^2)$.
Step (4.) takes $O(n^2)$ time.
□

Terminology: We say an LP, e.g., an OLP or a CU1LP, obeys the **VBD** restriction when that LP has an upper bound v on the number of variables appearing in any single rule, and is either Datalog (i.e., all function symbols are 0-ary) or ground. To indicate that the bound on the number of variables is v, we also say that the LP is VBD(v). Note that if the LP is ground, even if it is not Datalog, then there is an upper bound $v = 0$ on the number of variables per rule. Any ground LP thus VBD(0).

Output size complexity of instantiation (review):
In general, the result of (ground-)instantiating a given finite OLP containing variables may be infinite in size, e.g., when the Herbrand universe is infinite.
However, suppose the OLP is restricted to be VBD(v). Then the instantiated OLP's size is $O(h^{v+1})$, where h is the OLP's size. The Datalog restriction implies that the Herbrand universe is $O(h)$. The bound v on the number of variables per rule then implies that each rule has $O(h^v)$ instantiations.
□

Theorem 14 (Complexity of $\mathcal{O}^{instd}$'s size)

Suppose $\mathcal{C}$ is restricted to be VBD(v). Then $\mathcal{O}$ is also VBD(v), and $\mathcal{O}^{instd}$ has size $O(n^{(v+2)})$. By comparison, under the same VBD(v) restriction $\mathcal{C}^{instd}$ has size $O(n^{(v+1)})$.
Proof : (Detailed sketch)

Examining the particular form of the rules produced by `CR_Cour1`, we see that they preserve the restrictions: the post-transform OLP $\mathcal{O}$ is Datalog or ground, respectively, if $\mathcal{C}$ is Datalog or ground; and $\mathcal{O}$ has no more than v variables per rule. Moreover (also by such examination), the Herbrand universe — except for rule labels — of $\mathcal{O}$ is the same as that of $\mathcal{C}$.

The differences between $\mathcal{C}$'s and $\mathcal{O}$'s Herbrand universes are as follows. There are no more than $O(n)$ rule labels in $\mathcal{C}$; we will for clarity call these the "original" rule labels. These are part of $\mathcal{C}$'s Herbrand universe. $\mathcal{O}$'s rules have no labels. However, a subset (none, some, or all) of the original rule labels appear in $\mathcal{O}$'s rules as arguments to the prioritization predicate *Overrides*, i.e., in the rules generated from step (4.) of 5. Furthermore, *emptyLabel* may appear in $\mathcal{O}$'s such rules, while it need not appear in $\mathcal{C}$.

Therefore, $\mathcal{O}$'s Herbrand universe must (like $\mathcal{C}$'s) have size $O(n)$.

Recall (by Theorem 13) that $\mathcal{O}$ has size $O(n^2)$. Thus (by the above review of output size complexity of instantiation) $\mathcal{O}^{instd}$ has size $O(n^{(v+2)})$. □

By "inferencing" in the rest of this section, we mean either exhaustive or backward. By "exhaustive", we mean inferencing forward to compute the entire model, i.e., all its ground-literal conclusions. By "backward", we mean query-answering.

Algorithm 15 (CU1LP Inferencing)

To perform CU1LP inferencing (exhaustive or backward):

1. Perform the `CR_Cour1` transform on the input $\mathcal{C}$.
2. If inferencing backward, perform the ECN transform on the query.
3. Perform inferencing in $\mathcal{O}$ (with the post-ECN query if doing backward inferencing).
4. Perform the transformation of the results of inferencing back to the unadorned PCN version.

To do this, while linearly scanning the results, filter out conclusions mentioning adornment symbols and rewrite original negation predicates back to their classically-negated forms. This makes use of the *traceECN* and *traceAdorn* info generated by the `CR_Cour1` transform step. □

Theorem 16 (CU1LP Inference Complexity)

CU1LP inferencing (exhaustive or backward) has the following overall worst-case computational complexity bounds:

1. $O(n^{2\cdot(v+2)})$ time,
   $O(n^{(v+2)})$ output size for $\mathcal{C}$'s OLP conclusions, and
   $O(n^{v+1})$ output size for $\mathcal{C}$'s PCNLP conclusions,
   when $\mathcal{C}$ is restricted to be VBD(v).

2. $O(n^{(v+2)})$ time,
   when $\mathcal{C}$ is restricted to be acyclic and VBD(v).

3. $O(n^4)$ time, and $O(n^2)$ output size,
   when $\mathcal{C}$ is restricted to be ground, i.e., to be VBD(0).

4. $O(g^2)$ time, and $O(g)$ output size,
   when the size g of $\mathcal{O}^{instd}$ is finite.
   (Recall that g may be infinite, in general.)

In summary:

- When $\mathcal{C}$ is restricted to be VBD(v) ((1.), (2.) or (3.)),
CU1LP inferencing has the same worst-case time and space complexity as: OLP inferencing where the bound v on the number of variables per rule has been increased to $v+1$.

- More generally, CLP inferencing has time and space complexity that is *worst-case quadratically larger* than OLP inferencing. □

Proof : (Detailed Sketch)
Consider the steps of Algorithm 15.

Step (1.) takes time $O(n^2)$ and has output size $O(n^2)$, by Theorem 13.

Step (2.) takes time linear in the size of the query: thus $O(n)$ time, assuming the query is no longer than $\mathcal{C}$.

Step (3.) takes $O(g^2)$ time and has output size $O(g)$ — recall the complexity of OLP inferencing under the WFS.

Step (4.) takes $O(g)$ time, and has output size $O(g)$.

Suppose $\mathcal{C}$ is restricted to be VBD(v). Then $g = O(n^{v+2})$, by Theorem 14. Step (3.) thus takes $O(n^{2\cdot(v+2)})$ time and has output size $O(n^{(v+2)})$. And Step (4.) thus takes $O(n^{(v+2)})$ time and has output size $O(n^{(v+1)})$. The size of the unadorned PCN version is smaller than the adorned version because the adornments are filtered out, reducing the size of the Herbrand base from $O(n^{(v+2)})$ to $O(n^{(v+1)})$.
□

More Discussion of Overhead Compared to OLP Inferencing:
CU1LP inferencing has the same worst-case time and space complexity as: OLP inferencing where the bound v on the number of variables per rule has been increased to $v+1$, when $\mathcal{C}$ is restricted to be VBD(v).

By comparison, in WFS, under the same restrictions, OLP inferencing complexity is: $O(n^{(v+1)})$ time for acyclic case, $O(n^{2\cdot(v+1)})$ time more generally, and $O(n^{(v+1)})$ output size. In terms of worst-case time complexity, therefore, one is not paying a huge overhead for the expressive convenience of prioritized defaults with classical negation: only a polynomial degree or two beyond OLP inferencing, where OLP inferencing already itself costs multiple polynomial degrees when logical variables appear.

8 Implementation

We have built a running implementation [10] of general-case CU1LP. This will be demonstrated at the AAAI-99 conference during July 18–22, 1999. We built the `CR_Cour1` transformer ourselves; it is implemented in pure Java[2]. In addition, we use two previously existing WFS OLP inferencing engines built by others and implemented in C. One is exhaustive forward-direction: Smodels (version 1), by Ilkka Niemela and Patrik Simons, http://saturn.hut.fi/html/staff/ilkka.html. The other is backward-direction: XSB, by David Warren *et al*, http://www.cs.sunysb.edu/~sbprolog. We will be making our implementation publicly available via the Web in spring of 1999.

Note: The current implementation does not translate the results of OLP inferencing back to the unadorned PCNLP version. (I.e., it does not perform step (4.) of Algorithm 15.)

[2]trademark of Sun Microsystems

9 Variant Transforms

Our compilation approach enables one to use a variety of transforms from PCNLP to OLP, not just `CR_Cour1`. Changing the transform may result in different conclusions, i.e., a different semantics for PCNLP. In this section, we give a couple of example transforms that are simple modifications of `CR_Cour1`. Each modifies the semantics/behavior given to PCNLP by PDRC.

Grosof [8] gives an **alternative semantics for refutation**, called "**top-dog**":
q wins iff there is a candidate for q that
refutes all opposing candidates.
This behaves differently, e.g., when merging rule sets. Top-dog can be represented in our compilation approach by modifying the `CR_Cour1` transform as follows. In Definition 5: omit step (1.); and modify step (3.) by changing the head predicate of rule (3j) to be q instead of q_u.

When specifying or production-testing a rule set, it is often useful to be alerted to problems in the specification. To add an **alarm for the presence of active conflict that is not resolved by priority**, `CR_Cour1` can be modified as follows so as to generate additional adornments that represent mutual skeptical defeat. In Definition 5, add the extra step:
"(5.) For each $rule_j$ in $RuleLocale(q)$, add to $\mathtt{CR\_Cour1}(\mathcal{C}, q)$ the rule:

$$q_s(tj) \;\; \leftarrow \;\; q_u(tj) \wedge q_u^{\neg}(tj) \tag{5j}$$

Here, q_s is a newly introduced predicate. Intuitively, $q_s(t)$ stands for 'there is mutual skeptical defeat about $q(t)$', or 'there is stalemated, unresolved conflict about $q(t)$'."

Elsewhere, we give a further expressive generalization of PCNLP and CU1LP which permits the scope of conflict to be specified via **mutual-exclusion constraints**, a kind of integrity constraints. For that form of prioritized/courteous LP's, we apply our PDRC approach by developing a transform suitable for that KR formalism. A brief overview of this formalism, plus a long electronic-commerce example including this transform's output, is in [10].

10 Discussion

There are a number of previous approaches to prioritized logic programs with classical negation, pertinent to courteous LP's; see [8] for a review.

Compilation is of course a well-known idea in general programming languages and general software engineering. The idea of compiling more expressive KR's into less expressive KR's has received considerable attention in the last 10 years or so. However, to our knowledge, compilation has not been applied previously to compile prioritized default rules, e.g., prioritized LP's with classical negation, into ordinary LP's.

Gelfond & Son [7] give an approach to representing prioritized LP's in extended LP's, rather than OLP's. However, that approach is also quite different from ours in that it represents the specification of prioritized reasoning behavior at the meta-level, somewhat similar to a **Prolog meta-interpreter**. They also do not give as substantive characterizations of well-behavior as we do here and as the previous work on courteous LP's did, for particular prioritized reasoning schemes; rather, the emphasis is on enabling a mechanism for specification of a variety of such prioritized reasoning schemes.

Baral & Gelfond [2] give examples and discussion of using OLP's to represent prioritized default reasoning, but do not give a general method to go from prioritized defaults to OLP's.

11 Current Work

Current work takes several directions.

One direction is applications in electronic commerce. These include using courteous/prioritized LP's to represent: contractual agreements and product/service descriptions [15], e.g., in business-to-business and supply chain; business policies for security authorization [12]; and storefront personalization [10], e.g., in discounting, promotions, and advertising.

A second direction is an XML version of courteous/prioritized LP's used for interchange/translation [11] between heterogeneous rule-based intelligent agents / knowledge-based systems.

A third direction is generalizing further expressively.

A fourth direction is incremental compilation.

A fifth direction is relationships to Prioritized Default Logic [3], Defeasible Logic [13] and other prioritized default formalisms, including variants of LP's.

References

[1] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, Los Altos, CA, 1987.

[2] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994. Includes extensive review of literature.

[3] Gerhard Brewka. Reasoning about priorities in default logic. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 940–945, Menlo Park, CA / Cambridge, MA, 1994. AAAI Press / MIT Press.

[4] Gerhard Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.

[5] K. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[6] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991. An earlier version appears in *Proceedings of the Seventh International Conference on Logic Programming* (D. Warren and P. Szeredi, eds.), pp. 579–597, 1990.

[7] Michael Gelfond and Tran Cao Son. Reasoning with prioritized defaults. In Jurgen Dix, Luis Moniz Pereira, and Teodor Przymusinski, editors, *Logic Programming and Knowledge Representation (LPKR '97) (Proceedings of the ILPS '97 Postconference Workshop)*. http://www.uni-koblenz.de/~dix/LPKR97, 1997. Held Port Jefferson, NY, USA, Oct. 16, 1997. http://www.ida.liu.se/~ilps97.

[8] Benjamin N. Grosof. Courteous logic programs: Prioritized conflict handling for rules. Technical report, IBM T.J. Watson Research Center, http://www.research.ibm.com , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, Dec. 1997. IBM Research Report RC 20836. This is an extended version of [9].

[9] Benjamin N. Grosof. Prioritized conflict handling for logic programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, pages 197–211, Cambridge, MA, USA, 1997. MIT Press. Held Port Jefferson, NY, USA, Oct. 12-17, 1997. http://www.ida.liu.se/~ilps97. Extended version available as IBM Research Report RC 20836 at http://www.research.ibm.com .

[10] Benjamin N. Grosof. DIPLOMAT: Compiling Prioritized Default Rules Into Ordinary Logic Programs, for E-Commerce Applications (extended abstract of Intelligent Systems Demonstration). In *Proceedings of AAAI-99*, San Francisco, CA, USA, 1999. Morgan Kaufmann. Extended version available in May 1999 as an IBM Research Report RC21473, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[11] Benjamin N. Grosof and Yannis Labrou. An Approach to using XML and a Rule-based Content Language with an Agent Communication Language. In *Proceedings of the IJCAI-99 Workshop on Agent Communication Languages*, 1999. Held in conjunction with the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99) http://www.ijcai.org . Extended version available in May 1999 as IBM Research Report, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[12] Ninghui Li, Joan Feigenbaum, and Benjamin N. Grosof. A logic-based knowledge representation for authorization with delegation (extended abstract). In *Proceedings of 12th international IEEE Computer Security Foundations Workshop*, 1999. Extended version available in May 1999 as an IBM Research Report, http://www.research.ibm.com , navigate to Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[13] Donald Nute. Defeasible logic. In *Handbook of Logic in Artificial Intelligence and Logic Programming Vol. 3*, pages 353–395. Oxford University Press, 1994.

[14] Teodor Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann, San Francisco, CA., 1988.

[15] Daniel M. Reeves, Benjamin N. Grosof, Michael Wellman, and Hoi Y. Chan. Toward a Declarative Language for Negotiating Executable Contracts. In *Proceedings of the AAAI-99 Workshop on Artificial Intelligence in Electronic Commerce (AIEC-99)*, Menlo Park, CA, USA; http://www.aaai.org , search for workshop Technical Reports;, 1999. American Association for Artificial Intelligence (AAAI Press). Also available in May 1999 as an IBM Research Report, http://www.research.ibm.com , search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA. Earlier version appeared at the IBM Institute for Advanced Commerce Workshop on Internet Negotation Technologies, http://www.ibm.com/iac/ .

[16] A. Van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, 1991.

[17] Yan Zhang and Norman Y. Foo. Answer sets for prioritized logic programs. In Jan Maluszynski, editor, *Logic Programming: Proceedings of the International Symposium (ILPS-97)*, pages 69–83, Cambridge, MA, USA, 1997. MIT Press. Held Port Jefferson, NY, USA, Oct. 12-17, 1997. http://www.ida.liu.se/~ilps97.