RC 20836 (92273) 08 May 97; revised 30 Dec 97
Computer Science

IBM Research Report

Courteous Logic Programs:

Prioritized Conflict Handling For Rules

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@watson.ibm.com
(or grosof@us.ibm.com or grosof@cs.stanford.edu)
Web: http://www.research.ibm.com/people/g/grosof

Abstract

We define courteous logic programs, an expressive superclass of general logic programs, for the acyclic case. Courteous LP's feature not only classical negation as in extended LP's (Gelfond & Lifschitz), but also prioritized conflict handling. We show courteous LP's always have a consistent and unique answer set, which can be computed in $O(m^2)$ time, where m is the size of the ground-instantiated program, as compared to $O(m)$ time for general LP's. Courteous LP's provide a method to resolve conflicts that arise in authoring (specifying), updating, and merging. This is especially useful for creation of rule-based intelligent agents by non-technical authors, e.g., for commercial applications such as personalized information filtering and workflow. Current work includes: implementing courteous LP's for such applications, in IBM's RAISE system; generalizing expressively, e.g., to permit recursion; and developing methods for interactive acquisition of rules, e.g., conflict analysis and inter-agent communication.

Summaries of this paper: This paper includes various lengths of summaries of itself. In ascending order of size, see: this Abstract, the Long Abstract, the Introduction and Overview (Section 1), and the Talk Slides (Appendix B).

Intended Audience: This paper has been written so as to direct it towards an audience familiar with **logic programming**. However, **Appendix A includes review** of background logic programming concepts.

Publication Information: This paper is a considerably extended version of our earlier paper entitled "Prioritized Conflict Handling for Logic Programs" which appeared in the Proceedings of the 1997 International Logic Programming Symposium (i.e., of [21]). (That earlier paper does not include proofs.) This paper's fundamental technical content also essentially supersumes that of our previous paper entitled "Practical Prioritized Defaults Via Logic Programs", IBM Research Report RC 20464, which in turn is an extended version of the paper that appeared at the International Workshop on Non-Monotonic Reasoning, held in July 1996. That previous paper is directed towards an audience familiar with non-monotonic reasoning and artificial intelligence.

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

Keywords: logic programming, default reasoning, priorities, non-monotonic reasoning, intelligent agents, rules. More keywords: applications, knowledge acquisition, usability, locality, modularity, tractability, negation-as-failure, Prolog, updating, belief revision, advice-taking, artificial intelligence, machine learning, deductive databases.

Contents

Long Abstract

We define acyclic (non-recursive) courteous logic programs, an expressive superclass of acyclic ordinary ("general") logic programs. Courteous logic programs are equipped with classical negation and prioritized conflict handling. We show courteous inferencing is computationally tractable for the (acyclic) propositional case, e.g., under the Datalog restriction.

As in extended logic programs [11], classical negation is permitted in rule heads and bodies (in addition to negation-as-failure in the bodies). Rules having head p may thus conflict with rules having head $\neg p$. In extended logic programs, such conflict results in global inconsistency; every literal is a conclusion.

Courteous logic programs feature a disciplined form of conflict handling that guarantees a consistent and unique set of conclusions (answer set). Partially-ordered prioritization among rules is optionally specified explicitly via a prioritization sub-program: a (possibly empty) set of facts about a reserved predicate $Overrides$. Each rule has an optional label; $Overrides(label1, label2)$ specifies that a rule with $label1$ has higher priority than a rule with $label2$.

The courteous approach hybridizes ideas from the field of general non-monotonic reasoning with those of logic programming. Each rule $head \leftarrow body1$ is treated as a default: if $body1$ succeeds, then $head$ will succeed unless there is an unrefuted conflicting rule $\neg head \leftarrow body2$ whose $body2$ succeeds. Refutation is based on strict priority. Conflict unresolved by strict priority is treated skeptically: neither $head$ nor $\neg head$ is concluded.

Prioritized conflict handling is useful to represent updating and merging, as well as specificity and inheritance with exceptions, as we illustrate with several examples. Classical negation plus consistency offers the convenient capability to infer $\neg p$ from rules (as well as p from rules), while still enforcing mutual exclusion between p and $\neg p$.

We show that the entire courteous answer set can be computed in $O(m^2)$ time, where m is the size of the ground-instantiated program (i.e., the propositional representation), as compared to $O(m)$ time for ordinary logic programs. This is a relatively low overhead to pay for adding the features of prioritized conflict handling and classical negation.

By contrast, conflict handling in most expressively powerful formalisms for prioritized default reasoning is an additional source of NP-hard complexity beyond the base reasoning. Key to courteous programs' computational and conceptual simplicity is that conflicts are resolved locally: by refutation and skepticism among rules that mention (positively or negatively) the same head atom.

Courteous LP's provide a method to resolve conflicts that arise in authoring (specifying), updating, and merging. This is especially useful for creation of rule-based intelligent agents by non-technical authors, e.g., for commercial applications such as personalized information filtering and workflow.

Current work includes: implementing courteous LP's for such applications, in IBM's RAISE agent-builder system; generalizing expressively, e.g., to permit recursion; and developing methods for interactive acquisition of rules, e.g., conflict analysis and inter-agent communication.

For yet longer summaries of this paper, see the Overview (i.e., Section 1) and the Talk Slides (Appendix B).

1 Introduction and Overview

Our aim in developing courteous logic programs is to improve the expressive convenience of logic programs, especially for applications in intelligent information agents. We call the formalism "courteous" for two reasons. First, it respects precedence, i.e., priority relationships between rules. Second, even in the presence of conflict between rules, it is "well-behaved" in the sense of there being a consistent, tractably computable, and unique set of conclusions.

An interesting application area for logic programming that motivates our work is "information-flow" applications enhanced by rule-based intelligent agents. In these applications, agents control the flow of information items. Their tasks include not only finding and filtering, but also categorizing, prioritizing for attention, storing and managing, monitoring and notifying, and selectively forwarding, disseminating and sharing.

IBM has released Agent Building Environment (ABE) as a toolkit product alpha, for building such applications. It is currently available free on the World Wide Web (see http://www.raleigh.ibm.com/iag/iaghome.html). ABE is based on our group's research system RAISE (Reusable Agent Intelligence Software Environment) [22] [23] [20] (also see author's Web address for more). ABE/RAISE's approach revolves around a logic program. In the currently released version of ABE, this logic program is acyclic (non-recursive), Datalog, and positive (without negation-as-failure). Inferencing is in the forward direction, and is exhaustive; i.e., all conclusions are generated. An innovative feature in ABE/RAISE is its patent-pending approach to "situating" the logic program's reasoning by augmenting it with clean and dynamic procedural attachments for perception and action. IBM has already built several practical intelligent agents applications based on ABE/RAISE: in e-commerce shopping, customer service, e-mail, and netnews. Several others are underway, mostly in cooperation with IBM's customers and business partners.

In this application area, it is valuable to facilitate authoring (i.e., specifying) of logic programs by relatively non-technical users, e.g., as part of personalizing rule-based intelligent agents. Rule sets are not "shrink-wrapped" with the application. Rather, a user is the "domain expert" for her own "workflow", e.g., for specifying her mail handling or shopping interests. These users specify their rule sets via forms and templates that are often application-specific. "Under the covers", the rules are then re-formatted as a more standard logic program.

We are attracted by the expressive power of classical negation as in extended logic programs [11]. There, classical negation is permitted in rule heads and bodies (in addition to negation-as-failure in the bodies). This offers the convenient capability to reason in a first-class way about both sides of a proposition p (say, "Highly Important" for e-mail): i.e., about $\neg p$ as well as about p, in particular to infer $\neg p$ from rules (as well as p from rules). However, a rule $p \leftarrow Body1$ may *conflict* with a rule $\neg p \leftarrow Body2$. I.e., informally, both $Body1$ and $Body2$ may succeed (fire), creating a conflict about whether p or $\neg p$ should succeed. In extended logic programs, such conflict results in global inconsistency ("blow-up"); every literal is a conclusion. This is similar to the situation in classical logic, in which an inconsistent theory implies any sentence as a conclusion.

A difficulty with employing extended logic programs is thus that it is relatively easy to get conflicts in the rule sets, especially when authored by relatively non-technical end-users as in our intelligent information agents applications scenarios.

Also, it is desirable in our applications scenarios to facilitate modularity and merging of rule sets (including advice-taking, i.e., automatic merging based on inter-agent knowledge-level communication). This is important because it is relatively expensive in human effort to specify and debug rule sets. Re-use and sharing are highly advantageous; it's nice if a user (or his agent) can swap rule sets with his "friends", e.g., co-workers. The catch, though, is that conflicts can arise relatively easily as a result of merging.

Another difficulty is that the presence of two forms of negation, i.e., having negation-as-failure in addition to classical negation, is potentially quite conceptually confusing, again especially for relatively non-technical end-users.

For inspiration in grappling with these challenges, we have drawn on the idea of *prioritized defaults* (e.g., [30] [27] [14] [17] [15] (ch. 2 includes a literature review), and [3] (also includes a literature review)) from the field of general non-monotonic reasoning. As has been explored there (and in the related knowledge representation and common-sense reasoning literatures), it is often easier and more natural, especially for non-technical end-users, to specify rules in the manner of prioritized defaults. "Default" roughly means that a rule antecedent can succeed without the rule consequent succeeding: it may be that the rule consequent is blocked by another conflicting rule. "Prioritized" roughly means that in case of conflict between two rules, the rule with (strictly) higher priority has its consequent succeed, and the rule with lower priority does not.

We are attracted to partially-ordered prioritization in that it is relatively weak, qualitative information, yet it suffices to resolve conflicts. It can be specified by pairwise comparison of rules.

Prioritized defaults suffice as a representational approach for many interesting cases of conflictful reasoning, including: updating in (deductive-)database-flavor fashion, where more recent premises override previous premises; specificity dominance and inheritance with exceptions in which a more specific rule overrides a less specific rule; and legalistic regulations in which a rule whose source has greater authority (e.g., jurisdictional) overrides a rule whose source has less authority.

Taking all this together, we are motivated to provide a mechanism that can not only represent classical negation, but also handle conflicts and priorities in fashion akin to prioritized defaults. Speaking philosophically and slangily, we might say that extended logic programs "give up" in the face of conflicts, whereas we want in developing courteous logic programs to "deal with" conflicts, take them in stride, even "munch on" them.

The courteous logic program formalism meets our desiderata to a considerable extent. First, it preserves overall consistency in the presence of conflict. Conclusions not affected by the conflict are still inferred. (It "does something reasonable" until the the rule set is "totally debugged".)

Second, it provides a simple way to specify override: an optional label for each rule, plus a reserved binary predicate *Overrides* that takes rule labels as arguments. An *Overrides* fact as part of the overall logic program then specifies a pairwise priority comparison.

Third, inferencing in the formalism is tractable, under the Datalog restriction. By contrast, conflict handling in most expressively powerful formalisms for prioritized default reasoning (e.g., based in Default Logic [34] or circumscription [29]) is an additional source of NP-hard complexity beyond the base reasoning [13] [7]. Tractability makes applications practical beyond small scale.

Fourth, there is a unique answer set (i.e., set of conclusions). This helps provide a conceptually simple semantics, which facilitates understandability, especially by non-technical end-users. By contrast, many expressively powerful non-monotonic reasoning formalisms have multiple extensions (e.g., Prioritized Default Logic variants; see [3] for review).

Fifth, reassuringly, the (acyclic) courteous logic programs class includes (acyclic) consistent extended logic programs as a sub-class, both syntactically and semantically. (Acyclic) general logic programs [1] are essentially a sub-class of consistent extended logic programs (the only difference is that in the usual definition, $LP \not\vdash atom$ in general LP's is treated also as $LP \vdash \neg atom$, i.e., the closed world assumption is applied to all atoms).

Sixth, the courteous formalism enables one to avoid many typical uses of negation-as-failure, e.g., closed world assumption, blocking a less specific rule when a more specific rule applies, or blocking a less recent rule when updating with a more recent rule. The expressive mechanisms of classical negation, defaults, and priorities provide a more disciplined and modular way to achieve the same effect, in many cases. Obviating the need for negation-as-failure helps reduce the potential confusion, especially for non-technical end-users, caused by the presence of two forms of negation.

We have found in our previous work on ABE/RAISE that the acyclic Datalog restriction suffices for many interesting applications. Motivated by that work, we are especially interested in exhaustive forward inferencing, i.e., in computing the entire answer set. This can be viewed as a kind of

[1] under the usual, e.g., stratified, semantics; see Theorem 15 for details

deductive database.

Key to the courteous formalism's computational and conceptual simplicity is that conflicts are resolved **locally**: by refutation and skepticism among rules that mention (positively or negatively) the same head atom.

Tractability and uniqueness of the answer set are highly attractive properties, in our view. They make reasoning be practical: not only exhaustive forward inferencing, but also rapidly iterated belief revision. But more than that, they make authoring rules (knowledge acquisition) be more natural. They make it much easier to understand and predict the reasoning behavior, and thus to debug and to trust the program. Locality is a major help too. The author of rules (e.g., a non-technical end-user) need focus only on the set of rules in a locale, in a modular fashion. Tractability and locality enable a human to "simulate in his head" what the rules will "do" and thus what they "mean", even in relatively large rule sets. Modularity, understandability, and predictability are often crucial requirements for practical usage, e.g., as we have found at IBM in our agent-building experience.

2 Preliminary Definitions; Extended LP's

Background: We assume the reader is familiar with extended LP's [11], with the semantics of stratified (ordinary, non-extended) logic programs with negation as failure (e.g., [33]), and with the standard concepts in the logic programming literature (e.g., as reviewed in [2]), 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 LP's cf. [11].

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* 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, a "*general*" logic program is one in which each literal L_j above is an atom, i.e., where no classical negation is permitted.

The semantics of extended LP's treats a rule with variables as shorthand for the set of its ground instances. We will do likewise with courteous LP's. We write $\mathcal{E}^{instd}$ to stand for the LP that results when each rule in $\mathcal{E}$ having variables has been replaced by the set of all its possible ground **instantiations**. The semantics of extended LP's is further defined using the concept of an **answer set** (i.e., set of conclusions); again, we will do likewise with courteous LP's. An answer set is a subset of the ground literals. We write $\models$ to stand for truth relative to an answer set.

As we discussed in section 1, as Gelfond & Lifschitz define its semantics, an extended LP may be *contradictory*, i.e., **inconsistent**: it may have an inconsistent answer set. An answer set is inconsistent if it contains a pair of complementary literals; indeed in their semantics, an inconsistent extended LP has one answer set which is the set of all ground literals. For example, the extended LP consisting of the two conflicting rules

$$p \leftarrow$$
$$\neg p \leftarrow$$

is inconsistent. The answer set contains both the literals p and $\neg p$.

Observe that restricting extended LP's to be acyclic or stratified (and/or Datalog) does *not* ensure their consistency, e.g., as in the above example.

3 Definition: Courteous Logic Programs

Syntactically, a courteous logic program is defined as a restricted class of extended logic programs in which, additionally, rules have labels. These labels are used as handles for specification of prioritization between rules.

Definition 1 (Labelled Rule)

A *labelled rule* has the form:

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

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. □

Definition 2 (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. □

Definition 3 (Prioritization Sub-Program)

A *prioritization sub-program* is defined as a set, possibly empty, of positive ground facts about $Overrides$. Each of these is called a *prioritization fact*.

The prioritization relation $Overrides$ specified by the prioritization sub-program is required to be a **strict partial order**, i.e., transitive and anti-symmetric (and irreflexive). □

Since each prioritization fact is a rule of the program, it may in principle have a label. As we will see, however, such labels are effectively ignored semantically.

Definition 4 (Courteous LP: Syntax)

A courteous logic program $\mathcal{C}$ is defined as the disjoint union of a main (sub-)program with a prioritization sub-program:

$$\mathcal{C} \;=\; \mathcal{C}_{main} \;\dot{\cup}\; \mathcal{C}_{Overrides}$$

$Overrides$ is syntactically reserved: it must not appear in $\mathcal{C}_{main}$ but rather appear only within the prioritization sub-program.

$\mathcal{C}$ (i.e., its ground-*atom* dependency graph) is required to be **acyclic.** [2] □

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 $\mathcal{C}$.

Note that adding a prioritization sub-program does not affect the acyclicity of a program.

Terminology: Relative to $\mathcal{C}$: the **definitional locale** for a ground *atom* p, written as $Defn(p)$, is defined as the (possibly empty) subset of rules within $\mathcal{C}^{instd}$ in which p appears in the rule head (positively or negatively).

Let $\rho = p_1, \ldots, p_m$ be a sequencing of all the (ground) atoms of $\mathcal{C}^{instd}$. We say that ρ is a **total stratification** of the atoms when ρ is a reverse-direction topological sort of the atom dependency graph. "Reverse" here means that body comes before head.

Terminology: By **shallower** (respectively, **deeper**), we mean later (respectively, earlier) in the stratification sequence. ρ is thus a sequence in which deeper locales precede shallower locales. Observe that if atom b depends on atom a (i.e., in the atom depdendency graph), then b is shallower than a.

Associated with the total ordering of the atoms is the associated totally ordered partition of $\mathcal{C}^{instd}$'s rules into definitional atom locales $Defn(p_1), \ldots, Defn(p_m)$. Each *stratum*, i.e., element in this partition, is a single atom's definitional locale. □

Definition 5 (Courteous LP: Semantics)

$\mathcal{C}$ has a unique answer set S, defined as follows.

Let ρ be a total atom stratification of $\mathcal{C}$, such that all of the prioritization (i.e., $Overrides$) atoms come before all the other (i.e., main) atoms. (There may be several such total stratifications; the choice among them does not matter.)

Let p_i stand for the i^{th} (ground) atom in this sequence ρ. The answer set is constructed iteratively:

$S_0 = \emptyset$

$S_i = \bigcup_{j=1,\ldots,i} \; T_j \;\; , \;\; i \geq 1$

$S = \bigcup_i T_i$

where $\emptyset$ stands for the empty set,

[2]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*.

$T_i =$
$\{\sigma p_i \mid Cand_i^{\sigma} \neq \emptyset \;,\; \forall k \in Cand_i^{\neg\sigma}.\ \exists j \in Cand_i^{\sigma}.\ S_{i-1} \models Overrides(j,k)\}$
$Cand_i^{\sigma} = \{j \mid labels(j,r) \;,\; Head(r) = \sigma p_i \;,\; S_{i-1} \models Body(r)\}$

Here, σ stands for a classical sign, either positive (sometimes written as $+$) or negative ($\neg$, also written as $-$). $labels(j,r)$ stands for: j is the label for rule r. $Head(r)$ and $Body(r)$ stand for the head and body of rule r, respectively. Note that $Head(r) = \sigma p_i$ implies that $r \in Defn(p_i)$.

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 $\mathcal{C}$'s language). □

Notation: $\mathcal{C} \mid\!\sim p$ means that p is in the answer set of $\mathcal{C}$.

Explanatory Description: The answer set is defined incrementally and constructively, by means of a series of partial answer sets S_i, that are built up by iterating along a total atom stratification, generating conclusions for each ground atom (and thus each predicate) along the way.

The i^{th} stratum in the total atom stratification contributes an increment T_i to the answer set. Recall that a stratum corresponds to the atom p_i's definitional locale. T_i is the **conclusion**, if any, resulting from the rules in that atom locale. The set T_i either consists of a single ground literal, or is empty. The literal is of positive sign, i.e., p_i, or it is of negative sign, i.e., $\neg p_i$.

T_i is the **winner**, if any, that results from **prioritized competition among candidate arguments**. A candidate argument is generated by a rule $r \in Defn(p_i)$ whose body $Body(r)$ successfully "fires" in the sense of being true in the previous answer set iterate S_{i-1}, in which iterate the earlier strata's conclusions have accumulated. Such a rule has either p_i or $\neg p_i$ as its head $Head(r)$. Candidates are represented by their labels and are collected into two sets. There is one such set for each sign σ: $Cand_i^{\sigma}$, which can be viewed as a **team** arguing for the same conclusion, namely σp_i.

The prioritized competition among candidates can be viewed in terms of these two **opposing** teams. If both teams are empty, then there is no winner. Otherwise, if one team is empty, then the other (non-empty) team wins. The most interesting case is when both teams are non-empty. This case corresponds to **conflict**. In extended logic programs, conflict results in inconsistency. In this case (and only in this case), the prioritization $Overrides$ comes into play. **One team wins iff every member of the opposing team is refuted**. **Refutation is based on prioritization**: one candidate argument with label j refutes another candidate argument with label k iff j has higher priority than k, i.e., if $Overrides(j,k)$ is true in the previous answer set iterate S_{i-1}. Note that $Overrides(j,k)$ is true iff

$Overrides(j,k) \;\; \leftarrow$

is present as a prioritization fact in the prioritization sub-program $\mathcal{C}_{Overrides}$. If neither team is refuted, then neither team wins: i.e., the **teams can defeat each other**. (Note that due to the strict partial order property of the prioritization *Overrides*, it cannot happen that both teams refute each other.) This outcome of mutual defeat corresponds to **skepticality**: if there are unrefuted candidate arguments both for p_i and for $\neg p_i$, then no conclusion about p_i is drawn; intuitively, there is then no strong justification to believe either, so no "commitment" is made.

Figure 1 illustrates the above semantic steps for an atom locale. It emphasizes three phases: 1) rules firing to produce candidates, 2) candidates refuting each other based on priority, and 3) skepticism being applied.

Remarks: Our definition of courteous logic programs is as a pure formalism. It therefore supports inferencing in both the **forward** direction (as in the intelligent agents applications discussed in section 1) and the **backward** direction, i.e., query-answering (as in Prolog).

The prioritization (*Overrides*), refutation, and skepticality are **local**, in the sense that they apply directly only within the scope of each atom locale, then ramify indirectly to the entire program.

If a locale contains only rules whose heads share the same sign, then we say that the locale is **one-sided**. If a locale is one-sided, then there can be no conflict within it, and the prioritization within it is irrelevant.

In particular, since the prioritization sub-program is one-sided, its rules' labels and the prioritization among them (if, indeed, it has any) are irrelevant, and can thus be omitted equivalently.

Since *empty_label* cannot appear in the prioritization sub-program, any rule with empty label effectively does not participate in strict prioritization. More generally, if the prioritization sub-program is empty, then labels are superfluous.

In general, the prioritization relation *Overrides* may compare two labels i and j that belong to different locales. Such priority is ignored by the semantics. Similarly, prioritization facts mentioning non-label arguments may be permitted, but are ignored.

Lemma 6 (Independence of Total Stratification Choice)
The courteous LP answer set of Definition 5 is independent of the (non-deterministic) choice of total (atom) stratification. $\Box$

Proof : Our approach is to analyze the inferential dependencies.

By the definition of the construction, the conclusion drawn about p_i in p_i's locale depends only on the previous conclusions drawn about: the atoms in the locale's rules' bodies, plus the priority atoms relevant to the locale. (The priority atoms relevant to the locale are those that compare two labels that appear in the locale's rules.) But by our definition of total stratification, any total stratification has the following property: all of the atoms appearing in p_i's rules' bodies, and all the prioritization (*Overrides*) atoms, came before p_i in the sequence. Intuitively, those atoms' locales have been tried and milked for all they are worth. Thus any choice of total stratification results in the same T_i. Since this is true for each p_i, the overall answer set is thus the same no matter which total stratification is chosen.

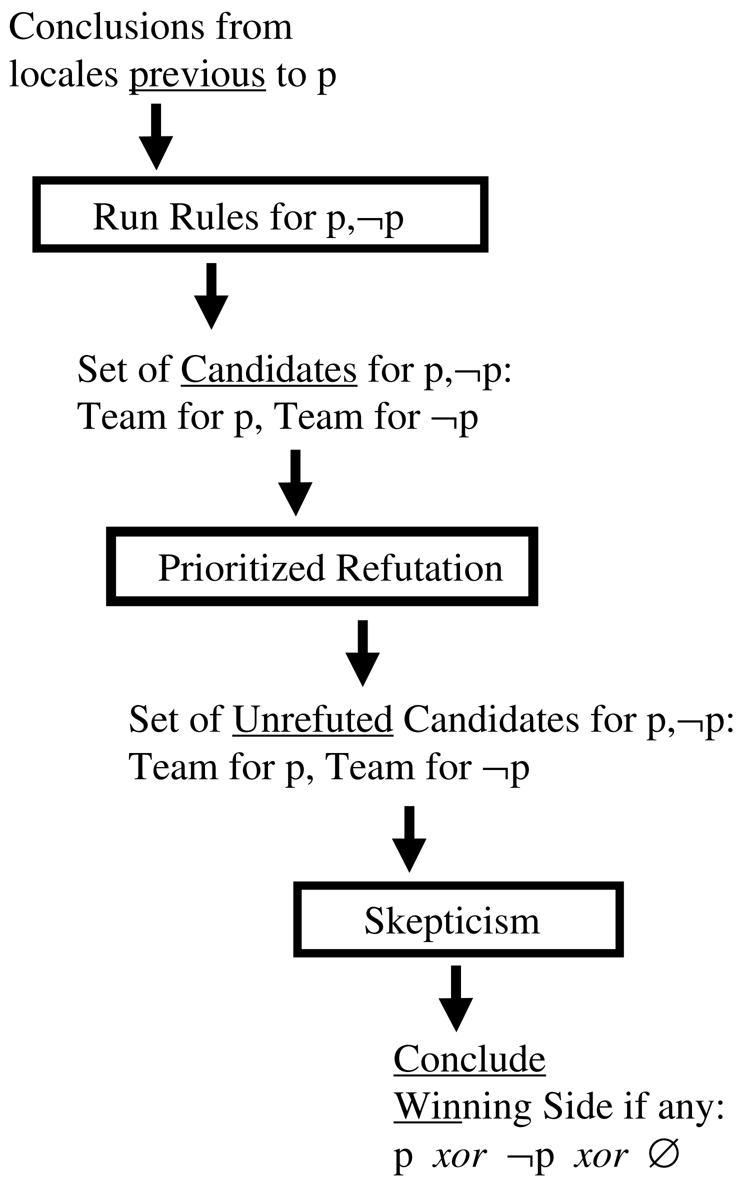

Figure 1: Semantic Steps of Inferencing Within the p Atom Locale

Actually, above we skipped over a subtlety: the special case of prioritization ($Overrides$) atom locales. Each prioritization atom's locale depends on no other atoms. It does not depend on any other atoms via the locale's rules' bodies because each prioritization rule is required by definition to have an empty body. Nor does it depend on prioritization atoms that compare the (labels) of rules (and thus candidates) within the locale: there can be no conflict within the locale, since the locale is one-sided (the prioritization sub-program is required by definition to be positive). Therefore, the prioritization atoms can "go first" (in the total stratification sequence, before the non-prioritization atoms) with no problem. □

4 Initial Examples

4.1 Nixon Diamond

Example 7 (Pacifism; Skepticality)

Consider the well-known Nixon Diamond example of conflict in default reasoning. This can be straightforwardly represented by program $\mathcal{C}_1$ consisting of:

$$\begin{array}{lrcl} \langle Qua \rangle & Pacifist(x) & \leftarrow & Quaker(x) \\ \langle Rep \rangle & \neg Pacifist(x) & \leftarrow & Republican(x) \\ & Quaker(Nixon) & \leftarrow & \\ & Republican(Nixon) & \leftarrow & \end{array}$$

Interpreted as an extended logic program, the first two rules conflict for instance $Nixon$, resulting in inconsistency of the entire program.

By contrast, the courteous interpretation behaves skeptically and consistently. The answer set is simply

$$\{Quaker(Nixon), Republican(Nixon)\}$$

, with no conclusion drawn either way about Nixon's Pacifism.

Next, consider program $\mathcal{C}_2$ in which the Republican rule is given higher priority, by adding the rule:

$$Overrides(Rep, Qua) \leftarrow$$

The (courteous) answer set now includes the conclusion $\neg Pacifist(Nixon)$. We find this to be a simple and intuitively natural representation. □

4.2 Inheritance and Specificity

A common pattern of priority is to have a more specific exception-case rule override a more general-case rule, e.g., in default inheritance reasoning.

Example 8 (Molluscs; Inheritance; Specificity Priority)

Etherington & Reiter's [9] example of a default inheritance hierarchy about molluscs can be straightforwardly represented, with prioritization corresponding to specificity, as the following program. There is no need to add the extra "interaction" conditions that make the defaults be non-normal there (in Default Logic, the formalism they used).

$$
\begin{array}{lrcl}
 & Mollusc(x) & \leftarrow & Cephalopod(x) \\
 & Cephalopod(x) & \leftarrow & Nautilus(x) \\
\langle Mol \rangle & ShellBearer(x) & \leftarrow & Mollusc(x) \\
\langle Cep \rangle & \neg ShellBearer(x) & \leftarrow & Cephalopod(x) \\
\langle Nau \rangle & ShellBearer(x) & \leftarrow & Nautilus(x) \\
 & Overrides(Nau, Cep) & \leftarrow & \\
 & Overrides(Cep, Mol) & \leftarrow & \\
 & Overrides(Nau, Mol) & \leftarrow &
\end{array}
$$

For example, with

$$
\begin{array}{l}
Mollusc(Molly) \ \leftarrow \\
Cephalopod(Sophie) \ \leftarrow \\
Nautilus(Natalie) \ \leftarrow
\end{array}
$$

the answer set for the $ShellBearer$ predicate locale is
$\{ShellBearer(Molly), \neg ShellBearer(Sophie), ShellBearer(Natalie)\}$.

By contrast, in Etherington & Reiter's style of representation without prioritization, the Mol and Cep rules above are modified to have extra "interaction" conditions involving negation-as-failure:

$$
\begin{array}{lrcl}
\langle Mol \rangle & ShellBearer(x) & \leftarrow & Mollusc(x) \wedge \sim CEPHALOPOD(x) \\
\langle Cep \rangle & \neg ShellBearer(x) & \leftarrow & Cephalopod(x) \wedge \sim NAUTILUS(x) \ \square
\end{array}
$$

4.3 Personal E-Mail Agents

Next, we give two examples in which a rule base controls handling of e-mail in a personal intelligent agent. One aspect of such handling is classifying mail into high importance vs. lower importance.

Example 9 (Mail Importance: Stores and Deliveries)

Karen has some rule-form knowledge she wants her agent to implement on her behalf. Mail from a retail store should be treated as not highly important (typically, it's junk). Mail from someplace from which Karen is awaiting a delivery should be treated as highly important. These two rules can be represented as:

$$
\begin{array}{lrcl}
\langle Jun \rangle & \neg Important(msg) & \leftarrow & From(msg, x) \wedge Retailer(x) \\
\langle Del \rangle & Important(msg) & \leftarrow & From(msg, x) \\
 & & & \wedge AwaitingDeliveryFrom(Karen, x)
\end{array}
$$

Also, Karen has various other facts and knowledge, e.g.,

$$AwaitingDeliveryFrom(Karen, ParisCo)$$

In addition, Karen has access to a shared background knowledge base containing facts about organizations. She includes this information in her knowledge base. E.g.,

$$
\begin{array}{l}
Retailer(FaveCo) \ \leftarrow \\
Retailer(BabyCo) \ \leftarrow \\
Retailer(ParisCo) \ \leftarrow
\end{array}
$$

Now Karen receives a mail item:

$$From(msg54, BabyCo) \leftarrow$$

The courteous program draws the conclusion $\neg Important(msg54)$. So far, so good — Karen is pleased when she inspects a trace of what her agent has done on her behalf. After a while, another mail item arrives:

$$From(msg81, ParisCo) \leftarrow$$

The courteous program has a conflict between the first two rules, and hence skeptically and consistently draws no conclusion about $Important(msg81)$. When Karen inspects the trace, she exclaims "Oops!"'. She wants this kind of item to be treated as high importance — the delivery rule should win over the junk rule. This is easy to remedy: all she has to do is to add the prioritization fact

$$Overrides(Del, Jun) \leftarrow$$

Things go smoothly until after a while arrives the mail item

$$From(msg117, FaveCo) \leftarrow$$

The courteous program concludes $\neg Important(FaveCo)$. However when Karen inspects the trace, she again exclaims "Oops!". She wants this kind of item to be treated as high importance — because it is from FaveCo, one of her favorite stores. Again, this is easy to remedy: all she has to do is to add a new rule, and prioritize it as an exception override to the junk rule:

$$\langle Fav \rangle \quad Important(msg) \leftarrow From(msg, FaveCo)$$
$$Overrides(Fav, Jun) \leftarrow \qquad \Box$$

Example 10 (Mail Importance: Family)

Fred has three rules he wants his agent to implement on his behalf. Mail from a close family member has high importance. But mail from Aunt Daisy does not (she tends to waste Fred's time and Fred does not like her much); this rule is an exception to the first, and hence is given higher priority than it. E-mail notifying Fred of a personal emergency has high importance; in case of conflict with any other rules, this rule should win, hence it is given higher priority than them. Fred can represent these rules straightforwardly as:

$$\langle Clo \rangle \quad Important(msg) \leftarrow From(msg, x) \wedge CloseFamily(x, Fred)$$
$$\langle Dai \rangle \quad \neg Important(msg) \leftarrow From(msg, AuntDaisy)$$
$$\langle Eme \rangle \quad Important(msg) \leftarrow NotificationOf(msg, es) \wedge PersonalEmergency(es)$$
$$Overrides(Dai, Clo) \leftarrow$$
$$Overrides(Eme, Dai) \leftarrow$$
$$Overrides(Eme, Clo) \leftarrow$$
$$PersonalEmergency(s) \leftarrow SevereIllnessOf(s, x) \wedge CloseFamily(x, Fred)$$
$$CloseFamily(Betty, Fred) \leftarrow$$
$$CloseFamily(AuntDaisy, Fred) \leftarrow$$

Fred receives a couple of messages:

$$From(Item19, Betty) \leftarrow$$
$$From(Item20, AuntDaisy) \leftarrow$$

Fred's agent concludes Betty's message is important ($Important(Item19)$) and that Daisy's message is not important ($\neg Important(Item20)$). After a while, Fred receives another message from Daisy:

$$From(Item115, AuntDaisy) \ \leftarrow$$
$$NotificationOf(Item115, Sit79) \ \leftarrow$$
$$SevereIllnessOf(Sit79, AuntDaisy) \ \leftarrow$$

(where the last two facts were extracted from the message's body by a natural language processing routine.) Fred's agent concludes Daisy's latest message is important ($Important(Item115)$), unlike her previous message, because it is notifies him of a personal emergency. □

5 Well-Behavior; Inferencing Algorithm

This section describes how courteous LP's behave well in several regards. We begin with discussion and examples, then proceed to theorems and an algorithm.

5.1 Discussion and Examples

A key to courteous LP's' computational and conceptual simplicity is that the **prioritization and conflict resolution is local, pairwise, and atomic**.

By local, we mean restricted to rules (rule instances, more precisely) within the same locale.

By pairwise, we mean that the only conflicts that need be considered are between two rule instances. In many other systems for default reasoning (e.g., circumscription and Default Logic and their prioritized versions), by contrast, conflict may involve $k > 2$ default instances together constituting a minimal conflict set (e.g., potentially any subset of the overall set of defaults); this is a source of exponential worst-case computational complexity.

By atomic, we mean that the essential focus of conflict is not even two entire rule instances, but rather only their heads, i.e., a pair of complementary ground literals competing about the truth assignment of the same ground atom.

Example 11 (Basics; Localization of Conflict; Chaining)

Next, we illustrate some basics of how courteous LP's behave, even with empty prioritization. The rule set

$$p \leftarrow \ , \ \neg p \leftarrow$$
$$q \ \leftarrow \ p \ , \ r \ \leftarrow \ \neg p$$
$$\neg u \ \leftarrow \ s \ \wedge \ \neg t \ \wedge \ \sim v \ , \ \neg t \leftarrow \ , \ s \leftarrow$$
$$w \ \leftarrow \ \neg u \ \wedge \ \sim \neg p$$

has answer set $\{s, \neg t, \neg u, w\}$. The unresolved conflict about p is localized in its impact: other conclusions are entailed (without *all* propositions being entailed as in extended LP's), including by some chaining. $\neg p$'s failure helps the last rule fire. (If formalized instead in Default Logic (as "normal" defaults), by contrast, there would be two "extensions": the first including p and thus q; the second including $\neg p$ and thus r.) $\Box$

Example 12 (Conflicting Chains; Pairwise; Non-Contrapositive) The following rule set illustrates the pairwise-ness of conflict resolution and prioritization.

$$\begin{array}{l} e \leftarrow b \;\; , \;\; b \leftarrow a \;\; , \;\; a \leftarrow \\ \neg e \leftarrow d \;\; , \;\; d \leftarrow c \;\; , \;\; c \leftarrow \\ f \leftarrow e \;\; , \;\; g \leftarrow \neg e \end{array}$$

Here, two chains of implication conflict with each other. The first chain consists of three rules that together "argue" for e. The second chain consists of three rules that together "argue" for $\neg e$. The resulting courteous LP answer set is $\{a, b, c, d\}$. The conflict about e is unresolved; neither e nor $\neg e$ is concluded, hence neither f nor g is.

Continuing this example: Suppose the rules in the first chain are given higher priority than those in the second chain. Then the resulting answer set is $\{a, b, c, d, e, f\}$. This includes e, now the winner in its locale, and f by chaining on e. This also still includes the earlier "steps" c and d of the other chain. That the rule for e wins does not contrapositively affect the status of d, for example. $\Box$

Terminology: By **contrapositive** here, we mean that when $\neg b$ is established somehow, the rule "if a then b" can be used to infer $\neg a$,

So far, our examples have been fairly simple: with few logical variables and relatively shallow chaining. In general, in courteous LP's: chaining may be through several prioritized locales, there may be predicates that have multiple logical variables as arguments, and logical functions may appear. The following rather abstract example illustrates.

Example 13 (Chaining Through Prioritized; Multiple Variables)

$\langle 1 \rangle$ $a(X, Z)) \leftarrow b(X, Y) \wedge c(Y, q(Z))$
$\langle 2 \rangle$ $\neg a(X, Y) \leftarrow d(X, Y)$
$\langle 3 \rangle$ $Overrides(1, 2) \leftarrow$
$\langle 4 \rangle$ $d(e1, e2) \leftarrow$
$\langle 4 \rangle$ $b(e1, e2) \leftarrow$
$\langle 4 \rangle$ $c(e2, q(e3)) \leftarrow$
$\langle 4 \rangle$ $b(e3, e7) \leftarrow$
$\langle 4 \rangle$ $c(e7, q(e8)) \leftarrow$
$\langle 5 \rangle$ $\neg d(e1, e2) \leftarrow$
$\langle 6 \rangle$ $Overrides(4, 5) \leftarrow$
$\langle 7 \rangle$ $f(X, Y) \leftarrow a(X, Y)$
$\langle 8 \rangle$ $g(r(X), e4) \leftarrow a(X, e4) \wedge j(e4)$
$\langle 9 \rangle$ $\neg f(X, Z) \leftarrow h(X, e5) \wedge \neg k(X, Z, e6) \wedge m(Z)$
$\langle 10 \rangle$ $h(e1, e5) \leftarrow$
$\langle 11 \rangle$ $\neg k(e1, e3, e6) \leftarrow$

$\langle 11\rangle \quad m(e3) \;\leftarrow$
$\langle 11\rangle \quad m(e8) \;\leftarrow$
$\langle 11\rangle \quad Overrides(9,7) \;\leftarrow$
$\langle 12\rangle \quad a(e2,e4) \;\leftarrow$
$\langle 13\rangle \quad j(e4) \;\leftarrow$

Here, X, Y, Z are logical variables; $e1, \ldots, e8$ are object constants (i.e., 0-ary logical function symbols); and q, r are logical function symbols.

The resulting answer set is:

$\{b(e1,e2), b(e3,e7), c(e2,q(e3)), c(e7,q(e8)), h(e1,e5), j(e4),$
$\neg k(e1,e3,e6), m(e3), m(e8), d(e1,e2), a(e2,e4), a(e1,e3), a(e3,e8),$
$f(e2,e4), \neg f(e1,e3), f(e3,e8), g(r(e2),e4)\}$

The chain of conclusions

$d(e1,e2), \;\; a(e1,e3), \;\; \neg f(e1,e3)$

is particularly interesting. In each of these three conclusions' (atom) locales, there is a conflict which is resolved (i.e., to produce a winner) by priority. □

5.2 Theorems and Algorithm

Theorem 14 (Consistency)

Every courteous LP has exactly one answer set which is consistent. □

Proof : Consider the answer set construction in Definition 5. Consider a given total (atom) stratification.

The answer set construction's definition directly implies there exists exactly one answer set.

Consistency is shown by an induction on the total stratification ρ (i.e., iteration $i = 0, 1, \ldots$) in the answer set construction. The inductive step is to show that each atom locale's incremental contribution T_i to the answer set iterate preserves consistency of the answer set iterate. By preserving consistency here, we mean that if S_{i-1} is consistent then $S_i = (S_{i-1} \cup T_i)$ is consistent. The base case of the induction is trivial: the empty answer set S_0 is consistent.

By definition of the construction, for atom p_i, p_i's locale's contribution T_i is either p_i or $\neg p_i$ or $\emptyset$. T_i is thus consistent in itself.

A simple way to see why S_i is consistent is the following. Each (ground) atom p_i is syntactically distinct from from each other previous p_j, for $j < i$. Each T_i if non-empty is thus a ground literal whose atom is distinct from the atoms appearing in each previous T_j, for $j < i$. S_i is, therefore, just an accumulation of ground literals whose atoms are distinct in this sense. And S_i contains at most one literal containing p_i. S_i is thus consistent.

More formally, we show next that S_i is a *conservative extension* of S_{i-1}. By conservative extension here, we mean in the usual sense cf. the classical-logic literature. We equivalently syntactically reformulate our representation to make the atom p_i be a primitive propositional (zero-argument) predicate. S_{i-1} can then be viewed as a formula of classical propositional logic: a conjunction of (ground) literals. By definition of the construction, the propositional-predicate symbol p_i does not syntactically appear as an atom in the formula S_{i-1}. I.e., p_i, and therefore T_i, are *new* with respect to S_{i-1}. Newness of the increment T_i implies the conservative extension property.

Because T_i is a consistent-in-itself, conservatively-extending increment to S_{i-1}, it must preserve consistency of the answer set iterate.
□

Next, we discuss how the semantics of courteous LP's relates to the semantics of extended LP's and general LP's.

Terminology: By an LP's **extended interpretation** (respectively, **courteous interpretation**), we mean the semantic interpretation of that LP as an extended LP (respectively, courteous LP). We say that an LP is **E-consistent** when its extended interpretation is consistent; intuitively, this corresponds to a situation in conflict is absent.

In the courteous LP representation, the labels and prioritization sub-program are permitted to be empty. Therefore, for any acyclic extended LP, one can treat it syntactically as a courteous LP and also one can interpret it semantically as a courteous LP. As we will see below, this results in no change to the semantics if that LP is E-consistent.

Conversely, for any courteous LP, one can treat it syntactically as an extended LP and also interpret it semantically as an extended LP: simply by ignoring the labels on the rules and the special role of the prioritization predicate *Overrides* in the courteous semantics. Terminology: We call this the *extended interpretation* of the courteous LP, i.e., of the **unlabelled** version of that courteous LP. Similarly, one can take the extended interpretation of any courteous LP's *main* part.

Next, we show that the courteous and extended semantics are essentially equivalent for the case of E-consistency. In the presence of conflict, by contrast, the courteous and extended semantics differ in that the **courteous interpretation of an extended LP is always consistent**.

Acyclic general LP's are essentially a special case of E-consistent acyclic extended LP's: they are conflict-free because each locale is one-sided; and, therefore, they are E-consistent.

Observe that the prioritization sub-program of a courteous LP is (syntactically) an acyclic general LP.

Theorem 15 (Agreement with Extended and General LP's)
Let $\mathcal{LP}$ be (syntactically) a courteous LP. Suppose it is E-consistent. Equivalently, suppose its main part ($\mathcal{LP}_{main}$) is E-consistent. Then:

1. $\mathcal{LP}$'s courteous interpretation and $\mathcal{LP}$'s extended interpretation are the same. That is, there is a single answer set under the extended interpretation, and it is the same as the answer set under the courteous interpretation.

2. The main part of $\mathcal{LP}$'s courteous interpretation is the same as the extended interpretation of its main part $\mathcal{LP}_{main}$. By main part of

the interpretation, we mean its restriction to the non-*Overrides* predicates.

A special case is whenever $\mathcal{LP}$ is (syntactically) an acyclic extended LP that is E-consistent, i.e., whenever $\mathcal{LP}$ lacks labels.

A further special case is whenever $\mathcal{LP}$, or its unlabelled version, is (syntactically) an acyclic *general* LP, i.e., whenever $\mathcal{LP}$ lacks classical negation. $\square$

Remark: Here, we interpret general LP's under the locally stratified semantics [1] [33], the stable semantics [10], or the well-founded semantics [37]. These semantics all coincide for the acyclic case since that is a special case of locally stratified (see, e.g., [2] for review of relevant concepts and literature). General LP's are syntactically a special case of extended LP's. Semantically, they are essentially a special case of extended LP's; the only difference is that in the usual definition, $LP \not\vdash atom$ in general LP's is treated also as $LP \vdash \neg atom$, i.e., the closed world assumption is applied to all atoms.

Proof :
Background: Extended LP's are defined by Gelfond & Lifschitz to have stable-style semantics. Stable semantics for the acyclic case coincides (see, e.g., [2] for review of literature) with the (locally) stratified semantics [33] and the well-founded semantics [37].

Overview and Introduction: If the main sub-program is consistent when interpreted as an extended LP, there is no conflict. The prioritization and labels are then irrelevant. The courteous semantics' stratified construction then essentially corresponds to the locally stratified semantics of [33]. For the acyclic case, the locally stratified semantics essentially coincides with [11]'s stable-style semantics for extended LP's.

Details:
Terminology: As usual throughout this paper, "atom" means ground atom.

We are given that $\mathcal{LP}$ is E-consistent. Let us take its unlabelled version and view it as an extended LP. By a result of [11], because it is E-consistent, it can be transformed equivalently into (re-represented as) a general LP. In that transform, $\neg p$ is simply made into (re-represented as) a new predicate p' distinct from p, for every pre-transform predicate p. Notationally, we say that each classically negated ground literal $\neg p_i$ is made into (re-represented as) p'_i.

Since the pre-transform LP had an acyclic atom dependency graph (ADG), the post-transform LP must also have an acyclic ADG (in the new representation). In particular, there can be no dependency of p'_i on p_i, nor vice versa, since that would correspond to a cycle from p_i to p_i in the pre-transform ADG. Therefore, the post-transform LP is locally stratifiable. Therefore, the post-transform LP has exactly one model under the locally-stratified semantics. Correspondingly, according to the transform's equivalence property, the pre-transform LP has a unique extended answer set. This model, and correspondingly this answer set, can be built via the locally stratified construction in the post-transform LP.

Because of the simple nature of the transform, the post-transform ADG is very similar to the pre-transform ADG. We can thus choose the local stratification of the post-transform LP as follows. Each stratum consists of a pair of post-transform atoms p_i, p'_i that correspond to (i.e., re-represent) a complementary pair of (ground) literals $p_i, \neg p_i$ for the same pre-transform atom p_i. We will call this stratum a "literal-pair" stratum. These strata (the i's) are sequenced (totally-ordered) according to a stratification ρ (a sequence of i's) that is chosen exactly as the sequence of i's was chosen in Definition 5: a reverse topological sort of the dependency graph in which the $Overrides$ locales precede all the other locales. We will call this stratification of the literal-pair strata a "total" stratification, since it totally orders the i's (though it does not totally order the post-transform *atoms*).

The i^{th} literal-pair stratum and the total-stratification ordering of the i's, in the post-transform LP, thus correspond respectively to the i^{th} atom locale stratum and the same total-stratification ordering, in the courteous LP answer set's definitional construction in Definition 5.

By an argument similar to that we used in the Proof of Theorem 14, the given E-consistency implies that each literal-pair stratum is making a contribution of one or zero literals, not both its literals, to the extended answer set. In other words, it contributes at most one side (p_i XOR $\neg p_i$) of the atom p_i, not both sides (p_i AND $\neg p_i$). This corresponds in the courteous answer set construction to the situation in which there is no conflict (which is defined precisely as: at most one side having candidates in p_i's locale).

In the courteous answer set construction, the result in each locale is thus independent of (unaffected by) the prioritization and labels.

Therefore, the courteous answer set is the same as the extended answer set. And because the main part of the program is unaffected by the prioritization, the main part of the courteous interpretation is thus the same as the courteous (or extended) interpretation of the main part of the program.

It remains to show that $\mathcal{LP}_{main}$'s E-consistency implies $\mathcal{LP}$'s E-consistency, and vice versa. Again, we reason about the stratified semantic construction for the extended LP (via its transform to a general LP). We choose the prioritization sub-program (i.e., the $Overrides$ atoms) now instead to be a set of *shallowest* strata in the extended interpretation's locally-stratified construction. [3] The prioritization sub-program (i.e., the $Overrides$ atoms' locales) is itself consistent (since one-sided), and logically orthogonal to (neither affects nor is affected by) the main part of the interpretation. $\mathcal{LP}_{main}$ corresponds to all the previous strata. E-consistency is simply the

[3]Recall that in the total stratification cf. Definition 5, the $Overrides$ atoms' locales were a set of *deepest* strata.

existence of a consistent answer set. It is easy to see then that both directions of the E-consistency equivalence follow. □

Next, we discuss inferencing and its computational complexity.

We begin by reviewing a few facts about how instantiation affects size. Let n stand for the size of $\mathcal{C}$. Let m stand for the size of $\mathcal{C}^{instd}$. An interesting question is how much larger m is than n. If $\mathcal{C}$ has no free variables, e.g., is propositional, then $m = n$. A common restriction in practice for logic programs is the **Datalog** condition, i.e., that there are no function symbols with arity greater than 0. Another common restriction in practice for logic programs is that there is a finite upper bound v on the number of variables appearing in any one rule. Taken together, the Datalog and variables-bound conditions on $\mathcal{C}$ imply that the size of the Herbrand base is $O(n)$, and that m is $O(n^{v+1})$.

Theorem 16 (Tractability of Inferencing; Algorithm)

Suppose $\mathcal{C}^{instd}$ is finite. Let its size be m. Then $\mathcal{C}$'s entire answer set can be computed in time $O(m^2)$ [4] .

As a special case, suppose $\mathcal{C}$ obeys the Datalog restriction and has a bounded number v of variables per rule. Let $\mathcal{C}$'s size be n. Then $\mathcal{C}$'s entire answer set can be computed in time $O(n^{2\cdot(v+1)})$. □

Proof : A conceptually simple **algorithm** (for exhaustive forward inferencing) is to implement directly the answer set construction that we gave in defining courteous programs. Next, we sketch this algorithm enough to analyze its computational complexity.

(In the following complexity analysis, as in the theorem statement, we ignore log factors, e.g., in sorting or for accessing an element in a list.)

Instantiation: Scan $\mathcal{C}$ to generate the Herbrand base. Then syntactically expand $\mathcal{C}$ to form $\mathcal{C}^{instd}$. This can be done in time $O(m)$.

Total Stratification: The size (edges plus vertices) of the atom dependency graph is $O(m)$; it can be built essentially by scanning $\mathcal{C}^{instd}$, in time $O(m)$. Topological sort can be done in time linear in the size of the graph, by doing a depth-first traversal (e.g., [8], pp. 485–487).

Atom locales: Consider an atom locale. Let d be its size. Generating candidates can be computed simply because satisfaction (e.g., of rule bodies) in the previous answer set iterate can be tested quickly, by lookup operations. The overall cost to generate these candidates, classified by sign, is thus $O(d)$. There are at most $O(d^2)$ pairs of opposing candidates to consider in the refutation process. For each pair, computing refuted-ness requires simply a lookup of the corresponding prioritization atom. Applying the skepticism principle requires a single XOR operation. The overall cost of inferencing per atom locale is thus $O(d^2)$.

There are $O(m)$ atom locales; however, the more lcales there are, the smaller they have to be, since the total size of all the locales together is

[4] ignoring log factors, e.g., for insertion or retrieval of an element in a list

$O(m)$. The overall effort bound of $O(m^2)$ is obtained by considering the situation where the locales are large. □

Discussion: The core of the extra computational cost, relative to (acyclic) general LP's or consistent extended LP's, is the refutation process. One must consider refutation for each ordered *pair* of opposing candidates in each locale; this results in quadratic complexity within the locale. For conflict-free programs, this overhead is absent, and the overall cost is $O(m)$. The above algorithm imposes non-constant-factor **overhead only when there actually is conflict to deal with**. In the general case, this overhead is an additional factor of $O(m)$, making the overall cost $O(m^2)$. However, often the overhead factor is lower than $O(m)$: e.g., if in each locale the number of conflicting rules is small (relative to m). More precisely, this observation and the proof above imply the following tighter complexity bound.

Corollary 17 (Overhead Factor from Conflict Handling)
In Theorem 16:
Suppose the size of the smaller of the two teams of candidates in each (atom) locale is bounded by $O(f(m))$, where $f(m)$ is smaller than m (e.g., $f(m) = m^\alpha$ where $0 \leq \alpha < 1$). Then the overhead factor is $O(f(m))$, i.e., overall cost is $O(m \cdot f(m))$. □
Proof : In the proof of Theorem 16:
In each locale, the number of pairs of opposing candidates, and thus the refutation cost, is bounded by $O(d \cdot f(m))$. The total size of all the locales together, i.e., the sum of all the d's, is $O(m)$. Therefore, the total refutation cost, summed across all locales, is $O(m \cdot f(m))$. The other costs besides refutation cost total $O(m)$. □

6 Simple Updating cf. Databases; Closed World Assumption

6.1 Database-Flavor Updating; Recency Priority

A common principle in databases and deductive databases is that more recent "update" information overrides less recent information. The update may overturn a previous conclusion directly, or may imply indirectly that it should be overturned. This is simple to represent in courteous LP's: by adding the more recent rules (e.g., facts), and prioritizing them higher than the previous rules. This has several virtues compared to general or extended LP's. Negation-as-failure is not required. Moreover, the previous rules do not have to be modified at all — the update is modular in that sense. As a bonus, a group of rules (e.g., a "module") can share a label, thereby reducing the number of prioritization facts needed to specify the overriding preference for recency.

Example 18 (Directly Overturning Previous Conclusion)

E.g., suppose the previous rule set is:

$\langle prev1 \rangle \quad p \;\leftarrow$

At this point, p holds as a conclusion. Then updating with (i.e., adding) the new, more recent fact

$\langle upd1 \rangle \; \neg p \;\leftarrow$

$Overrides(upd1, prev1) \;\leftarrow$

results in concluding $\neg p$.

Similarly, the previous conclusion might have been derived from chaining. E.g.,

$\langle prev2 \rangle \quad q \;\leftarrow$

$\langle prev2 \rangle \quad p \;\leftarrow\; q$

updated by

$\langle upd2 \rangle \; \neg p \;\leftarrow$

$Overrides(upd2, prev2) \;\leftarrow$

results in concluding $\neg p$. □

Example 19 (Indirectly Overturning Derived Previous Concl.)

Yet more generally, the new overriding conclusion might also have been derived by chaining from the update, rather than directly appearing in the update. E.g., from the previous

$\langle prev3 \rangle \quad r \;\leftarrow$

$\langle prev3 \rangle \quad s \;\leftarrow$

$\langle prev3 \rangle \quad t \;\leftarrow$

$\langle prev3 \rangle \quad q \;\leftarrow\; r \wedge s$

$\langle prev3 \rangle \quad p \;\leftarrow\; q \wedge t$

p is a conclusion. Updating by

$\langle upd3 \rangle \; \neg p \;\leftarrow\; u$

$\langle upd3 \rangle \quad u \;\leftarrow$

$Overrides(upd3, prev3) \;\leftarrow$

results in concluding $\neg p$ instead, as well as u.

6.2 Forcing; Cumulativity

Theorem 20 (Forcing a Conclusion)

Courteous LP's have a relatively simple way to **force a conclusion** q: simply include a fact $q\leftarrow$, at (strictly-)highest priority within its locale. □

Terminology: By **strictly-highest priority**, we mean having priority higher than any other rule in its locale.

Proof : Consider the answer set construction. A strictly-highest-priority candidate refutes every opposing candidate in its locale, hence its side wins. □

Related to this point is the following property.

Theorem 21 (Cumulativity)

Courteous LP's are **cumulative** in the sense of [28] [25]: if p is a conclusion of $\mathcal{C}$, then adding the fact $p \leftarrow$ (with any priority within its locale) to $\mathcal{C}$ results in a $\mathcal{C}'$ that has the same answer set as $\mathcal{C}$. $\Box$

Proof : Consider the answer set construction, and a given total atom stratification sequence ρ. Compare the change as one goes from $\mathcal{C}$ before to $\mathcal{C}'$ after.

For every locale i that precedes p's locale in ρ, there can be no change in that locale's set of rules, its set of candidates, or its contribution T_i to the answer set.

In p's locale, the new fact simply adds a new candidate for p in p's locale. p's side's team was already winning even before adding this new candidate — every opposing candidate was refuted — and the new candidate does not alter this. Thus p's locale is unchanged in its contribution T_i.

Consider the next locale following p's locale in ρ. This locale's rules are unchanged. All of its candidates (i.e., the truth of its rules' bodies) are unchanged because all of the previous contributions are unchanged. Hence its contribution is unchanged. By an inductive argument, ditto for every later locale. $\Box$

6.3 Closed World Assumption

A common representational device adopted in databases and many other settings is the Closed World Assumption (CWA).

The Closed World Assumption for any given literal (e.g., for a given predicate), can be represented in a simple fashion using priority plus classical negation. To achieve the effect of minimizing the predicate p after all the other rules "have had their say", include the rule $\neg p(x)$ with lowest priority within its locale.

By contrast, [11] give an approach in extended LP's — they represent the CWA for p by the rule $\neg p(x) \leftarrow \sim p(x)$. Observe that this employs negation-as-failure, and also creates a cycle in the dependency graph.

Example 22 (CWA: Airline Flights)

A typical kind of relational database over which CWA is adopted is for airline flights. Each flight is represented as a positive ground literal in the predicate $flight(source, destination, time, airline)$. Implicitly, there are definitely no flights other than the explicitly asserted ones. Let $\mathcal{C}$ be a courteous LP whose $flight$ locale is:

$\langle scheduled \rangle \quad flight(Miami, Detroit, 10am, Elysian\_Air) \ \leftarrow$
$\langle scheduled \rangle \quad flight(JFK, New\_Orleans, 4pm, Fountain\_Air) \ \leftarrow$
$\langle scheduled \rangle \quad flight(Dallas, Seattle, 7pm, Middle\_Air) \ \leftarrow$
$\langle CWA\_flight \rangle \ \neg flight(s, d, t, a) \ \leftarrow$
$\quad Overrides(scheduled, CWA\_flight) \ \leftarrow$

(Here, s, d, t, a are variables.)

The resulting answer set consists of three positive $flight(\ldots)$ ground conclusions

$flight(Miami, Detroit, 10am, Elysian\_Air)$
$flight(JFK, New\_Orleans, 4pm, Fountain\_Air)$
$flight(Dallas, Seattle, 7pm, Middle\_Air)$

plus a bunch of negative $\neg flight(\ldots)$ ground conclusions, e.g.,

$\neg flight(New\_Orleans, Dallas, 4pm, Middle\_Air)$ □

7 Merging Rule Set Modules; Advice Taking; Teams

Courteous programs have desirable properties with regard to **merging** of rule sets, i.e., of (sub-)programs. Such merging may be a simple union, or more generally it may involve prioritization between the rule sets being merged, i.e., the merge may be prioritized.

7.1 Parallel Merges

Consider the program $\mathcal{C}_{1\&2}$ formed by taking the union of two programs $\mathcal{C}_1$ and $\mathcal{C}_2$. We **define** this as **merging** them **in parallel**, i.e., without adding prioritization. We view each of the programs as **modules**.

To be fully precise, in order to ensure that concept of merging is well-defined, there is a subtlety. In this paper, for simplicity's sake in defining merging (both parallel and prioritized), we **require** as a condition that, $\mathcal{C}_1$ does not contain prioritization facts mentioning labels from $\mathcal{C}_2$, nor vice versa. In other words, before merging two modules, there is no inter-module (strict) prioritization.

Theorem 23 (Preservation in Parallel Merging)

Suppose p is a conclusion in both $\mathcal{C}_1$ and $\mathcal{C}_2$. Consider their parallel merge $\mathcal{C}_{1\&2}$. Suppose that for each rule in p's locale, the truth of its *body* is unchanged after the merge. Then p is conclusion of $\mathcal{C}_{1\&2}$. □

Proof : In p's locale, the set of candidate arguments in $\mathcal{C}_{1\&2}$ is simply the union of the set of candidate arguments in $\mathcal{C}_1$ and the set of candidate arguments in $\mathcal{C}_2$. This is because the truth of the rule bodies was unaffected.

Terminology: By **refutation pair**, we mean an ordered pair of candidate arguments where the first refutes the second.

The set of refutation pairs in $\mathcal{C}_{1\&2}$ is, likewise, the union of the set of refutation pairs for $\mathcal{C}_1$ with the set of refutation pairs for $\mathcal{C}_2$. This is because there is no new strict prioritization after the merge.

The set of unrefuted arguments in $\mathcal{C}_{1\&2}$ is thus the union of the set of unrefuted arguments before the merge in $\mathcal{C}_1$ with the set of unrefuted arguments in $\mathcal{C}_2$.

We are given that p is a conclusion in both $\mathcal{C}_1$ and $\mathcal{C}_2$. In both $\mathcal{C}_1$ and $\mathcal{C}_2$, therefore, in p's locale, the set of unrefuted arguments is non-empty and all for p; none of the unrefuted arguments is for $\neg p$. The union of these sets, i.e., the set of unrefuted arguments in $\mathcal{C}_{1\&2}$, is thus non-empty and all for p. But this is just equivalent to p being a conclusion in $\mathcal{C}_{1\&2}$. $\Box$

Example 24 (Preservation in Parallel Merging)
Let $\mathcal{C}_1$ consist of

$$
\begin{array}{lll}
\langle 1 \rangle & p & \leftarrow \; a \\
 & a & \leftarrow \\
\langle 2 \rangle & \neg p & \leftarrow \; b \\
 & b & \leftarrow \\
 & \multicolumn{2}{l}{Overrides(1,2) \;\; \leftarrow}
\end{array}
$$

$\mathcal{C}_1$'s answer set includes p: there is conflict about p between rules 1 and 2, but it is resolved by the priority in 1's, and thus p's, favor.

Let $\mathcal{C}_2$ consist of

$$
\begin{array}{lll}
\langle 3 \rangle & \neg p & \leftarrow \; c \\
 & c & \leftarrow \\
\langle 4 \rangle & p & \leftarrow \; d \\
 & d & \leftarrow \\
 & \multicolumn{2}{l}{Overrides(4,3) \;\; \leftarrow}
\end{array}
$$

$\mathcal{C}_2$'s answer set includes p: there is conflict about p between rules 3 and 4, but it is resolved by the priority in 4's, and thus p's, favor.

Intuitively, we expect that p should also be a conclusion of the merged program $\mathcal{C}_{1\&2}$. All of the rules in p's locale still "fire", just as before the merge: i.e., *their bodies remain true*, since there is no conflict involving them. And indeed, p is a conclusion of $\mathcal{C}_{1\&2}$.

However, there is a subtlety. No one rule / candidate refutes all the rules / candidates that oppose it. There is no strict priority between rules 1 and 3, nor between rules 4 and 2. Nevertheless, the *team* of rules 1 and 4 is able to refute all their opposers. This kind of situation, in part, motivates the team aspect of our definition of prioritization's semantics — we will return to this point in subsection 7.3. $\Box$

7.2 Prioritized Merges

Let us **define** the **prioritized merge** $\mathcal{C}_{1>2}$ of two modules (programs) $\mathcal{C}_1$ and $\mathcal{C}_2$ as the set of rules formed by first taking their union, and then adding prioritization facts representing that each rule in $\mathcal{C}_1$ is higher priority than each rule in $\mathcal{C}_2$. There is a subtlety in ensuring this is well-defined. It may be that, before merging, some of the rules have empty labels; for purposes of defining the merging priority, an explicit label is assigned to these rules, essentially as in our earlier discussion of *empty_label*. (Also, recall that prioritization between the prioritization facts themselves has no effect.)

More generally, it is straightforward to **define merging many modules** (not just two) with an **inter-module prioritization** partial order, using the concept of **composing prioritization** given in [14] [15] [17].

In intelligent agent applications such as those we are pursuing at IBM, modules may correspond to different users'/agents' knowledge bases; merging then corresponds to **taking advice** from other agents [17]. E.g., Joe's workflow rule set may be formed by combining rules he writes (and maintains) himself with rules written (and maintained) by Sue, his supervisor. E.g., Joe may also merge in a news-handling rule module from his buddy Angela, and merge in rule modules on different subjects from various "enterprise" rule sets associated with different organizational levels or units.

There are many natural bases for inter-module prioritization information [14] [17]. E.g., Joe may merge in a newer (fresher) module, or a more special-case (specificity) module. Relative authority (e.g., organizational) or reliability (e.g., expertise) of modules' sources may be the basis for prioritizing between the merges.

Theorem 25 (Preservation in Prioritized Merging)

Suppose p is a conclusion of $\mathcal{C}_1$. Then p is a conclusion of $\mathcal{C}_{1>2}$. $\Box$

Proof : Consider the answer set construction (according to Definition 5) of $\mathcal{C}_{1>2}$. The proof is by induction along the stratification of the locales. The inductive hypothesis is that the theorem property holds for the (partial) answer set iterate up through locale i, i.e., for S_i. In each locale, after the merge, the candidates from $\mathcal{C}_1$'s rules are still candidates — i.e., their bodies are true, because of the inductive hypothesis — and refute any opposer candidates from $\mathcal{C}_2$'s rules (because of the merge's prioritization). In the base case of the induction, the bodies are true because they are empty — recall that the first locale in the stratification is chosen to be for an *Overrides* atom. $\Box$

Example 26 (Preservation in Prioritized Merge)

Let $\mathcal{C}_1$ consist of

$$\begin{array}{rcl} p & \leftarrow & a \\ a & \leftarrow & \\ q & \leftarrow & b \wedge c \\ b & \leftarrow & \\ c & \leftarrow & \end{array}$$

$\mathcal{C}_1$'s answer set is $\{p, q, a, b, c\}$:

Let $\mathcal{C}_2$ consist of

$$\begin{array}{rcl} \neg p & \leftarrow & d \\ d & \leftarrow & \\ \neg a & \leftarrow & \\ \neg q & \leftarrow & b \end{array}$$

$\mathcal{C}_2$'s answer set is $\{\neg p, \neg a, d\}$. Notice that it has the opposite conclusion about a and about p than $\mathcal{C}_1$ does.

Then $\mathcal{C}_{1>2}$'s answer set is $\{p, q, a, b, c, d\}$. Notice that in a's locale, the rule from $\mathcal{C}_1$ wins over the conflicting rule from $\mathcal{C}_2$. Thus the body of $\mathcal{C}_1$'s rule about p was satisfied. (This illustrates the inductive aspect mentioned in the Proof of Theorem 25.) The body of $\mathcal{C}_2$'s rule about p was also satisfied. In p's locale, the candidate from $\mathcal{C}_1$ wins over the conflicting rule from $\mathcal{C}_2$; in q's locale, likewise. □

7.3 Teams in Refutation, and Merging

Behavior under merging is a major desideratum motivating our choice of how to define prioritization, especially the team aspect of refutation.

A "naive" **alternative approach to defining refutation** is the following, which we dub "**top dog**" or "single combat":

p wins iff there is a candidate for p that
refutes all opposing candidates.

However, this alternative approach is problematic, as the next example illustrates.

Example 27 (Simple Merge of 2 Cloned Modules)

Consider the following parallel merge of a simple module with a clone of itself.

$\langle 1a \rangle \quad p \ \leftarrow$
$\langle 1b \rangle \ \neg p \ \leftarrow$
$\langle 1c \rangle \quad Overrides(1a, 1b) \ \leftarrow$
$\quad Module1 \mid\!\sim p$

$\langle 2a \rangle \quad p \ \leftarrow$
$\langle 2b \rangle \ \neg p \ \leftarrow$
$\langle 2c \rangle \quad Overrides(2a, 2b) \ \leftarrow$
$\quad Module2 \mid\!\sim p$

(Here, $Module1$ refers to rules $\{1a, 1b, 1c\}$, and $Module2$ refers to rules $\{2a, 2b, 2c\}$. Recall the $\mid\!\sim$ notation from after Definition 5.)

With our definition of courteous LP's, $(Module1 \cup Module2) \mid\!\sim p$. We find this behavior intuitively desirable.

However, if instead the "top-dog" definition is adopted, then the merge does not preserve p as a conclusion:

$(Module1 \cup Module2) \mid\!\not\sim p$!!

The difficulty is that rule $(1a)$ does not override rule $(2b)$, and rule $(2a)$ does not override rule $(1b)$; neither rule $(1a)$ nor rule $(2a)$ is a "top dog" able single-handedly to refute all opposers. □

The above example was extremely simple. However, *this kind of behavior and issue motivating our team definition arises generally, e.g., for inferences via chaining, for merging with non-clones, for prioritized merging, and for* *rule sets not created by merging.* Example 24 illustrates this point, as does the next, slightly more complicated version of the example above.

Example 28 (Merge of 2 Cloned Modules, with Chaining)

$\langle Mod4\_R1\rangle \quad q \leftarrow r$
$\langle Mod4\_R2\rangle \quad r \leftarrow$
$\langle Mod4\_R3\rangle \quad s \leftarrow$
$\langle Mod4\_R5\rangle \quad p \leftarrow q \wedge r$
$\langle Mod4\_R8\rangle \quad \neg p \leftarrow s$
$\langle Mod4\_R9\rangle \quad Overrides(Mod4\_R5, Mod4\_R8) \leftarrow$
$Module4 \mid\!\sim \{p, \;\; q, r, s\}$

$\langle Mod7\_R1\rangle \quad q \leftarrow r$
$\langle Mod7\_R2\rangle \quad r \leftarrow$
$\langle Mod7\_R3\rangle \quad s \leftarrow$
$\langle Mod7\_R5\rangle \quad p \leftarrow q \wedge r$
$\langle Mod7\_R8\rangle \quad \neg p \leftarrow s$
$\langle Mod7\_R9\rangle \quad Overrides(Mod7\_R5, Mod7\_R8) \leftarrow$
$Module7 \mid\!\sim \{p, \;\; q, r, s\}$

We desire, as in courteous, that
$(Module4 \cup Module7) \mid\!\sim \{p, \;\; q, r, s\}$
But under the alternative "top-dog" definition:
$(Module4 \cup Module7) \not\mid\!\sim p$!! $\square$

Summary: The *team* aspect of refutation in courteous LP's handles well a **subtlety in prioritization**, and **facilitates merging** (prioritized or unprioritized), e.g., "advice-taking" between agents.

8 More Analytic Properties

NB: This section can be skipped by the reader less interested in comparisons with the non-monotonic reasoning literature.

In this section, we give some analysis of properties of the courteous LP formalism, that are especially of interest for purposes of comparison with other approaches in the non-monotonic reasoning literature.

8.1 Local Monotonicity of Prioritization

Theorem 29 (Prioritization is Locally Monotonic)

Prioritization is locally monotonic in the sense of the following.

I. Within an *atom* locale, adding prioritization preserves the previous conclusion if there was one, and may result in an additional conclusion.

II. Suppose the *predicate* dependency graph is acyclic. In other words, suppose that for each predicate p, no instance of p depends on any other instance of p. Then within a *predicate* locale, adding prioritization preserves all the previous conclusions, and may result in additional conclusions.

By adding prioritization, we mean adding more prioritization facts about the labels of the rules in that locale (and only in that locale).

Proof : (I.): There are four possibilities for the atom locale a previous to adding the prioritization: (1) no candidates, (2) a winning side (i.e., sign of a) with no opposing candidates, (3) a winning side with every opposing candidate being refuted, or (4) mutual defeat (unrefuted candidates for both sides). Adding prioritization does not affect which rules' bodies are satisfied, and thus does not change the set of candidates; it only changes the refutations. Thus (1) and (2) are not affected by adding prioritization. In case (3), all of the refuted opposing candidates are still refuted, hence the previous winning side still wins. In case (4), it may be that all of one side becomes refuted, creating a winner.

(II.): We can view p's predicate locale as consisting of the set of atom locales, one per instance of p. Adding prioritization to p's predicate locale is thus equivalent to adding prioritization to these instance atom locales. Let atoms a and b be distinct instances of predicate p. The given acyclicity of the predicate dependency graph implies that b does not depend on a. Therefore, adding prioritization to a's locale (i.e., possibly adding a conclusion to a's locale) does not affect which rules fire in b's locale, i.e., do not affect the set of candidates in b's locale. b's locale prioritization is also unaffected by adding prioritization to a's locale. Thus the b atom locale behaves as in (I.). This argument holds for all of the instances of p, not just b in particular. $\Box$

More prioritization may resolve conflict in the sense of creating winners where before there was mutual defeat. The Pacifism example (Example 7) illustrated this local monotonicity.

The local monotonicity of prioritization **aids incremental development**: it makes it easier for a rule-base author, i.e., courteous logic programmer, to resolve conflicts and to anticipate the effects of program modifications.

Terminology: By the global monotonicity of prioritization, we mean that adding prioritization facts never causes retraction of conclusions from the (overall, entire) answer set.

In courteous LP's, prioritization is **not globally monotonic** (rather, it is only *locally* monotonic). This contrasts with some other prioritized default formalisms (e.g., prioritized circumscription or Prioritized Default Logic).

The next example illustrates.

Example 30 (Prioritization Not Globally Monotonic)

The rule set

$\langle 1 \rangle \quad a \leftarrow$

$\langle 2 \rangle \ \neg a \leftarrow$

$\quad\quad b \leftarrow a$

$\langle 3 \rangle \quad c \ \leftarrow$
$\langle 4 \rangle \quad \neg c \ \leftarrow \ b$
$\quad Overrides(4,3) \ \leftarrow$
has unresolved conflict in the a locale, and thus neither a nor b is concluded. This results in c as conclusion.

Consider the effect on the answer set of adding
$\quad Overrides(1,2)$
a is concluded (the update is locally monotonic within the a locale). b is thus concluded. Therefore, $\neg c$ is concluded, reversing and retracting the previous conclusion c. The c locale's conclusion thus changes non-monotonically.

Analysis: Here, the newly resolved conflict in a deeper locale results in a new candidate firing in the shallower locale, with non-monotonic effect in that shallower locale. □

8.2 No Propagation of Ambiguity

Courteous LP's **do not propagate ambiguity** in the sense of [36]'s paper on "clash of intuitions" in default reasoning.

Example 31 (No Propagation of Ambiguity)

E.g., consider the LP formed by adding the following rules to the unprioritized Pacifism example (Example 7):
$$FootballFan(x) \ \leftarrow \ Republican(x)$$
$$AntiMilitary(x) \ \leftarrow \ Pacifist(x)$$
$$\neg AntiMilitary(x) \ \leftarrow \ FootballFan(x)$$
Then $\neg AntiMilitary(Nixon)$ is concluded, rather than not being concluded. This can be viewed as follows. The Quaker and Republican rules conflict about Nixon's Pacifism, and the conflict is handled skeptically in a local fashion, resulting in a single partial "extension" (in the Default Logic sense of "extension"), i.e., a single answer set iterate, which contains neither $Pacifist(Nixon)$ nor $\neg Pacifist(Nixon)$. In the AntiMilitary locale, the FootballFan rule thus "fires" while the Pacifist rule does not.

Terminology: By contrast, Touretzky & *et al* call the following behavior: **"propagating ambiguity"**. Some default reasoning formalisms spawn multiple extensions in this example: one extension with $Pacifist(Nixon)$ and $AntiMilitary(Nixon)$, a second extension with $Pacifist(Nixon)$ and $\neg AntiMilitary(Nixon)$, and a third extension with $\neg Pacifist(Nixon)$ and $\neg AntiMilitary(Nixon)$. Observe that neither sign (side) of $AntiMilitary(Nixon)$ nor of $Pacifist(Nixon)$ is present in the intersection of all these multiple extensions. Terminology: We can call this intersecting (i.e., disjoining) after extension-spawning: **"global" skepticism**. By contrast, the courteous semantics has **"local" skepticism**.

9 Discussion: Related Work

9.1 Courteous Can Avoid Negation-As-Failure

To achieve somewhat similar effects to defaults or priorities, previous logic programming work has often resorted to the non-monotonic mechanism of negation-as-failure, which provides a "hook" for blocking a rule.

In courteous LP's, **the prioritization mechanism reduces the need for the representational use of negation-as-failure**. (Recall especially our discussions of specificity-based override and of the Closed World Assumption.) This lessened need is, in part, significant because negation-as-failure causes many of the difficulties in the semantics of logic programs, both computationally and conceptually / intuitively (especially for non-technical authors faced with classical negation as well).

9.2 Previous Logic Programming Approaches

Brewka [4] has a form of prioritized extended LP's that is closely related to courteous LP's. Courteous LP's (first introduced in [19] whose content partially overlaps with this paper) and Brewka's approach were developed independently. Brewka's syntax is similar to courteous LP's; however, he is concerned with two additional directions of expressive generality: cyclic dependencies, especially negative (through $\sim$); and reasoning about the prioritization itself, e.g., cf. [16] [3] [18] [17]. He modifies the well-founded semantics to strengthen its conclusions; prioritization specifies how rules defeat each other via negation-as-failure. For comparison, each courteous LP rule r is most appropriately treated as "seminormal", i.e., as including additional body literal $\sim head(r)$. His semantics, including for priorities, behaves differently from courteous LP's, in general, even for some expressively simple cases. Worth considering is Example 27 which in courteous results in conclusion p but in Brewka's does not [5]. His formalism, like courteous, has the great virtue of computational tractability; conclusions can be computed in $O(n^3)$ time, as compared to $O(n^2)$ for courteous. Unlike courteous, his approach can result in an inconsistent conclusion set; he gives no consistency guarantee even for restricted cases.

Next, we summarize Brewka's [4] review of three other relevant approaches. Praaken & Sartor [32] has an argument-theoretic approach very similar to Brewka's. Buccafurri, Leone, & Rullo's [6] ordered LP's use an inheritance hierarchy as the basis of priority, but allow only one form of negation. Kowalski & Sadri [24] has a conflict handling approach in which rules that are "exceptions" to others are treated as having implicit preference.

Baral & Gelfond [2] discuss techniques for achieving some of the effects of priorities within extended LP's, e.g., via abnormalities, but do not give a general method. Pereira *et al* [31] address the issues of consistency, and

[5] personal communication with Brewka

priorities between abnormalities, in extended LP's. Their approach to modifying the semantics is quite different in spirit and detail from ours, e.g., they force in $\sim$ literals contrapositively. An interesting issue for future work is to investigate the relationship between the courteous approach and the representational methods based on abnormalities given by Baral & Gelfond and Pereira *et al.*

Hierarchical Constraint Logic Programming (HCLP) [38] has a notion somewhat similar to defaults with priorities: soft constraints, which are used in an abductive-flavor manner. In details of concept, inferencing, and application, HCLP differs markedly from courteous logic programs.

Contemporaneously and independently of this paper, the following other related work appeared, too late to provide detailed comparisons here. Brewka [5] and Zhang & Foo [39] each give an approach to prioritizing extended LP's. Gelfond & Son [12] give an approach to specifing prioritized defaults in extended LP's. Leite & Pereira [26] gives an approach to updating LP's that can achieve some similar effects as prioritizing LP's.

9.3 Previous Other Non-Monotonic Reasoning Approaches

As our examples illustrated, courteous LP's have more expressive power on several dimensions than default inheritance systems in the vein of Touretzky [35]: e.g., multiple body conditions, multiple variables, prioritization beyond specificity, and appearance of negation (including by failure) in the body. An interesting issue that we are pursuing in current work is how to automatically infer implicitly the higher priority of more specific rules.

Prioritized default circumscription [14] [15] (generalizing [30] [27]) has a concept of prioritization that is similar to courteous', though more general. However, circumscription behaves contrapositively, and thus quite differently from logic programs.

Extended LP's are equivalent to a special case of Default Logic [34], as Gelfond & Lifschitz showed. Prioritized variants of Default Logic (see [3], which includes a review of that literature) are thus a target for comparison. Courteous LP's' semantics differ significantly from that of these Prioritized Default Logic (PDL) variants: the latter proliferate *multiple extensions* (of which there are exponentially many, in the worst case). In logic programming terms, each extension can be viewed as an alternative candidate set of conclusions, i.e., an alternative answer set. This multiplicity of extensions derives from PDL's heritage: Default Logic.

E.g., in the unprioritized Nixon diamond ($\mathcal{C}_1$ in Example 7), two extensions are spawned: one with $Pacifist(Nixon)$ and one with $\neg Pacifist(Nixon)$. In the first extension, $Pacifist(Nixon)$ is then available for further rules to "use" in the sense of satisfying their antecedents (bodies in LP terminology); while in the second extension, $\neg Pacifist(Nixon)$ is available to do likewise. In effect, the reasoning across the multiple extensions must "carry along" the size-2 disjunction of possibilities about Nixon's Pacifism. Informally: the presence of conflict, especially conflict unresolved by priority, can generate such a disjunctive split; k such unresolved conflicts may result in k such disjunctive splits, and hence 2^k extensions.

Another way to put this is that the PDL approaches *propagate ambiguity* in the sense of [36]'s paper on "clash of intuitions" in default reasoning (recall the discussion in subsection 8.2).

Courteous LP's are, roughly speaking, less expressive than PDL or prioritized circumscription; there, arbitrary formula expressions, including quantifiers, are permitted in the consequent or antecedent of a rule.

However, courteous LP's are much less complex computationally and conceptually than those relatively expressively powerful formalisms.

Inference in courteous logic programs is tractable (polynomial-time); unlike prioritized Default Logic and circumscription and their ilk, courteous programs **avoid having a second source of NP-hard complexity beyond the underlying monotonic-logical inferencing** [13]. The locality of skepticism and prioritization is crucial in this regard: it nips in the bud the potential branching into multiple extensions. Together with the (inter-locale) stratification, this locality ensures a single answer set.

9.4 Alternative Variants of Courteous LP's

Alternative variants of courteous logic programs can be defined which **do propagate ambiguity and proliferate multiple extensions**. One variant can be defined by using Brewka's style of definition in which all sequences of rules are attempted that are compatible with the prioritization information (including not only the explicit prioritization but also the stratification). Another variant can be defined by branching into two extensions, one with p and one with $\neg p$, when a locale has unrefuted candidate(s) for each. A major practical difficulty with each of these variants is that there are, in the worst case, an exponential number of extensions to consider.

9.5 Simplicity of Courteous LP Approach

Part of courteous LP's' simplicity derives from the logic programming approach: e.g., definiteness (no "carrying around" or "propagating" of disjunctions or "ambiguity") and no contrapositivity; in particular, these properties keep the underlying monotonic-logical inferencing tractable.

Another key to courteous LP's' computational simplicity is that the prioritization and conflict resolution is local, pairwise, and atomic (recall the discussion near the beginning of Section 5).

The value of the courteous approach to logic programming, in our view, is in **finding a definition** such that the resulting formalism is simple conceptually and computationally, yet useful expressively. (This definition combines expressive generalization, e.g., about prioritization, with expressive restriction, e.g., about dependencies.) The lack of tremendous mathematical complexity is thus a feature and virtue, and is no accident — it is closely related to the simplicity with which the courteous knowledge representation can be explained to, or explained by, a new rules-writer.

10 Current Work

In current work, we are implementing courteous logic programs (acyclic Datalog case) as an improved fundamental knowledge representation in RAISE, for use in building commercial intelligent agent applications (recall section 1). A RAISE **prototype is currently running**.

Our directions of generalizing representationally include: permitting recursion; implicit specificity priority, inheritance, and integrity constraints; and rich reasoning about the prioritization relation.

Our directions for interactive knowledge acquisition include analyzing conflict statically and dynamically, soliciting prioritization rules from users, and inter-agent advice-taking (prioritized merging of rule sets).

Acknowledgements

Thanks to Hoi Y. Chan, my co-conspirator in the RAISE implementation. Thanks also to Leora Morgenstern, Chitta Baral, Gerhard Brewka, Jeffrey Kephart, and anonymous reviewers for discussions and encouragement.

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] Gerhard Brewka. Preferred answer sets. 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.

[6] F. Buccafurri, N. Leone, and P. Rullo. Stable models and their computation for logic programming with inheritance and true negation. *Journal of Logic Programming*, 27(1):5–43, 1996.

[7] Marco Cadoli and Marco Schaerf. A survey on complexity results for nonmonotonic logics. *Journal of Logic Programming*, 17:127–160, 1993.

[8] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[9] D. Etherington and R. Reiter. On inheritance hierarchies with exceptions. In *Proceedings of the Second National Conference on Artificial Intelligence (AAAI-83)*, pages 104–108, Washington, D.C., 1983.

[10] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings of the Fifth International Conference and Symposium*, pages 1070–1080, 1988.

[11] 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.

[12] 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.

[13] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2:397–425, 1992.

[14] Benjamin N. Grosof. Generalizing Prioritization. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR-91)*, pages 289–300, April 1991. Also available as IBM Research Report RC15605, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598.

[15] Benjamin N. Grosof. *Updating and Structure in Non-Monotonic Theories*. PhD thesis, Computer Science Dept., Stanford University, Stanford, California 94305, October 1992. Published by University Microfilms, Inc.. Also available as IBM Research Report RC 20683, dated Jan. 1997; see http://www.research.ibm.com .

[16] Benjamin N. Grosof. Prioritizing Multiple, Contradictory Sources in Common-Sense Learning by Being Told; or, Advice-Taker Meets Bureaucracy. In Leora Morgenstern, editor, *Proceedings of the Second Symposium on Logical Formalizations of Common-Sense Reasoning (Common-Sense '93)*, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, 1993. Available as an IBM Research Report. Copies of the Proceedings are available via the editor (wider publication being arranged). Held Guest Quarters Hotel, Austin, Texas, Jan. 11-13, 1993.

[17] Benjamin N. Grosof. Conflict Resolution in Advice-Taking and Instruction. In Diana Gordon, editor, *Proceedings of the ML-95 Workshop on Agents that Learn From Other Agents*. Proceedings are available on the World Wide Web: http://www.cs.wisc.edu/~shavlik/ml95w1/ . Extended Version of the paper is available as IBM Research Report RC 20123 on the World Wide Web: http://www.research.ibm.com ., July 1995. Workshop held in conjunction with the 1995 International Conference on Machine Learning, held Tahoe City, CA.

[18] Benjamin N. Grosof. Defeasible and pointwise prioritization: Preliminary report. Technical report, IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, USA. http://www.research.ibm.com ., July 1995. IBM Research Report RC 20125. Revised from 4/92 Working Paper and 1/93 Research Report versions.

[19] Benjamin N. Grosof. Practical prioritized defaults via logic programs. In *Proceedings of the Sixth International Workshop on Nonmonotonic Reasoning.* http://www.kr.org/nm/nm96.html and http://www.cs.utexas.edu/users/vl/, 1996. Held Timberline, OR, June 10–12, 1996. Extended version available as IBM Research Report RC 20464, at http://www.research.ibm.com .

[20] Benjamin N. Grosof. Building Commercial Agents: An IBM Research Perspective (Invited Talk). In *Proceedings of the Second International Conference and Exhibition on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM97)*, P.O. Box 137, Blackpool, Lancashire, FY2 9UN, UK. http://www.demon.co.uk./ar/PAAM97, April 1997. Practical Application Company Ltd. Held London, UK. Also available as IBM Research Report RC 20835 at World Wide Web http://www.research.ibm.com .

[21] 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.

[22] Benjamin N. Grosof and Davis A. Foulger. Globenet and RAISE: Intelligent Agents for Networked Newsgroups and Customer Service Support. In *Proceedings of the 1995 AAAI Fall Symposium on AI Applications in Knowledge Navigation and Retrieval*, Menlo Park, CA; World Wide Web http://www.aaai.org, November 1995. American Association for Artificial Intelligence. Held at MIT, Cambridge, MA. Paper also available as IBM Research Report RC 20226. World Wide Web http://www.research.ibm.com .

[23] Benjamin N. Grosof, David W. Levine, Hoi Y. Chan, Colin P. Parris, and Joshua S. Auerbach. Reusable Architecture for Embedding Rule-Based Intelligence in Information Agents. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM-95) Workshop on Intelligent Information Agents*, http://www.cs.umbc.edu/iia/, December 1995. Published via the World Wide Web. Held Baltimore, MD. Paper also available as IBM Research Report RC 20305. World Wide Web http://www.research.ibm.com .

[24] Robert Kowalski and Fariba Sadri. Logic programs with exceptions. *New Generation Computing*, 9:387–400, 1991.

[25] S. Kraus, D. Lehmann, and M. Magidor. Preferential models and cumulative logics. *Artificial Intelligence*, 44:167–207, 1990.

[26] Joao Alexandre Leite and Luis Moniz Pereira. Generalizing updates: From models to programs. 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.

[27] V. Lifschitz. Computing circumscription. In *Proceedings IJCAI-85*, pages 121–127, Los Angeles, CA, 1985.

[28] D. Makinson. General theory of cumulative inference. In M. Reinfrank *et al.*, editor, *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, pages 1–18, Berlin, Germany, 1989. Springer Lecture Notes on Computer Science.

[29] J. McCarthy. Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[30] J. McCarthy. Applications of circumscription to formalizing commonsense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[31] Luis Moniz Pereira, Joaquim N. Aparicio, and Jose J. Alferes. Non-monotonic reasoning with logic programming. *Journal of Logic Programming*, 17:227–263, 1993.

[32] Henry Prakken and Giovanni Sartor. On the relation between legal language and legal argument: Assumptions, applicability, and dynamic priorities. In *Proceedings of the International Conference on AI and Law*, 1995. Washington.

[33] 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.

[34] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 12:81–132, 1980.

[35] D. Touretzky. *The Mathematics of Inheritance Systems.* Pitman, London, 1986.

[36] David S. Touretzky, John F. Horty, and Richmond H. Thomason. A clash of intuitions: The current state of nonmonotonic multiple inheritance systems. In *IJCAI-87*, pages 476–482, San Francisco, CA, 1987. Morgan Kaufmann.

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

[38] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16:277–318, 1993.

[39] 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.

A Appendix: Review of Logic Programming Concepts

Body; Head

In standard logic programming terminology/notation, a rule has the form:
head $\leftarrow$ *body*
body is the antecedent of the rule, and *head* is the consequent of the rule; intuitively, the rule is to be read as: "if *body* then *head*". In a rule, *body* may be empty, but *head* may not be empty. (A query, by contrast, is written in the form $\leftarrow$ *body*)

Logic Programs Without Negation; Stratified

We say that an extended logic program without classical negation, but possibly with negation-as-failure, is "*general*" We say that an extended logic program without classical negation and without negation-as-failure is "*monotonic*" or "*Horn*". The kind of logic programs found in today's implementation, e.g., in the Prolog programming language, are usually in the "general" class.

The Horn class is especially important because of its tractability: in the propositional case, linear time to answer a query, quadratic time for exhaustive forward inference. Even without classical negation, negation-as-failure is problematic in many cases. The conceptual intuition behind negation-as-failure involves invoking a recursive sub-inference: $\sim a$ succeeds when a is not inferred. A sticky issue in general is: what is the set of rules from which a may be inferred in this recursive sub-inference — may it involve the rule which had $\sim a$ in its body (antecedent), hence involving a kind of circularity? The theoretical debate about formal semantics in the logic programming community reflects this problematic character. There is, however, virtual consensus about the semantics of the **stratified** class of general logic programs. This class has a single answer set / model; the recursive sub-inference (for testing negation as failure) can exclude from consideration the rule which spawned the call.

Truth Relative to an Answer Set

Let U be a subset of the ground literals (e.g., an answer set), and let L be a ground literal. $U \models L$ stands for: L is *true* in U, i.e., $L \in U$. One can view $\models$ as entailment, or as satisfaction, or as the answering of queries: for ground formulas.

We extend this notion straightforwardly to permit conjunction and $\sim$ as well:
$U \models (L_1 \wedge \ldots \wedge L_m \wedge \sim L_{m+1} \wedge \ldots \sim L_n)$
when
$U \models L_i \ , \ i = 1, \ldots, m$
$U \not\models L_i \ , \ i = m+1, \ldots, n$
where $n \geq m \geq 0$ (and each L_i is ground).

Atom and Predicate Dependency Graphs

An important concept in the logic programming literature, e.g., in the semantics of stratified logic programs, is that of a dependency graph. (Next, in reviewing it, we follow closely [2], pages 79 and 83.)

Let $\mathcal{E}$ be a logic program (general, extended, or courteous). $\mathcal{E}$'s **predicate dependency graph** $PDG_{\mathcal{E}}$ consists of the predicates as vertices. A triple $\langle p_i, p_j, s \rangle$

is a labelled edge in $PDG_{\mathcal{E}}$ iff there is a rule r in $\mathcal{E}$ with p_i in its head and p_j in its body and the label $s \in \{+,-\}$ denoting whether p_j appears within the scope of the negation-as-failure operator $\sim$ in the body of r. s being $-$ denotes that such appearance is within the scope of $\sim$; s being $+$ denotes that such appearance is not within the scope of $\sim$. Note that an edge may be labeled both by $+$ and $-$.

Intuitively, $\mathcal{E}$'s **atom dependency graph** is analogous to its predicate dependency graph, but has as its vertices ground atoms instead of predicates. The atom dependency graph $ADG_{\mathcal{E}}$ consists of the ground atoms as vertices. A triple $\langle p_i, p_j, s\rangle$ is a labelled edge in $ADG_{\mathcal{E}}$ iff there is a rule r in $\mathcal{E}^{instd}$ (i.e., after replacing rules by their instances) with p_i in its head and p_j in its body and the label $s \in \{+,-\}$ denoting whether p_j appears within the scope of the negation-as-failure operator $\sim$ in the body of r.

$\mathcal{E}$ is said to be **acyclic**, also known sometimes as **non-recursive**, when its atom dependency graph does not have a cycle. Observe that, for any $\mathcal{E}$: if its predicate dependency graph is acyclic, then its atom dependency graph must be acyclic; however, the converse does not hold in general. A cycle in the predicate or atom dependency graph is said to be a *negative cycle* if it contains at least one edge with a negative label.

An acyclic dependency graph is a directed acyclic graph (dag), and thus can be viewed as a strict partial order. Recall that a topological sort of a strict partial order is a total order that is compatible with that partial order. The edges' direction in the dependency graph is from shallower (heads) to deeper (bodies) with respect to dependency; hence in the total stratification sequence of section 3 (which is reversed in direction relative to the dependency graph), earlier corresponds to deeper and later corresponds to shallower. In general, there may be multiple topological sorts for a given dag / strict partial order: i.e., if the given strict partial order is not a total order. Thus there may be multiple total stratifications for a given acyclic extended logic program.

B Appendix: Talk Slides

This appendix contains talk slides that summarize the more basic content of this paper. They are oriented towards a scientific conference audience, suitable for a 25-minute presentation.

Talk Slides
as companion to IBM Research Report RC20836.

(This talk first presented at International Logic Programming Symposium,
Port Jefferson, NY, USA, Oct. 1997.)

Courteous Logic Programs: Prioritized Conflict Handling For Rules

Benjamin N. Grosof

IBM T.J. Watson Research Center
grosof@watson.ibm.com
(or grosof@us.ibm.com or grosof@cs.stanford.edu)
http://www.research.ibm.com/people/g/grosof

Motivating Applications and Requirements

Aim: improve expressive convenience, esp. for KR in ...

Application Area:
Information Flow enhanced by rule-based intelligent Agents

- finding and filtering
- categorizing, prioritizing for attention
- selectively: forwarding, disseminating, sharing

IBM's Agent Building Environment Toolkit Released 7/96

- product alpha, free on Web; C++/Java
- core is our RAISE research engine
- approach revolves around "*situated*" logic program
- innovative procedural attachments & embeddability
- acyclic, Datalog, positive;
  exhaustive forward inferencing cf. deductive DB

Practical ABE applications built:

- e-commerce shopping, customer service,
- e-mail, netnews, ...

Rule base personalizes agent.
User is relatively non-technical.
User is "domain expert" for own "workflow".
e.g., mail handling or shopping interests.
Application-specific rule forms/templates.

Crucial requirements: **facilitate authoring**

- modularity, understandability, predictability
- to debug and trust

Challenge of Conflict Handling; Prioritized Defaults As Approach

like: expressive power of **classical negation**:

- to infer both $\neg p$ via rules and p via rules, with enforcement of mutual exclusion / negation, e.g., important vs. not, urgent vs. not, actions

problem: easy to **get conflicts**, esp. by non-technical authors

want: facilitate **modularity** and **merging** of rule sets

- incl. automatic, inter-agent
- collect 'em and swap 'em with your friends

problem: **2 forms of negation** is conceptually confusing, esp. for non-technical users

inspiration: idea of **prioritized defaults** from general non-monotonic reasoning literature

- deal with conflicts, not give up

like: **partially-ordered** prioritization

- relatively weak, **qualitative** info to specify
- suffices to resolve conflicts for many interesting cases:
  - **updating** cf. deductive DB
  - **specificity** and inheritance
  - legalistic regulations, **authority**

Overview of Courteous LP's

1. ensure **consistency**; unaffected still concluded
2. **simple** to specify **override** (priorities)
3. inferencing **tractable** for Datalog
   (By contrast, conflict handling is a source of NP-hardness for most expressively powerful prioritized default formalisms.)
4. **unique** answer set; thus conceptually simple
5. **includes** consistent "extended" LP and "general" LP (acyclic)
6. **avoid** many typical uses of **negation-as-failure**
   - more disciplined and modular
   - conceptually simpler

Meets desiderata to considerable extent:
- inferencing: practical; usefully expressive
- authoring:
  - ∗ modularity, understandability, predictability
  - \+ rapidly iterated belief revision
  - \+ simulate in author's head

Key to How: **locale** = {rules mentioning predicate p in head}
is focus of conflict handling

Review: Extended Logic Programs [Gelfond & Lifschitz 1991]

Allow classical negation $\neg$. Rule has form

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

where $n \geq m \geq 0$.
Literal L_i is A or $\neg A$, where A is an atom.
$\sim$ is negation-as-failure.

ELP with variables stands for its set of ground instances.

Semantics: **Answer set** is a set of literals that is **stable**. **0, 1, or many** answer sets per ELP, cf. # of extensions in DL.
Unique if stratifiable, e.g., acyclic.

Inconsistency Possible: "blow-up" similar to classical.
Even for simplest expressive fragment.

$$\begin{array}{rl} p & \leftarrow \\ \neg p & \leftarrow \end{array}$$

results in unique answer set containing every literal.

Thus, **does not directly handle conflict**.

Courteous LP's: Definition

"Courteous" because:
respects precedence, i.e., priority between rules.
well-behaved: **consistent unique** answer set.

Based on Extended Logic Programs (ELP).

Augment ELP syntax and modify ELP semantics:

- Add optional **label** (name) to each rule.
- Include **prioritization** facts of form
  $Overrides(i, j) \ \leftarrow$
  $Overrides$ is a special **reserved predicate**.
  $Overrides(i, j)$ means i has higher priority than j.
  $Overrides$ is a **strict partial order** on labels.
- **Locale** is "definition" of one ground atom.
  $Locale(p) \ = \ \{$ all rules whose head is p or $\neg p$ $\}$
- **Stratify** the CLP into locales.
  (Dependency graph)
  The answer set is defined via constructive induction along the stratification.

Courteous LP's, Definition, cont'd

- **Priorities are applied locally.**
  Defined via refutation process with teams of candidate arguments for p versus $\neg p$.

  Each rule whose body "fires" generates a candidate **argument** for its head.
  The argument takes its label from the rule's label.

  Argument i for p **refutes** argument j for $\neg p$ when $Overrides(i, j)$. Vice versa when $Overrides(j, i)$.

  Unrefuted arguments survive. If only one side survives, it is the winner. If two sides survive, they mutually defeat each other: i.e., skepticism is applied.

- **Skepticism** is applied **locally** (after refutation).
  Prevents branching into multiple extensions.
  Consistent. Provides alternative ELP semantics.

- Restrictions here:
  acyclic atom dependency graph.
  (Current work: relax; e.g., recursion)

Example: Pacifism

$$
\begin{array}{lrcl}
\langle Qua \rangle & Pacifist(x) & \leftarrow & Quaker(x) \\
\langle Rep \rangle & \neg Pacifist(x) & \leftarrow & Republican(x) \\
 & Quaker(Nixon) & \leftarrow & \\
 & Republican(Nixon) & \leftarrow &
\end{array}
$$

Inconsistent as an ELP, due to conflict.

CLP is skeptical and consistent; answer set is:

$$\{Quaker(Nixon), Republican(Nixon)\}$$

If add:

$$Overrides(Rep, Qua) \leftarrow$$

Then answer set also has:

$$\neg Pacifist(Nixon).$$

Example: Molluscs; Inheritance

Inheritance chain [Etherington & Reiter, AAAI-83].

CLP can represent prioritization due to specificity.
Obviates need for extra "interaction" conditions that lead to semi-normality in Default Logic representation.

$$
\begin{array}{lrcl}
 & Mollusc(x) & \leftarrow & Cephalopod(x) \\
 & Cephalopod(x) & \leftarrow & Nautilus(x) \\
\langle Mol \rangle & ShellBearer(x) & \leftarrow & Mollusc(x) \\
\langle Cep \rangle & \neg ShellBearer(x) & \leftarrow & Cephalopod(x) \\
\langle Nau \rangle & ShellBearer(x) & \leftarrow & Nautilus(x) \\
 & Overrides(Nau, Cep) & \leftarrow & \\
 & Overrides(Cep, Mol) & \leftarrow & \\
 & Overrides(Nau, Mol) & \leftarrow &
\end{array}
$$

E.g., with

$$
\begin{array}{l}
Mollusc(Molly) \leftarrow \\
Cephalopod(Sophie) \leftarrow \\
Nautilus(Natalie) \leftarrow
\end{array}
$$

the answer set for the $ShellBearer$ predicate is

$$
\begin{array}{l}
\{ShellBearer(Molly), \\
\neg ShellBearer(Sophie), \\
\ ShellBearer(Natalie)\}.
\end{array}
$$

Version WITH Default Logic style interaction conditions: Example: Molluscs; Inheritance

Inheritance chain [Etherington & Reiter, AAAI-83].

CLP can represent prioritization due to specificity.
Obviates need for extra "interaction" conditions that lead to semi-normality in Default Logic representation.

$$
\begin{array}{lrcl}
 & Mollusc(x) & \leftarrow & Cephalopod(x) \\
 & Cephalopod(x) & \leftarrow & Nautilus(x) \\
\langle Mol \rangle & ShellBearer(x) & \leftarrow & Mollusc(x) \wedge \sim CEPHALOPOD(x) \\
\langle Cep \rangle & \neg ShellBearer(x) & \leftarrow & Cephalopod(x) \wedge \sim NAUTILUS(x) \\
\langle Nau \rangle & ShellBearer(x) & \leftarrow & Nautilus(x) \\
 & Overrides(Nau, Cep) & \leftarrow & \\
 & Overrides(Cep, Mol) & \leftarrow & \\
 & Overrides(Nau, Mol) & \leftarrow &
\end{array}
$$

E.g., with

$$
\begin{array}{l}
Mollusc(Molly) \leftarrow \\
Cephalopod(Sophie) \leftarrow \\
Nautilus(Natalie) \leftarrow
\end{array}
$$

the answer set for the $ShellBearer$ predicate is

$$
\begin{array}{l}
\{ShellBearer(Molly), \\
\neg ShellBearer(Sophie), \\
\ ShellBearer(Natalie)\}.
\end{array}
$$

Conflict & Augmentability in Example Personal E-Mail Agent

$\Delta_{initial}$:

$\langle Junk\rangle\ \neg Important(?msg)$
$\quad\leftarrow\ From(?msg, ?x) \wedge RetailStore(?x).$

$\langle Delv\rangle\ \ Important(?msg)$
$\quad\leftarrow\ From(?msg, ?x) \wedge AwaitingDelivery(Karen, ?x).$

$AwaitingDelivery(Karen, ParisCo).$

$RetailStore(FaveCo)\leftarrow.$

$RetailStore(BabyCo)\leftarrow.$

$RetailStore(ParisCo)\leftarrow.$

...

$\Delta_{mail}:\quad From(msg54, BabyCo)\leftarrow.$
$\quad\mid\!\sim \neg Important(msg54)$

...

$\Delta_{mail}:\quad From(msg81, ParisCo)\leftarrow.$ **conflict case**
$\quad\not\mid\!\sim Important(msg81)\ ,\ \not\mid\!\sim \neg Important(msg81)$

$\Delta_{instruct}:\quad Overrides(Delv, Junk)\leftarrow.$

augment: priority

$\quad\mid\!\sim Important(msg81)$

...

$\Delta_{mail}:\quad From(msg117, FaveCo)\leftarrow.$
$\quad\mid\!\sim \neg Important(msg117)$

$\Delta_{instruct}:$

$\langle Fave\rangle\ \ Important(?msg)\ \leftarrow\ From(?msg, FaveCo).$

$Overrides(Fave, Junk)\leftarrow.$ **specificity**

$\quad\mid\!\sim Important(msg117)$

Example: Basic Behavior, Localization of Conflict, Chaining

$$p \leftarrow .$$
$$\neg p \leftarrow .$$
$$q \leftarrow p.$$
$$r \leftarrow \neg p.$$
$$s \leftarrow .$$
$$\neg t \leftarrow .$$
$$\neg u \leftarrow s \wedge \neg t \wedge \sim v.$$
$$w \leftarrow \neg u \wedge \sim p.$$

has answer set $\{s, \neg t, \neg u, w\}$

- unresolved conflict about p is localized.

Results: Well-Behavior and Inferencing

- Consistent, unique answer set.
- Subsumes "general" LP (acyclic).
- Agrees with ELP when ELP is consistent.
  Alternative semantics for ELP.
- Tractable inferencing.
  Overhead is relatively low for conflict handling feature.
  $O(m^2)$ to compute entire answer set
  $m =$ size of *instantiated* LP
  vs.
  $O(m)$ to compute for "general" LP (acyclic).

  Refutation is core of extra cost.

  (m finite if Datalog.
  $m = n^{v+1}$, where $n =$ size of uninstantiated LP,
  when Datalog & bounded $\#$ v of variables per rule.)
- Algorithm: for exhaustive forward inferencing
  is straightforward (see expanded version of paper).
- (Current work:) Priorities behave similarly to
  circumscription and Prioritized Default Logic.

More about Behavior

- conflict resolution is
  - local,
  - pairwise,
  - atomic
- no propagation of ambiguity
- cumulative
- to force a conclusion p:
  include $p \leftarrow$ at highest-priority
- for database-flavor updating:
  recency priority; simple, modular
- for Closed World Assumption:
  include $\neg P(x) \leftarrow$ at lowest-priority
- obviate most need for explicit negation-as-failure
- merging and advice taking:
  well-behaved composition of modules' conclusions

Teams in Refutation, Merging

- Naive alternative approach to defining refutation:
  "top dog" / "single combat"
  p wins iff there is a candidate for p that
  refutes all opposing candidates

  Problematic Example:
  simple merge of 2 cloned modules
  does not preserve conclusions!

  $\langle 1a \rangle \quad p \;\leftarrow$
  $\langle 1b \rangle \; \neg p \;\leftarrow$
  $\langle 1c \rangle \quad Overrides(1a, 1b) \;\leftarrow$
  $Module1 \mid\!\sim p$

  $\langle 2a \rangle \quad p \;\leftarrow$
  $\langle 2b \rangle \; \neg p \;\leftarrow$
  $\langle 2c \rangle \quad Overrides(2a, 2b) \;\leftarrow$
  $Module2 \mid\!\sim p$

  but: $(Module1 \cup Module2) \not\mid\!\sim p$!!

- **Team** aspect of refutation in courteous:
  - $*$ handles well a nasty **subtlety** in prioritization
  - $*$ facilitates **merging**:
    - o prioritized or unprioritized merges
    - o "advice-taking" between agents

Related Work

- Brewka's form of prioritized ELP's [independent]:
  – similar syntax
  – different semantics: modifies well-founded; prioritizes $\sim$
  – emphasis: cyclicity, reasoning about $Overrides$
  – $O(m^3)$ time inference
  – problematic for ex. where team refutation needed
  – no consistency guarantee
- Abnormalities in ELP's [Baral & Gelfond]:
  no general method
- Hierarchical Constraint LP [Wilson & Borning]:
  soft constraints, used abductively.
- Consistency and priorities between abnormalities, in ELP's [Pereira $et\ al$]. Force in $\sim$ literals contrapositively.
- Prakken & Sartor; Buccafurri $et\ al$; Kowalski & Sadri
- ILPS-97 and its KR workshop: several on prioritized LP's
  Area seems to be heating up!

Current Work

Current Status: **prototype** running as new RAISE engine.

- Further implementation; workflow applications
- Generalizing representation:
  - recursion,
  - inheritance and specificity,
  - advice taking; reasoning about prioritization
- User interaction
- Relationships to other formalisms

Courteous LP's: Advantages

- Consistent set of conclusions, always
- Computationally low overhead
- Priorities; subtlety: teams
- Clean semantics, conceptual simplicity $\Longrightarrow$:
- **Human-agent** interaction: instruct, edit; predict, explain
- **Inter-agent** communication: merge, update
- Learning: from advice or induction
- Embedding: situating; SQL DB's
- Other rule-based applications: interface; direct use