Version of July 20, 1999

A Courteous Compiler
From Generalized Courteous Logic Programs
To Ordinary Logic Programs
(Preliminary Report)

(Supplementary Update Follow-On to IBM Research Report RC 21472)

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
(or grosof@watson.ibm.com or grosof@cs.stanford.edu)
Web: http://www.research.ibm.com/people/g/grosof;
or http://www.research.ibm.com to find IBM Research Reports.

Abstract

We define a generalized form of courteous logic programs (GCLP), and how to transform (i.e., compile) GCLP into ordinary logic programs (OLP). This builds upon the transformational approach in [5]. We detail the syntax of GCLP, and briefly discuss the intended semantics of GCLP in an informal fashion. Its precise semantics are defined in terms of the semantics of the OLP outputted by the transform, in a manner similar to that in [5]. We briefly discuss the computational complexity of GCLP; it retains the attractive tractability of the previous less general form of courteous logic programs. In a future version of this paper, we will give formal well-behavior results about GCLP.

The GCLP form of courteous LP's, and of the courteous compiler that transforms a GCLP into an OLP, described here corresponds to the version currently implemented in the IBM CommonRules prototype system, an alpha release of which will be publicly available free on the Web in summer of 1999.

Contents

1 Introduction and Overview

In this paper, we define **generalized courteous logic programs (GCLP)**, as implemented in the IBM CommonRules prototype system. This prototype is a Java class library. An alpha version of this prototype is scheduled for public release on the Web, at IBM's AlphaWorks in July or August of 1999, with a free trial license: see http://alphaworks.ibm.com , or see the pointer to there from http://www.research.ibm.com/rules/ .

GCLP is further expressively generalized relative to the form of courteous logic programs (BCLP) previously defined in [5], which was itself expressively generalized from the basic form in [4] [3]. Relative to the basic form of courteous LP's in [4], the generalization in this paper has four aspects.
1) Mutual exclusion constraints (*mutex*'s), each between a pair of classical literals, are permitted. The mutual exclusion between p and $\neg p$ is a special case of such a mutex. Consistency with respect to such mutex's is then enforced.
2) Recursive dependence among the predicates is permitted without restriction.
3) The appearance of the prioritization predicate is unrestricted.
4) Reasoning about the prioritization ordering is permitted. I.e., inferring conclusions about the *overrides* predicate, derived from rules possibly via chaining, is permitted.
Semantic guarantees are stronger, however, under additional expressive restrictions.
Only aspects (2)–(4), but not aspect (1), were contained in [5].

As part of defining GCLP, we describe a "Conflict-Resolving" ("CR") transformer from generalized courteous logic programs (GCLP's) to ordinary logic programs (OLP's): for each input GCLP, the transformer outputs a corresponding OLP. This transformer is also known as a "**courteous compiler**". The CR transformation provides an indirect semantics for GCLP in terms of OLP. Relative to OLP, GCLP provides additional expressive features and functional capabilities, which include classical negation and mutual exclusion constraints, as well as prioritized conflict handling. The CR transformer provides a means to achieve this higher level of functionality, by composing the transformer (embodied as a software component) with (software) components that can manipulate OLP's, e.g., with OLP inference engines or OLP rule editors. Multiple such OLP inference engines and OLP rule editors previously exist commercially. The CR transformer is thus suitable for commercial application as a pre-processor for rule-based systems.

This paper is an update follow-on to [5].

2 Preliminary Definitions

Background: We assume the reader is familiar with: the previous published version of courteous logic programs [4] [3], extended logic programs [2], the concept of ordinary, non-extended logic programs, the semantics of stratified (ordinary, non-extended) logic programs with negation as failure (e.g., [7]), and the standard concepts in the logic programming literature (e.g., as reviewed in [1]), including predicate / atom dependency graph and its acyclicity / non-recursiveness; and instantiation. **Appendix A contains some review** of these concepts.

In this section, we introduce some preliminary definitions, notation, and terminology. This includes reviewing extended logic programs (ELP's) cf. [2].

Each *rule* r in an *extended* logic program $\mathcal{E}$ has the form:

$$L_0 \;\leftarrow\; L_1 \wedge \ldots \wedge L_m \wedge {\sim}L_{m+1} \wedge \ldots \wedge {\sim}L_n$$

where $n \geq m \geq 0$, and each L_i is a literal.

We will define courteous LP's' rule syntax to be similar but not identical to that of extended LP's.

Notation and Terminology: A *literal* (which we will also call 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 $\sim$ for the *negation-as-failure* operator symbol. In English, we read the former as "not" and the latter as "fail". We say that an unnegated literal (i.e., an atom) is *positive*. A ground rule with empty body is called a *fact*. Syntactically, an "**ordinary**" **logic program (OLP)** (also known as a "*general*" logic program) is one in which each literal L_j above is an atom, i.e., where no classical negation is permitted.

3 Definition: Generalized Courteous LP's

Next, we define an expressively-generalized form of courteous logic programs, concentrating on its syntax. We call these *Generalized Courteous Logic Programs (GCLP)* The theory of GCLP's semantics is the subject of other papers rather than of this paper.

Our point of departure is the previous published version of courteous LP's, which we will call here "Basic Courteous Logic Programs" (BCLP).

3.1 Overview

The generalization of GCLP (relative to BCLP) has three aspects.

- **Mutual exclusion constraints ("mutex"'s)** are permitted. Each such constraint specifies mutual exclusion **between a pair of (classical) literals**.

  Intuitively, the mutual exclusion specifies that at most one of the literals, rather than both, should be inferrable from the GCLP.

  Intuitively, the set of mutual exclusion constraints in a GCLP taken together specify the scope of conflict, and thereby the scope of conflict handling.

  Implicitly, A and $\neg A$ are treated as mutually exclusive, for each atom A. This is essentially similar to BCLP.

- Recursive dependence among the predicates is permitted without restriction. This relaxes the restriction in BCLP; there, such recursive dependence was prohibited.

- The appearance of the prioritization predicate (*Overrides*, which has a special role semantically) is unrestricted.

  This relaxes two restrictions in BCLP; there, *Overrides* was required to appear only in rules having the form of facts, and the prioritization relation specified by the set of rules about *Overrides* as required to be a strict partial order.

Note that BCLP is a special case of GCLP, syntactically.

Semantic guarantees will be stronger under additional expressive restrictions. However, that theory about the semantics is the subject of other papers rather than of this document.

Relative to ordinary logic programs (OLP), GCLP is expressively-generalized in the following ways:

- Each rule is permitted to have a (rule) label. These labels are used as handles for specification of prioritization between rules. (This is as in BCLP.)

- Classical negation is permitted to appear in any rule literal. If it appears, it must be inside the scope of negation-as-failure. (This is as in BCLP.)
- Mutual exclusion constraints (as described above) are permitted.

Note that OLP is a special case of GCLP as well as of BCLP, syntactically.

3.2 Syntax

Next, we define GCLP's syntax in detail.

Syntactically, a GCLP is defined as a class of extended logic programs in which, additionally, rules have (optional) labels and mutual exclusion constraints may appear. These labels are used as handles for specification of prioritization between rules.

Definition 1 (Labelled Rule)

A *labelled rule* has the form:

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

where *lab* is the rule's label (and, as before, $n \geq m \geq 0$, and each L_i is a literal). 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*. □

Terminology: henceforth, when the context is clear, "rule" will be used to mean "labelled rule", unless the distinction is made explicitly.

In the newly generalized version of CLP, the CLP also optionally includes a set of pairwise mutual exclusion (*mutex*) statements, along with the rules. These mutex's specify the scope of conflict.

Definition 2 (Mutual Exclusion Constraint, i.e., Mutex)

A *mutual exclusion constraint* (*mutex* for short) has one of the two syntactic forms below; it is either unconditional or conditional.

An *unconditional* mutex has the syntactic form:

$\perp \ \leftarrow \ L_1 \land L_2.$

where each L_i is a classical literal.

Intuitively, the mutex specifies that at most one of the literals, rather than both, should be inferrable from the GCLP. More precisely, the mutex specifies this for any instantiation of the logical variables appearing in the mutex.

Intuitively, the set of mutual exclusion constraints in a GCLP taken together specify the scope of conflict, and thereby the scope of conflict handling.

As we will discuss more later when defining the transform from GCLP to OLP: implicitly, A and $\neg A$ are treated as mutually exclusive, for each atom A. This is essentially similar to BCLP.

These mutex's are particularly useful for specifying that at most one of a set of alternatives is to be permitted. E.g., it is straightforward to specify via 3 mutex's that the lead time must be at most one of the values {2 days, 14 days, 30 days}. E.g., it is straightforward to specify via 3 mutex's that the discount percentage must be at most one of the values {0%, 5%, 10%}.

It is expressively more convenient sometimes to use a more general form of mutex: a *conditional* mutex, which has the syntactic form:

$\perp \ \leftarrow \ L_1 \land L_2 \ | \ G_1 \land \ldots \land G_g.$

Here, $g \geq 0$, each L_i is a classical literal, and each G_j is a literal (in which $\sim$, as well as $\neg$, may appear). If g=0, then the " $|$ " is omitted. The conjunctive formula $G_1 \wedge \ldots \wedge G_g$ is called the "*given*" part of the mutex. Intuitively, the given part of the mutex specifies conditions under which L_1 and L_2 oppose each other. More precisely, it specifies that L_1 and L_2 should not both be inferrable if the given is true (i.e., if the given is inferrable).

At this point in the evolution of the courteous LP formalism, the main motivation for permitting conditional (rather than simply unconditional) mutex's is to permit conditional mutex's that have the syntactic form:

$$\perp \;\leftarrow\; L_1 \wedge L_2 \;|\; (?Y \neq ?Z).$$

where $?Y$ and $?Z$ are logical variables that appear respectively in L_1 and L_2. E.g.,

$$\perp \;\leftarrow\; giveDiscount(?Cust, ?YPercent) \wedge giveDiscount(?Cust, ?ZPercent)$$
$$|\; (?YPercent \neq ?ZPercent).$$

This conditional mutex enables one to specify with a single mutex statement (rather than with three or more unconditional mutex's) that for a given customer $?Cust$ there must be at most one value concluded for the discount percentage.

Beyond this restricted form of conditional mutex's, we are treating the usefulness of conditional mutex's as an issue to investigate both theoretically and empirically. The full expressive generality of conditional mutex's as defined above may be unnecessary or even undesirable. In short, **the full expressive generality of conditional mutex's is experimental** in this sense.

The enforcement of classical negation can be viewed as set of *implicit* unconditional mutex's, one for each predicate Q, that each have the form

$$\perp \;\leftarrow\; Q(?X1, \ldots, ?Xm) \wedge \neg Q(?X1, \ldots, ?Xm).$$

where Q's arity is m. This is called a *classical* mutex. □

Definition 3 (Prioritization Predicate)

A special binary predicate *Overrides* is used to specify prioritization. $Overrides(i, j)$ specifies that the label i has (strictly) higher priority than the label j. □

Intuitively, prioritization is used as part of handling conflicts that arise when two rules (whose bodies are satisfied) have mutually exclusive heads; prioritization influences which rule's head will be inferred as a conclusion.

Definition 4 (Generalized Courteous LP: Syntax; PCNMLP)

A generalized courteous logic program $\mathcal{C}$ is defined as a collection of labelled rules and mutex's, i.e., as the union of a set of (labelled) rules and a set of mutex's:

$$\mathcal{C} \;=\; \mathcal{C}_{rule} \cup \mathcal{C}_{mutex}$$

Both $\mathcal{C}_{rule}$ and $\mathcal{C}_{mutex}$ may be empty.

We also call this GCLP syntax: **PCNMLP** syntax. "P" stands for "with Priorities", "CN" stands for "with Classical Negation", and "M" stands for "with Mutex's". □

Note that the prioritization predicate *Overrides* and the labels are treated as part of the language of the logic program, similarly to other predicate and function symbols appearing in $\mathcal{C}$.

Terminology and Notation: We call $\mathcal{C}_{rule}$ the *rule set* or *rules* of $\mathcal{C}$. We call $\mathcal{C}_{mutex}$ the *mutex set* or *mutex's* of $\mathcal{C}$. We write $\mathcal{C}_{Overrides}$ for the subset of $\mathcal{C}_{rule}$ in which *Overrides* appears, and call this the *prioritization rules subset* of $\mathcal{C}$.

3.3 Semantics Overview

In this paper, we formally do *not* define semantics directly for GCLP, but rather describe how to transform a GCLP into an OLP (in section 4). This continues the overall transformational approach described in [5].

In [3], by contrast, semantics for BCLP are defined directly in an abstract, model-theoretic fashion.

For now, our intention for usage of GCLP is summarized in the following overview of its *intended* semantics and behavior, *not yet proven as results*, which probably require some expressive restrictions to ensure that they hold. Elsewhere, we will give more detailed theory about GCLP, including theorems about its semantics and behavior.

3.3.1 Intended Semantics and Behavior; not yet proven as results, probably needs additional expressive restrictions

Semantically, the prioritized conflict handling in CLP is defined by a process of **prioritized argumentation among opposing candidates**. Opposition is specified by the set of mutex's. Each rule r whose body is satisfied, i.e., which "fires", generates a candidate c for r's (appropriately instantiated) head p. This candidate has an associated label, which is simply that rule r's label. In general, there may be multiple candidates for a given p, i.e., a *team* for p. Iff there is an opposing candidate d (i.e., a candidate for an opposing literal q) that has higher priority than candidate c, then c is *refuted*. Suppose there is an unrefuted candidate for p. If there are no unrefuted candidates for any opposer of p, then p *wins*, i.e., p is concluded. However, it may be that there is an unrefuted candidate for an opposer of p; in this case, the opposing unrefuted candidates *skeptically defeat* each other. The conflict cannot be resolved by the specified priority; neither p nor its opposer is concluded.

Another way to view this is as follows. An **opposition-locale** is a set of $h \geq 2$ ground classical literals that oppose each other, such that at most one of those literals is permitted (by the specified mutex's) to be concluded. In each opposition-locale, if the maximal-priority candidates are all for the same literal, then that literal wins.

Figure 1 illustrates the above semantic steps for an opposition-locale. It emphasizes three phases: 1) rules firing to produce candidates, 2) candidates refuting each other based on priority, and 3) skepticism being applied. A "rule for pi" means a rule having head pi. "Running" a rule means generating candidates from it. "Previous" means in the sense of well-founded induction in the well-founded semantics' semantic construction.

4 Transforming GCLP to OLP

In this section, we describe how to transform any given GCLP $\mathcal{C}$ into an OLP $\mathcal{O}$. We write this transform $\mathtt{CR}(\mathcal{C})$.

Relative to OLP, GCLP contains additional expressive features, e.g., mutex's, classical negation, rule labels. These together with a special role of the prioritization predicate *Overrides* are used to specify prioritized conflict handling. In this sense, GCLP has a higher level of expressiveness than OLP.

The transform thus specifies how this higher level of expressiveness can be supported in OLP, albeit indirectly.

4.1 Overview

In this subsection, we give an overview of the transform's 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; and a new (explicit) mutex between P and n_P is introduced.

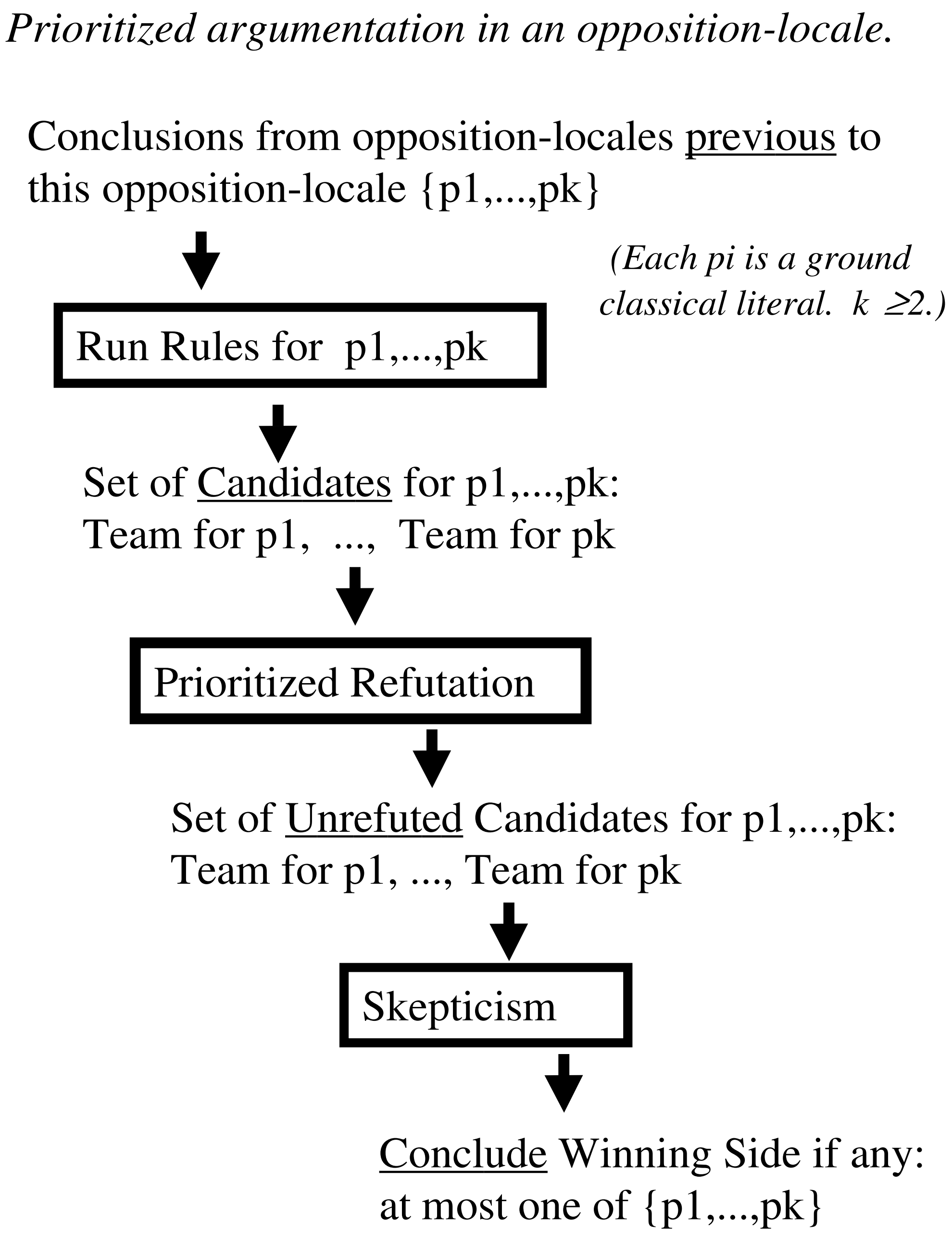

Figure 1: Semantic Steps of Inferencing Within an Opposition-Locale

Step 2: Analyze which pairs of rules are in opposition.
Opposition between two rules means that there is a mutex relating their rule heads.

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

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

4.2 Definitions

Definition 5 (Transform to Eliminate Classical Negation (ECN))

We define the *ECN transform*, which eliminates (the appearance of) classical negation, as follows. It takes an input GCLP $\mathcal{C}$ and produces an output GCLP which we write as $\mathcal{ECN}(\mathcal{C})$. This is done by a simple rewriting operation, similar to that in [2]. This rewriting has two aspects. The first aspect is that each appearance of $\neg P$ (in the mutex's as well as in the rules) 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 PCNMLP. Note that if the input PCNMLP does not contain any appearances of classical negation, then the output PCNMLP is simply the same as the input PCNMLP.

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

n_P is only introduced if there is actually an appearance of $\neg P$ in the input GCLP.

The second aspect of the rewriting (which is not present in [2]) is the following. If n_P is introduced, then a new (explicit) mutex between P and n_P is also introduced (i.e., added to the output GCLP); we call this a *classical mutex*. This mutex has the form:

$$\bot \ \leftarrow \ P(x) \wedge n\_P(x). \qquad \text{(classicalMutexP)}$$

where x is a tuple of logical variables, of arity appropriate for P.

Note that if the input GCLP does not contain any appearances of classical negation, then the output GCLP is simply the same as the input GCLP.

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

Intuitively, the classical mutex between P and n_P corresponds to there being an implicit mutex between P and $\neg P$ before the ECN transform.

Definition 6 (A Predicate's Rule Locale)

Relative to a GCLP $\mathcal{C}$: the *rule locale* for a predicate P, written $RuleLocale(P)$, is defined as the (possibly empty) subset of rules in $\mathcal{C}$ in which P appears in the rule head (positively or negatively). □

Roughly speaking, opposition between two rules means that there is a mutex relating their heads. The following makes this more precise.

Definition 7 (Opposition between Rules, Predicates, Literals)

Relative to a GCLP $\mathcal{C}$ in which classical negation does not appear:
Let rule $r1$ have head $P1(t1)$ and rule $r2$ have head $P2(t2)$, where $P1$ and $P2$ are predicates, and $t1$ and $t2$ are term tuples. The rules $r1$ and $r2$ are in *opposition*, i.e., are *opposers* of each other, if and only if (iff) there is some mutex m having the form

$$\bot \ \leftarrow \ P1(u1) \wedge P2(u2) \ | \ Ei[zi].$$

, where $u1$ and $u2$ are term tuples, such that $t1 = u1$ and $t2 = u2$ are *simultaneously* unifiable in the sense that there is a substitution θ that both unifies $t1$ with $u1$ and unifies $t2$ with $u2$.

(Note that the above concept of opposition can be straightforwardly generalized to the case where classical negation does appear: simply match literals' classical signs, and include consideration of classical mutex's.)

If such an m exists, we say that:
m is a *relating mutex* for the pair $\langle r1, r2\rangle$;
m is a *relevant mutex* for $r1$ and for $r2$;
$\langle r1, \theta, r2\rangle$ is a *relevant opposition triple*, with associated relating mutex m, where θ is the maximum general unifier that simultaneously both unifies $t1$ with $u1$ and $t2$ with $u2$ (as described above).
Also, if such an m exists, we say that:
the *predicates* $P1$ and $P2$ are in opposition, i.e., are opposers of each other;
m is a relating mutex for the pair $\langle P1, P2\rangle$; and
m is a relevant mutex for $P1$ and for $P2$.
Furthermore, if such an m exists, we say that:
the *literals* $P1(t1)$ and $P2(t2)$ are in opposition, i.e., are opposers of each other;
m is a relating mutex for the pair $\langle P1(t1), P2(t2)\rangle$; and
m is a relevant mutex for $P1(t1)$ and for $P2(t2)$.

Note that θ above is equivalent to the maximum general unifier of:

$$\langle t1, t2\rangle = \langle u1, u2\rangle$$

where $\langle t1, t2\rangle$ stands for the tuple formed by concatenating the tuples $t1$ and $t2$; and, likewise, $\langle u1, u2\rangle$ stands for the tuple formed by concatenating the tuples $u1$ and $u2$.

We also write θ as $mgu(r1, m, r2)$. Note that θ is required above to be non-empty (i.e., a non-empty unifier / substitution).

We write $RelOppTriples(rule_j)$ to stand for the set of all relevant opposition triples in which $rule_j$ appears as the first member of the triple.

We write $RelMuts(q)$ to stand for the set of all mutex's that are relevant to q.

We write $HasRelOppTriples(q)$ to stand for whether: there is a rule $rule_j$ in $RuleLocale(q)$ such that $RelOppTriples(rule_j)$ is non-empty. (This is a boolean; iff true, it indicates there are indeed such triples). $\Box$

Definition 8 (Conflict Resolution Transform Per Locale)

Relative to a GCLP $\mathcal{C}$ in which classical negation does not appear: for each predicate q, we define the per-locale transform $\mathtt{CR}(\mathcal{C}, q)$ as follows.

Below, we will leave $\mathcal{C}$ implicit notationally.

If $HasRelOppTriples(q)$ is false, then $\mathtt{CR}(q)$ is $RuleLocale(q)$. I.e., in this case, the per-locale transform simply passes through the input's rule locale for predicate q, unchanged. A special case is when $RuleLocale(q)$ is empty: then $HasRelOppTriples(q)$ is false, and $\mathtt{CR}(q)$ is empty.

Otherwise, i.e., if $HasRelOppTriples(q)$ is true, then $\mathtt{CR}(q)$ is defined (more complexly) as follows.

"Include" below means "include in the output of the transform".

Let $RuleLocale(q)$ stand for the set of all rules that are in the rule locale for predicate q. For each rule $rule_j$ in $RuleLocale(q)$, include the rule:

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

Here, q_u and q_s 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", and $q_s(t)$ stands for "q is skeptically defeated for instance t".

For each rule $rule_j$ in $RuleLocale(q)$, include the rule:

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

where $rule_j$ has the form:

$$q(tj) \;\leftarrow\; Bj[yj]. \tag{$rule_j$}$$

Here, $Bj[yj]$ 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 $rule_j$. q_{cj} is a newly introduced predicate. Intuitively, $q_{cj}(t)$ stands for "q has a candidate for instance t, generated by rule $rule_j$"

For each rule $rule_j$ in $RuleLocale(q)$, include 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".

Recall from Definition 7 that $RelOppTriples(rule_j)$ stands for the set of all relevant opposition triples in which $rule_j$ appears as the first member of the triple.

For each rule $rule_j$ in $RuleLocale(q)$ and each relevant opposition triple $jikTriple$ in $RelOppTriples(rule_j)$, include the rule:

$$\begin{aligned} q_{rj}(tj{\cdot}\theta_{jik}) \;\leftarrow\; & q_{cj}(tj{\cdot}\theta_{jik}) \wedge p^i_{ck}(wk{\cdot}\theta_{jik}) \\ & \wedge Overrides(lab_k, lab_j) \wedge Ei[(zi{\cdot}\theta_{jik})]. \end{aligned} \tag{4jik}$$

where $jikTriple$ has the form:

$$\langle rule_j, \theta_{jik}, rule_k \rangle \tag{$jikTriple$}$$

with associated relating mutex mut_i having the form:

$$\perp \;\leftarrow\; q(ui) \wedge p^i(vi) \;\mid\; Ei[zi]. \tag{mut_i}$$

Here, $rule_k$ has the form:

$$p^i(wk) \;\leftarrow\; Bk[yk]. \tag{$rule_k$}$$

p^i is a predicate, which may possibly be q. wk is the term tuple appearing (as argument tuple to p^i) in the head of $rule_k$. $Bk[yk]$ stands for the body of $rule_k$. yk is the tuple of logical variables that appear in Bk. ui and vi are term tuples of arity appropriate to q and p^i respectively. $\cdot$ stands for the operation of applying a substitution, as usual with unifiers. θ_{jik} stands for $mgu(rule_j, mut_i, rule_k)$, i.e., the maximum general unifier that simultaneously unifies both tj with ui, and vi with wk (recall Definition 7). p^i_{ck} bears the same relationship to $\langle p^i, rule_k\rangle$ as q_{cj} bears to $\langle q, rule_j\rangle$. $Overrides$ stands, as usual, for the prioritization predicate. lab_j is the rule label of $rule_j$. lab_k is the rule label of $rule_k$. If the rule label of $rule_j$ is empty, then lab_j is assigned to be $emptyLabel$. Likewise, if the rule label of $rule_k$ is empty, then lab_k 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 that Ei is a formula (conjunction of literals) whose logical variables are zi. $Ei[(zi{\cdot}\theta_{jik})]$ stands for the result of applying the substitution θ_{jik} to zi, and then substituting that result into Ei.

For each rule $rule_j$ in $RuleLocale(q)$ and each relevant opposition triple $jikTriple$ in $RelOppTriples(rule_j)$, include the rule:

$$q_s(tj{\cdot}\theta_{jik}) \;\leftarrow\; q_u(tj{\cdot}\theta_{jik}) \wedge p^i_u(wk{\cdot}\theta_{jik}) \wedge Ei[(zi{\cdot}\theta_{jik})]. \tag{5jik}$$

Here, p^i_u is a newly introduced predicate that bears the same relationship to p^i as q_u bears to q; i.e., intuitively, pi_u stands for "pi has an unrefuted candidate".
□

We are now ready to define the entire output of transforming a whole GCLP.

Definition 9 (Overall Conflict Resolution Transform For GCLP)
Let $\mathcal{C}$ be a GCLP. The conflict resolution transform's output $\texttt{CR}(\mathcal{C})$ is defined as follows. Let $\mathcal{C}_{NCN}$ stand for $\mathcal{ECN}(\mathcal{C})$, i.e., for the result of applying the ECN transform to $\mathcal{C}$. "NCN" here is mnemonic for "No Classical Negation". Let $Predicates(\mathcal{C}_{NCN})$ stand for the set of all predicates that appear in $\mathcal{C}_{NCN}$ (in the mutex's as well as in the rules).

$$\texttt{CR}(\mathcal{C}) \quad = \quad \bigcup_{q \in Predicates(\mathcal{C}_{NCN})} \texttt{CR}(\mathcal{C}_{NCN}, q)$$

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

Recall from Definition 8 that intuitively, *emptyLabel* represents the empty rule label. Note that *emptyLabel* is introduced only if needed, i.e., iff the set of relevant opposition triples is non-empty. Note also that *emptyLabel* is introduced at most once, i.e., the same *emptyLabel* is shared by all the per-locale CR transforms.

We write this version of the overall conflict resolution transform as `CR_Cour2`, to distinguish it from the previous version (called `CR_Cour1`) in [5]; we write the OLP outputted by this version of the overall as $\texttt{CR\_Cour2}(\mathcal{C})$. □

5 Semantics of GCLP; Inferencing; C3LP and CU2LP

Recall: the semantics overview in section 3.3.1.

PCNMLP syntax, i.e., GCLP syntax, has many possible semantics as a knowledge representation. Among these possible choices, we next define one particular choice.

Semantically, we treat a PCNMLP 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, i.e., the basic form of courteous LP's in [4]). 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 the well-founded semantics (WFS) [8], we define the **model**, (which we also call the **set of conclusions**) of a PCNMLP 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.

Our semantics for PCNMLP syntax, i.e., for GCLP syntax, is defined by compiling GCLP (syntax) to OLP via `CR_Cour2`, and adopting the well-founded semantics (WFS) for the resulting OLP. We call this formalism: version 2 of **unrestricted courteous-flavor** PCNMLP, abbreviated **CU2LP**.

Let $\mathcal{C}$ be a given PCNMLP. It has an *original* set of predicate and function symbols, i.e., ontology. $\texttt{CR\_Cour2}(\mathcal{C})$ 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 `CR_Cour2`($\mathcal{C}$). 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 the *PCNMLP version* of that conclusion set. The PCNMLP-version conclusion set contains classical negation rather than negation predicates. We define the (unadorned) PCNMLP-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 PCNMLP) of the conclusion set.

We summarize all this as follows.

Definition 10 (Compile PCNMLP Via CR_Cour2)

Let $\mathcal{C}$ be a PCNMLP. The CU2LP semantics for $\mathcal{C}$ is defined as the tuple
$\langle \mathcal{C}, \mathcal{O}, \langle T_O, F_O \rangle, \langle T_{PCNM}, F_{PCNM} \rangle \rangle$
Here, the post-transform (adorned) OLP $\mathcal{O}$ is `CR_Cour1`($\mathcal{C}$). $\langle T_O, F_O \rangle$ is the (WFS OLP) model for $\mathcal{O}$. $\langle T_{PCNM}, F_{PCNM} \rangle$ is the unadorned PCNMLP version of $\langle T_O, F_O \rangle$. □

Next, we use the `CR_Cour2` transform and the compilation approach to define a generalized version of courteous LP's. To keep to the spirit of "courteous"-ness cf. C1LP (i.e., cf. the basic version of courteous LP's in [4]), we wish to have some strong semantic guarantees about well-behavior that are similar to those in C1LP. Accordingly, we thus restrict CU2LP somewhat so as to ensure such well-behavior.

Definition 11 (Courteous LP's, Version 3)

Let $\mathcal{C}$ be a CU2LP. We say that $\mathcal{C}$ is a generalized, i.e., *version-3*, courteous LP, abbreviated as **C3LP**, when the following three restrictions are satisfied:
(1.) `CR_Cour2`($\mathcal{C}$) is locally stratified.
(2.) The mutex's (including any implicit classical mutex's) specify a collection of disjoint opposition-locales. We call this the "**one-of**" restriction on the mutex's/opposition.
(3.) Prioritization is a strict partial order *"within"* every opposition-locale.

By (2.), we mean that after ground-instantiating the mutex's (including any implicit classical mutex's), the opposition specified by those mutex's is equivalent to: a collection of ground opposition-locales that are disjoint. (Recall the definition of "opposition-locale" in section 3.3.1.) Here, by "disjoint"-ness of two opposition-locales A and B, we mean that the set of ground classical literals in A does not intersect with the set of ground classical literals in B. Here, the opposition within each ground opposition-locales is unconditional. In summary, the one-of restriction says that the mutex's specify opposition that is equivalent to a disjoint collection of opposition-locales, where each opposition-locale specifies that at most one of a set of ground classical literals is permitted to be concluded.

An important useful form of conditional mutex that obeys the one-of restriction, is:
$\perp \ \leftarrow \ P(?X1, \ldots, ?Xk, ?Y) \wedge P(?X1, \ldots, ?Xk, ?Z) \ \mid \ (?Y \ \neq ?Z)$. This specifies that the

predicate P is partial-function-al, i.e., that for a given value (instance) of P's first k arguments, one is permitted to conclude at most one value for P's last argument. After instantiating it, this mutex is equivalent to a set of unconditional mutex's, one for each instance of the first k arguments and disjoint pair of instances for the last argument. E.g.,

$$\perp \;\leftarrow\; \mathit{giveDiscount}(?Cust, ?YPercent) \wedge \mathit{giveDiscount}(?Cust, ?ZPercent)$$
$$\mid\; (?YPercent \neq ?ZPercent).$$

is equivalent, after instantiation, to a set of unconditional mutex's:

$$\perp \;\leftarrow\; \mathit{giveDiscount}(Joe, 23\%) \wedge \mathit{giveDiscount}(Joe, 31\%).$$
$$\perp \;\leftarrow\; \mathit{giveDiscount}(Joe, 31\%) \wedge \mathit{giveDiscount}(Joe, 42\%).$$
$$\perp \;\leftarrow\; \mathit{giveDiscount}(Joe, 23\%) \wedge \mathit{giveDiscount}(Joe, 42\%).$$
$$\perp \;\leftarrow\; \mathit{giveDiscount}(Ann, 23\%) \wedge \mathit{giveDiscount}(Ann, 31\%).$$
$$\perp \;\leftarrow\; \mathit{giveDiscount}(Ann, 31\%) \wedge \mathit{giveDiscount}(Ann, 42\%).$$

. . . .

By (3.) we mean the following. For every opposition locale, the set of *Overrides* tuples in T_O is a strict partial order when restricted to the set of rule labels appearing in $\mathit{RuleLocale}(p) \cup \mathit{RuleLocale}(p^{\neg})$, where p is the locale predicate. □

6 Examples

For examples of GCLP's, including the output of the transform and the results of inferencing, see the many examples included in the alpha release of the IBM CommonRules prototype, available at http://www.research.ibm.com/rules/ in summer 1999. Also, a long example of a GCLP (with unconditional mutex's) is given in [6]: its domain is personalized discounting and promotions in a Web bookstore storefront. In addition, a number of examples of BCLP's (i.e., less-general-form GCLP's, where mutex's are classical only) are given in [3] (extending [4]) and [5].

7 Well-Behavior

In a future version of this paper, we will give formal well-behavior results for GCLP, i.e., for C3LP and CU2LP. The well-behavior properties of C3LP and CU2LP are similar to those given in [5] (for C2LP and CU1LP there, respectively).

CLP always produces a **consistent** set of conclusions, enforcing all the mutex's. A number of other well-behavior properties also hold for CLP, including about merging and about natural behavior of prioritization; however, we defer detailing those until a future version of this paper.

The courteous compiler in effect represents the prioritized argumentation process in OLP. It introduces some extra "*adorning*" predicates to represent the intermediate stages of argumentation: candidates, unrefuted candidates, etc..

The courteous compiler can be "instrumented" to raise an alarm when unresolved conflict occurs, via introducing further adorning predicates that represent skeptical defeat.

The computational time and space complexity of the `CR_Cour2` transform is worst-case cubic, i.e., $O(n^3)$ where n is the size of the input GCLP, i.e., of the input PCNMLP. From the tractability of courteous compilation, it follows directly that Courteous LP inferencing under the VBD restriction is tractable: for CU2LP and thus for its special case CU2LP. Here, we say that an LP is "*VBD*" when either (1.) it is ground, or (2.) it has no logical functions of non-zero arity (a.k.a. the *Datalog* restriction), and it has a bounded number of logical variables appearing in each rule. The VBD restriction is commonly met in practice.[1] Under the VBD restriction, Courteous LP inferencing

---

[1] It is usually straightforward to representationally reformulate a rule-set so as to replace a logical function f

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 $v + 2$.

---

having arity $k > 0$ by a predicate fp having arity $k + 1$, where intuitively the $(k + 1)^{th}$ argument corresponds to the result of applying the function.

References

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

[2] 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.

[3] 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 [4].

[4] 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 .

[5] Benjamin N. Grosof. Compiling Prioritized Default Rules Into Ordinary Logic Programs. 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. USA, May 1999. IBM Research Report RC 21472.

[6] 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 IBM Research Report RC21473, http://www.research.ibm.com, search for Research Reports; P.O. Box 704, Yorktown Heights, NY 10598, USA.

[7] 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.

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