RC 20464 (05/20/96; revised 05/29/96 and 06/07/96)
Computer Science

IBM Research Report

Practical Prioritized Defaults via Logic Programs

Benjamin N. Grosof

IBM Research Division
T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
(914) 784-7100 main
Internet: grosof@watson.ibm.com (alt.: grosof@cs.stanford.edu)
WWW: http://www.research.ibm.com

Publication Information: *This paper is an extended, revised, and reformatted version of the paper to appear in the Proceedings of the Sixth International Workshop on Nonmonotonic Reasoning, to be held Timberline, OR, June 10–12, 1996, editors Moises Goldszmidt and Vladimir Lifschitz (http://www.cs.utexas.edu/users/vl/). Information, and soon on-line Proceedings, for the Workshop are available via the World Wide Web (interim URL: http://www.kr.org/nm/nm96.html).*

Abstract

We define courteous logic programs, a variant of extended logic programs. Courteous logic programs constitute a kind of prioritized default reasoning that is computationally tractable, fairly powerful expressively, and commercially applicable to the creation of rule-based intelligent information agents.

Priorities are, in part, specified explicitly: within the locale of each predicate's "definition", i.e., between rules mentioning the same predicate in their consequent. (Recall that extended logic programs represent classical negation as well as negation-as-failure. Hence a rule having consequent p may conflict with another rule having consequent $\neg p$.) In addition, priorities are, in part, specified implicitly: based on inferential dependency depth between the different predicate locales, as in the usual semantics for negation as failure in stratified logic programs.

Syntactically, a courteous logic program is an extended logic program augmented with a prioritization sub-program. In this paper, we restrict our attention to the non-recursive Datalog case. We show not only that there always exists a unique answer set, but also that the answer set is guaranteed to be consistent, unlike in the previous concept of extended logic programs. We give a tractably-computable reduction of courteous logic programs to extended logic programs. We show furthermore that the entire answer set can be computed in time polynomial in the size of the courteous logic program.

We illustrate with examples the representational adequacy and intuitive behavior of courteous logic programs for several interesting scenarios of prioritized default reasoning.

Courteous logic programs provide a method to resolve conflicts in authoring (specifying), updating, merging, and advice-taking during the creation of rule-based intelligent agents. Their prioritized default flavor enables a more common-sense style of specification, especially important for non-technical authors.

In current work, we are implementing courteous logic programs as the fundamental knowledge representation for commercial intelligent agent applications such as personalized information filtering and workflow, e.g., cf. the Globenet+RAISE system for networked newsgroups and customer service support at IBM.

Other current work includes developing methods for interactive knowledge acquisition, and generalizing representationally.

Keywords: *default reasoning, priorities, nonmonotonic reasoning, logic programming, intelligent agents. More keywords: applications, knowledge acquisition, Internet, usability, locality, modularity, tractability, negation as failure, Prolog, updating, belief revision, advice-*

taking, artificial intelligence, machine learning.

Version Notes; Follow-on Papers; Talk Slides: *The level of detail given in this version of this paper varies: more about the formalism's definition and basic properties, and less about most else. This weighting is in part a hangover from the space constraints in the Workshop Proceedings. It is also oriented towards a relatively specialized technical audience already familiar with nonmonotonic reasoning and logic programming. For updated versions and follow-on papers, see Research Reports and personal/project pages available via the World Wide Web http://www.research.ibm.com . This version corrects two technical errata in the May 1996 versions. The first is in the statement of the reduction transform from courteous to extended (Definition 13). The second is in the statement of the result on prioritized merging (Theorem 11). Talk slides will soon be available via the Web site, e.g., in postscript.*

1 Motivations

An interesting application area for rule-based inferencing that motivates our work is intelligent information agents. In such agents, a rule base is an interesting approach to specifying the control of filtering and handling of information. Handling tasks include, for example: categorizing, prioritizing for attention, storing and managing, and selectively disseminating and sharing.

Example intelligent information agent applications include: (incoming) e-mail; news items from networked newsgroup-like sources (including electronic conferencing forums and bulletin boards); and customer service support based on such newsgroup-like resources. For example, the RAISE system ("Reusable Agent Intelligence Software Environment") is a object-oriented class library that has been used to embed rule-based intelligence into information agent applications [Grosof and Foulger, 1995] [Grosof *et al.*, 1995].

An important requirement is to ease the process of authoring knowledge bases, i.e., rule sets: e.g., syntactic natural-ness, easy-to-understand semantics, ease in debugging and modification, and facilitation of sharing / merging knowledge bases. In particular, a major goal of current work in the field of intelligent agents is to facilitate the process of authoring by non-technical users. A major difficulty is the tendency of such authors to generate conflicting rule sets. They tend to "think common-sense-ically", including defeasibly.

The concept of priorities from the field of default reasoning offers an attractive tool to cope with conflicting rules. Rule priorities are naturally available based, e.g., on relative freshness, specificity, authority, and reliability [Grosof, 1991] [Grosof, 1995].

However, a challenge for prioritized default reasoning is to find a middle ground that maintains both computational tractability and adequate expressiveness. We believe that courteous logic programs occupy such a useful position.

2 Preliminary Definitions

Background: We assume the reader is familiar with extended logic programs [Gelfond and Lifschitz, 1991], with the semantics of stratified (ordinary, non-extended) logic programs with negation as failure (e.g., [Przymusinski, 1988]), and with the standard concepts in the logic programming literature (e.g., as reviewed in [Baral and Gelfond, 1994]), including predicate / atom dependency graph and its acyclicity / non-recursiveness; and instantiation. The **Appendix** contains some review of these concepts.

Recall that each *rule* r in an extended logic program has the form:

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

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

Following [Gelfond and Lifschitz, 1991]: we will define the semantics of courteous logic programs using the concept of an **answer set**. An answer set is a subset of the ground literals. We write $\models$ to stand for truth relative to an answer set.

As Gelfond & Lifschitz define its semantics, an extended logic program 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 logic program has one answer set which is the set of all ground literals. Intuitively, this corresponds to the situation of inconsistency in classical logic, in which an inconsistent set of axioms entails every closed formula (i.e., any conclusion is sanctioned). For example, the extended logic program consisting of the two rules

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

is inconsistent. In the terminology of default reasoning, the two rules conflict with each other. The answer set contains both the literals p and $\neg p$.

In the remainder of this section, let $\mathcal{E}$ be an extended logic program.

The semantics of extended logic programs ([Gelfond and Lifschitz, 1991]) treats a rule with variables as shorthand for the set of its ground instances. We will do likewise with courteous logic programs. We write $\mathcal{E}^{instd}$ to stand for the program in which each rule in $\mathcal{E}$ having variables has been replaced by the set of all its possible ground **instantiations**.

Another important concept in the logic programming literature, e.g., in the semantics of stratified logic programs, is that of a **definitional locale** of a predicate or ground atom.

Relative to $\mathcal{E}$: the definitional locale for a *predicate* p, written as $Defn(p)$, is defined as the subset of rules within $\mathcal{E}$ in which p appears in the rule head. Similarly: the definitional locale for a *ground atom* p, also written as $Defn(p)$, is defined as the subset of rules within $\mathcal{E}^{instd}$ (i.e., after replacing rules by their instances) in which p appears in the rule head. We will also speak more tersely of $Defn(p)$ as the "*definition*" or "*locale*" of p. Note that a definitional locale may be an empty set.

A third important concept in the logic programming literature is that of stratification of predicates or atoms. We will employ a special case of it.

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

Associated with the total ordering of the atoms (respectively, predicates) is the associated totally ordered partition of $\mathcal{E}^{instd}$'s (respectively, $\mathcal{E}$'s) rules into definitional atom (respectively, predicate) locales $Defn(p_1), \ldots, Defn(p_m)$. [1] Each **stratum**, i.e., element in this partition, is a single atom's (respectively, predicate's) definitional locale.

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.

It is named "courteous" for several 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 used by [Gelfond and Lifschitz, 1991]: it has **exactly one answer set**, and this answer set is **consistent**.

The focus of (explicit) prioritization in courteous logic programs is within definitional locales, i.e., between rules that mention the same predicate symbol in their head. Such appearance may be with negative (or positive) classical sign; two rules having opposite sign may conflict. In extended logic programs, such conflict may result in inconsistency of the entire program.

In addition, priorities are, in part, specified implicitly: based on inferential dependency depth between the different predicate (or atom) locales, as in the usual semantics for negation as failure in stratified logic programs.

In the remainder of the paper, "prioritization" will refer to **explicit** prioritization, unless specifically prefixed with "implicit" or otherwise made clear in the context.

In the remainder of this section, $\mathcal{C}$ stands for a courteous logic program.

Labelling of rules: Each rule of $\mathcal{C}$ is labelled, i.e., has the form:

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

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

Labels are optional syntactically, i.e., a label may be empty. Semantically, 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).

Labels need not be unique, i.e., two rules may have the same label.

Labels are preserved during instantiation. All the ground instances of a rule r have the same label as r. I.e., in $\mathcal{C}^{instd}$, each instance of the above rule has label *lab*.

Prioritization sub-program: $\mathcal{C}$ includes a prioritization sub-program $\mathcal{C}_R$. This prioritization sub-program is a subset of the overall program's ($\mathcal{C}$'s) rules. It has the form of a set, possibly empty, of positive (ground) facts about a special binary predicate R. $R(i,j)$ specifies that the label i has (strictly) higher priority than the label j. We call each such fact a **prioritization fact**. Conceptually, R can be read as shorthand for "*Overrides*"; $R(i,j)$ says that i overrides j.

R is syntactically reserved: it must not appear in any rules other than within the prioritization sub-program. R takes only labels as arguments. Labels are treated as part of the language of the logic program for this purpose.

The prioritization relation R specified by the prioritization sub-program is required to be a **strict partial order**, i.e., transitive and anti-symmetric.

[1] Remember that $\mathcal{E}$'s atom dependency graph is actually defined most directly in terms of $\mathcal{E}^{instd}$.

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.

$\mathcal{C}$'s non-prioritization sub-set $\mathcal{C}_{main} = \{\mathcal{C} - \mathcal{C}_R\}$ is called the **main** part of $\mathcal{C}$, or, more simply, the *main (sub-)program*.

Acyclicity: $\mathcal{C}$ (i.e., its atom dependency graph) is required to be acyclic.

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

Answer Sets: $\mathcal{C}$ has a unique answer set S. We define this answer set 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.

First, let us select a **total atom stratification** ρ of $\mathcal{C}$. Recall that, in general, there may be multiple total stratifications of a given program. Any choice among these will do, subject to the following constraint on the chosen stratification's sequence: **the prioritization atoms must come before all the main atoms**.

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

$S_0 = \emptyset$

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

$S = \bigcup_i T_i$

where $\emptyset$ stands for the empty set, and

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

$Cand_i^{\sigma} = \{j \mid labels(j,r) \ , \ Head(r) = \sigma p_i \ , \ S_{i-1} \models Body(r)\}$

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.

This iterative construction can be described as follows. 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 R 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 $R(j,k)$ is true in the previous answer set iterate S_{i-1}. Note that $R(j,k)$ is true iff

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

is present as a prioritization fact in the prioritization sub-program $\mathcal{C}_R$. 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 R, it cannot happen that both teams refute each other.) This outcome of mutual defeat corresponds to **skepticality**: if there are un-refuted candidate arguments both for p_i and for $\neg p_i$, then no conclusion about p_i is drawn.

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

The atom dependency graph and its associated total **stratification can be viewed as implicit prioritization**, affecting especially `fail` , as in the semantics of stratified (ordinary) logic programs cf. [Przymusinski, 1988] [Lifschitz, 1987].

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 R may compare two labels i and j that belong to different locales. Such priority is ignored by the semantics. In effect, the only prioritization between locales is implicit, based on the dependency graph.

The syntactic class of courteous logic programs can be viewed as an augmentation of the class of acyclic extended logic programs: by adding labels and a prioritization sub-program. Since the labels and prioritization sub-program may be empty, one can take any acyclic extended logic program and interpret it as a courteous logic program. As we will see, this results in identical semantics for consistent extended programs; however, in the presence of conflict, the semantics differ in that the **courteous interpretation of an extended program is consistent**.

4 Consistency and Well-Behavior

Theorem 1 (Consistency; Agreement with Extended Semantics)
(I) Every courteous logic program $\mathcal{C}$ has exactly one answer set which is consistent.
(II) If $\mathcal{C}$'s main part $\mathcal{C}_{main}$ is consistent when interpreted semantically as an extended logic program $\mathcal{E}_{main}$, then $\mathcal{C}$'s answer set is identical to $\mathcal{E}_{main}$'s answer set.

Proof Sketch: (I) The answer set construction implies there exists exactly one answer set. Consistency is shown by induction on the stratification in the answer set construction. In the inductive step, we show that each locale's contribution is consistent, and preserves the consistency of the previous answer set iterate.

(II) If the main sub-program is consistent, there is no conflict. The prioritization and labels are then irrelevant. □

5 Basic Examples

Example 2 (Pacifism; Skepticality and Explicit Priority)

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

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

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:

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

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

Example 3 (Molluscs; Inheritance)

Etherington & Reiter's [1983] 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 semi-normal there.

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

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

$$\{ShellBearer(Molly), \neg ShellBearer(Sophie), ShellBearer(Natalie)\}.$$

6 Examples: Personal E-Mail Agent

Next, we give two examples of a rule base that 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 4 (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:

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

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.

Example 5 (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:

$\langle Jun \rangle \ \neg Important(msg) \ \leftarrow \ From(msg, x) \wedge RetailStore(x)$
$\langle Del \rangle \quad Important(msg) \ \leftarrow \ From(msg, x) \wedge AwaitingDeliveryFrom(Karen, x)$

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.,
$RetailStore(FaveCo) \ \leftarrow$
$RetailStore(BabyCo) \ \leftarrow$
$RetailStore(ParisCo) \ \leftarrow$

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. Next, another mail item arrives:

$$From(msg55, ParisCo) \;\leftarrow$$

The courteous program has a conflict between the first two rules, and hence draws no conclusion about $Important(msg55)$. 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

$$R(Del, Jun) \;\leftarrow$$

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:

$$\langle Fav \rangle \quad Important(msg) \;\leftarrow\; From(msg, FaveCo)$$

$$R(Fav, Jun) \;\leftarrow$$

7 Discussion; Properties and Behavior

The above examples illustrate that courteous logic programs have more expressive power on several dimensions than default inheritance systems in the vein of [Touretzky, 1986]: e.g., multiple body conditions, multiple variables, prioritization beyond specificity, and (though we did not see it in the examples) appearance of negation (including by failure) in the body.

On the other hand, **courteous logic programs are much less complex computationally and conceptually than relatively expressively powerful formalisms for prioritized default reasoning such as prioritized circumscription** [McCarthy, 1986] [Lifschitz, 1985] [Grosof, 1991] [Grosof, 1992] **and Prioritized Default Logic** [Brewka, 1994]. **Courteous logic programs do not spawn multiple extensions** or model classes as often happens in Default Logic [Reiter, 1980], circumscription [McCarthy, 1980], and their prioritized versions. 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 inferencing** [Gottlob, 1992].

Part of courteous logic programs' simplicity derives from the logic programming approach: e.g., **definiteness** (lack of "carrying around" or "propagating" disjunctions) and **lack of contrapositivity**; in particular, these properties keep the underlying monotonic inferencing tractable.

Another key to courteous logic programs' computational simplicity is that the **prioritization and conflict resolution is local, pairwise, and atomic**.

By local, we mean restricted to rules (instances, actually) 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), 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 default 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 6 (Conflicting Chains; Pairwise-ness)

The following rule set illustrates the pairwise-ness of conflict resolution and prioritization. Here, two chains of implication conflict with each other. The first chain consists of three rules that together "argue" for e:

$e \leftarrow b$, $b \leftarrow a$, $a \leftarrow$

The second chain consists of three rules that together "argue" for $\neg e$:

$\neg e \leftarrow d$, $d \leftarrow c$, $c \leftarrow$

In the courteous formalism: no conclusion is drawn about e; however, the resulting answer set does include: $\{a, b, c, d\}$.

Local Monotonicity of Prioritization

Theorem 7 (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. □

More prioritization may resolve conflict in the sense of creating winners where before there was mutual defeat. The Pacifism example (Example 2) 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.

Forcing; Cumulativity; Hypotheticals

Courteous programs also have a relatively simple way to **force a conclusion**: simply add a new rule, e.g., a fact, at highest priority within its locale.

Related to this point is the following property. Courteous programs are **cumulative** in the sense of [Makinson, 1989] [Kraus *et al.*, 1990]: if p is a conclusion of $\mathcal{C}$, then adding the fact $p \leftarrow$ to $\mathcal{C}$ results in a new program that has the same answer set as $\mathcal{C}$.

Closed World Assumption

The **Closed World Assumption** (CWA) for any given literal (e.g., predicate), can be represented in a simple fashion using priority. 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. Likewise, to maximize it, include $p(x) \leftarrow$ instead, with similar priority. Actually, the priority mechanism allows more subtlety in the control of the CWA: for example, it might be given highest or some other priority.

By contrast, [Gelfond and Lifschitz, 1991] give an approach to representing the CWA in extended programs which involves cyclicity of the dependency graph, and hence is disallowed in courteous programs. In their approach, CWA for p is represented by the rule $\neg p(x) \leftarrow \texttt{fail}\ p(x)$.

The representation of the CWA exemplifies a larger point. **The prioritization mechanism reduces the need for the representational use of negation as failure.** 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.

No Propagation of Ambiguity

Courteous programs **do not propagate ambiguity** in the sense of [Touretzky *et al.*, 1987]'s paper on "clash of intuitions" in default reasoning.

Example 8 (No Propagation of Ambiguity)

E.g., consider the program formed by adding the following rules to the unprioritized Pacifism example (Example 2):

$$\begin{array}{rcl} FootballFan(x) & \leftarrow & Republican(x) \\ AntiMilitary(x) & \leftarrow & Pacifist(x) \\ \neg AntiMilitary(x) & \leftarrow & FootballFan(x) \end{array}$$

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), i.e., 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.

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 of $AntiMilitary(Nixon)$ nor of $Pacifist(Nixon)$ is present in the intersection of all these multiple extensions. We can call this intersecting (i.e., disjoining) after extension-spawning: "global" skepticism. By contrast, the courteous semantics has "**local**" **skepticism**.

8 Merging Rule Set Modules; Advice Taking

Courteous programs have desirable properties with regard to **merging** of rule sets, i.e., of (sub-)programs.

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**. (In this paper, for simplicity's sake in defining merging, 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 9 (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 Sketch: 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.

Likewise, the set of refutation pairs after the merge is 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. By refutation pair here, we mean an ordered pair of candidate arguments where the first refutes the second.

Hence, there are unrefuted candidate arguments for p, but none for $\neg p$, after the merge. □

Example 10 (Preservation in Parallel Merging)
Let $\mathcal{C}_1$ consist of

$$\begin{array}{ll} \langle 1 \rangle & p \leftarrow a \\ & a \leftarrow \\ \langle 2 \rangle & \neg p \leftarrow b \\ & b \leftarrow \\ & R(1,2) \leftarrow \end{array}$$

$\mathcal{C}_1$'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

$$\begin{array}{ll} \langle 3 \rangle & \neg p \leftarrow c \\ & c \leftarrow \\ \langle 4 \rangle & p \leftarrow d \\ & d \leftarrow \\ & R(4,3) \leftarrow \end{array}$$

$\mathcal{C}_2$'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.

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 $\mathcal{C}_2$.

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 [Grosof, 1991] [Grosof, 1992] [Grosof, 1995].

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 [Grosof, 1995]. 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 [Grosof, 1991] [Grosof, 1995]. 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 11 (Preservation in Prioritized Merging)

Suppose p is a conclusion of $\mathcal{C}_1$. Then p is a conclusion of $\mathcal{C}_{1>2}$.
Proof Sketch: Inductive along the stratification. 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). □

Example 12 (Preservation in Prioritized Merge)

Let $\mathcal{C}_1$ consist of

$p \leftarrow a$
$a \leftarrow$
$q \leftarrow b \wedge c$
$b \leftarrow$
$c \leftarrow$

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

Let $\mathcal{C}_2$ consist of

$\neg p \leftarrow d$
$d \leftarrow$
$\neg a \leftarrow$
$\neg q \leftarrow b$

$\mathcal{C}_2$'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 sketch of Theorem 11.) 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.

9 Reduction to Extended Logic Programs

We would like to take advantage of the analytic understanding and available algorithms that belong to extended logic programs and their sub-classes, including ordinary ("general") logic programs. Recall that an attractive feature of extended logic programs is that when consistent they can be reduced by a simple equivalence transform to ordinary logic programs [Gelfond and Lifschitz, 1991]. In that transform, $\neg p$ is made into a new predicate separate from p, for any original predicate p.

Next, we define a computationally relatively simple transform that equivalently reduces any courteous logic program to an equivalent extended logic program. The transform is defined per locale, essentially syntactically (unifications are performed, however). It corresponds relatively closely to the explanation we gave in the last section of the prioritized competition among candidates. However, the locales are by predicate, i.e., in terms of the original representation, rather than by atom. This has advantages for implementation and knowledge acquisition.

Definition 13 (Reduction Transform)

Let $\mathcal{C}$ be a courteous logic program. We define its **ELP reduction transform** $\texttt{CR}(\mathcal{C})$ as follows. ($\texttt{CR}()$ is mnemonic for "Conflict Resolution").

The transform is defined in terms of a transform for each predicate locale.

For each predicate q, let $Defn^{\sigma}(q)$ contain all the rules of q's definitional locale in which the head literal has sign σ.

We can write the j^{th} rule r_j in $Defn^{\sigma}(q)$ as:

$\sigma q(t_j[x_j]) \leftarrow A_j[y_j]$

where t_j a term mentioning tuple of variables x_j, and A_j is an body expression mentioning a tuple of variables y_j which may intersect with x_j.

Similarly, we can write the k^{th} rule r_k in $Defn^{\neg\sigma}(q)$ as:

$\neg\sigma q(t_k[x_k]) \leftarrow A_k[y_k]$

Let lab_j and lab_k stand for the labels of the rules r_j and r_k, respectively. Let $Unifiable(t_j, t_k)$ stand for the proposition that the terms $t_j[x_j]$ and $t_k[x_k]$ are unifiable. If they are indeed unifiable, let $mgu(t_j, t_k)$ stand for their maximum general unifier. Let the $\cdot$ operator stand for the usual unification concept of applying a substitution.

We are now ready to give locale $Defn(q)$'s contribution to the transform's output $\texttt{CR}(\mathcal{C})$.

1. For each rule $r_j \in Defn^{\sigma}(q)$, $\texttt{CR}(\mathcal{C})$ contains the rule

$q^{\sigma}_{unrefuted}(t_j[x_j]) \;\leftarrow\; A_j[y_j] \;\wedge\; \texttt{fail}\; refuted_j(t_j[x_j])$

where $refuted_j$ is a newly introduced predicate symbol. Intuitively, for a given ground instance u, $refuted_j(u)$ represents the proposition that rule r_j's u^{th} instance, if its body is true and it thus is a candidate argument, is nevertheless refuted by another higher-priority candidate. $q^{+}_{unrefuted}$ and $q^{\neg}_{unrefuted}$ are each a newly introduced predicate symbol. Intuitively, $q^{\sigma}_{unrefuted}(u)$ represents the proposition that there is an unrefuted candidate argument for $\sigma q(u)$.

2. For each pair of unifiable opposing rules $r_j \in Defn^{\sigma}(q)$ and $r_k \in Defn^{\neg\sigma}(q)$ such that $Unifiable(t_j, t_k)$, $\texttt{CR}(\mathcal{C})$ also contains the rule

$(refuted_j(t_k[x_k]) \cdot mgu(t_j, t_k)) \;\leftarrow\; (A_k[y_k] \cdot mgu(t_j, t_k)) \;\wedge\; R(lab_k, lab_j)$

Intuitively, this rule represents the process of refutation. Note that if lab_j or lab_k is empty, the R atom above is false and can equivalently simply be dropped from the rule.

3. Finally, for each sign σ, $\texttt{CR}(\mathcal{C})$ also contains the rule

$\sigma q(x) \;\leftarrow\; q^{\sigma}_{unrefuted}(x) \;\wedge\; \texttt{fail}\; q^{\neg\sigma}_{unrefuted}(x)$

where x is a tuple of variables, having the same arity as q. Intuitively, this rule represents the processes of winning and mutual defeat.

Theorem 14 (Equivalence of Reduction)

$\texttt{CR}(\mathcal{C})$ has the same answer set as $\mathcal{C}$, when restricted to the original language (i.e., considering only each original predicate q and ignoring each $q^{\sigma}_{unrefuted}$ and each $refuted_j$).

Proof Sketch: Inductive along the total atom stratification. The unification aspect handles the mapping to the atom locales. □

Theorem 15 (Tractability of Transform)

Let n stand for the size of $\mathcal{C}$. The reduction transform $\texttt{CR}()$ for $\mathcal{C}$ can be computed in time polynomial in n. The size of the resulting extended program is also polynomial in n.

10 Inference Algorithms and Complexity

Our results about the reduction transform CR() imply an attractive property: inferencing in courteous logic programs can be implemented by **tractably piggybacking** on available inferencing algorithms for extended logic programs, which can in turn be implemented by tractably piggybacking ([Gelfond and Lifschitz, 1991]) on available inferencing algorithms for ordinary logic programs, e.g., Prolog for **query-answering**.

Next, we give an algorithm for inferencing directly in the courteous representation, and show its complexity to be tractable.

Theorem 16 (Tractability of Inference; Direct Algorithm)

Suppose that $\mathcal{C}$ has no function symbols with arity greater than 0: i.e., its rules are restricted to fulfill the **Datalog** constraint. Suppose furthermore that there is a finite upper bound v on the number of variables appearing in any one rule.

Then $\mathcal{C}$'s entire answer set can be computed in time polynomial in the size of $\mathcal{C}$.

Algorithm and Proof, Sketch: Let n stand for the size of $\mathcal{C}$. The given conditions on $\mathcal{C}$ ensure that the Herbrand base is $O(n)$, and that the number of ground rule instances is $O(n^{v+1})$. The number of atoms is thus also $O(n^{v+1})$.

A conceptually simple algorithm (for exhaustive forward inferencing) is then to directly implement the answer set construction that we gave in defining courteous programs. (An alternative is to use the reduction to extended programs, as we discussed earlier.)

Instantiation: Syntactically expand $\mathcal{C}$ to form $\mathcal{C}^{instd}$.

Stratification: Topological sort of a graph of size g (edges) can be done in time polynomial in g.

Atom locales: Consider an atom locale. Let d be its size. Generating candidates can be computed simply because truth (e.g., of rule bodies) in the previous answer set iterate can be tested quickly, essentially by lookup operations. There are at most $O(d^2)$ pairs of opposing rules to consider in the refutation process. □

11 Related Work

Prioritized default circumscription [Grosof, 1991] [Grosof, 1992] (generalizing [McCarthy, 1986] [Lifschitz, 1985]) has a similar, though more general, concept of prioritization. However, circumscription behaves contrapositively, and thus quite differently from logic programs.

Baral & Gelfond [1994] discuss techniques for achieving some of the effects of priorities within extended logic programs, e.g., via abnormalities, but do not give a general method. Pereira *et al* [1993] address the issues of consistency, and priorities between abnormalities, in extended logic programs. Their approaach to modifying the semantics is quite different in spirit and detail from ours, e.g., they force in `fail` 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*.

Extended logic programs are equivalent to a special case of Default Logic, as Gelfond & Lifschitz showed. Prioritized variants of Default Logic (see [Brewka, 1994], which includes a review of that literature) are thus a target for comparison. Courteous logic programs' semantics differ significantly from that of these Prioritized Default Logic (PDL) variants: they propagate ambiguity and proliferate multiple extensions (of which there are exponentially many, in the worst case).

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.

Hierarchical Constraint Logic Programming (HCLP) [Wilson and Borning, 1993] 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.

12 Current Work

In current work, we are **implementing** courteous logic programs as the fundamental knowledge representation **for commercial intelligent agent applications** such as personalized information filtering and workflow, e.g., cf. the Globenet+RAISE system for networked newsgroups and customer service support at IBM.

Other current work includes developing methods for interactive knowledge acquisition, and generalizing representationally.

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

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).

13 Conclusions

Courteous programs do not spawn multiple extensions, and thus avoid a source of (worst-case) intractability beyond the underlying monotonic-logical inferencing. The locality of skepticism and prioritization is crucial in this regard: it nips in the bud such branching into multiple extensions. Together with the (inter-locale) stratification, this locality ensures a single answer set.

Tractability and uniqueness of the answer set are highly attractive properties, in our view. They make reasoning be practical: including exhaustive forward inferencing and rapidly iterated belief revision. But more than that, they make **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 "programmer") need focus only on the set of rules in a locale, in a **modular fashion**. Tractability and locality enable a human to "simulate in their 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 experience building applications in personalized intelligent agents.

SUMMARY: see the Abstract.

A Appendix: Review of Logic Programming Concepts

Logic Programs Without Negation; Stratified

We say that an extended logic program without classical negation, but possibly with negation-as-failure, is "*ordinary*". We say that an extended logic program without classical negation and without negation-as-failure is "*monotonic*" or "*Horn*".

(Note that "ordinary" is also known in much of the logic programming literature as "general" to stress its increased generality compared to Horn. We avoid "general" as misleading, and use "ordinary" to stress that the kind of logic programs found in today's implementation, e.g., in the Prolog programming language, are usually in the "ordinary" class.)

The Horn class is especially important because of its tractability: in the propositional case, linear time to answer a query, square 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: `fail` 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 `fail` 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 ordinary 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 `fail` as well:

$$U \models (L_1 \wedge \ldots \wedge L_m \wedge \texttt{fail } L_{m+1} \wedge \ldots \texttt{fail } L_n)$$

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 [Baral and Gelfond, 1994], pages 79 and 83.)

$\mathcal{E}$'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 `fail` in the body of r. s being $-$ denotes that such appearance is within the scope of `fail` ; s being $+$ denotes that such appearance is not within the scope of `fail` . 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 `fail` in the body of r.

$\mathcal{E}$ is said to be **acyclic**, also known 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. "Reverse" here means reversing the direction of the ordering. 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, 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.

Acknowledgements

Thanks to Leora Morgenstern, Stephen Brady, David W. Levine, Vladimir Lifschitz, Chitta Baral, Davis A. Foulger, Michael Maher, Hoi Y. Chan, Gerd Brewka, Joshua S. Auerbach, Jeffrey Kephart, and an anonymous reviewer for helpful discussions and encouragement.

References

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

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

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

[Gelfond and Lifschitz, 1991] 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.

[Gottlob, 1992] Georg Gottlob. Complexity results for nonmonotonic logics. *Working Notes of the Fourth International Workshop on Non-Monotonic Reasoning*, pages 111–125, Technical University of Vienna, Institute for Information Systems, Christian Dppler Laboratory for Expert Systems, Paniglgasse 16, A-1040 Wien, Austria, 1992. Available as a report from the author. Shortened version of a paper with the same title to appear in *The Journal of Logic and Computation*. The Workshop was held at the Hawk Inn, Plymouth, Vermont, May 28–31, 1992. Co-chaired by David Etherington and Henry Kautz of AT&T Bell Labs. Sponsored in part by AT&T and AAAI.

[Grosof and Foulger, 1995] Benjamin N. Grosof and Davis A. Foulger. Globenet and raise: Intelligent agents for networked newsgroups and customer service support. *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. Proceedings available through AAAI on the World Wide Web: http://www.aaai.org . Symposium held at MIT, Cambridge, MA. Paper also available as IBM Research Report RC 20226. IBM T.J. Watson Research Center, Hawthorne, NY 10532. World Wide Web http://www.research.ibm.com .

[Grosof *et al.*, 1995] 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. *Proceedings of the ACM CIKM-95 Workshop on Intelligent Information Agents*, http://www.cs.umbc.edu/iia/, December 1995. Information and on-line proceedings available via the World Wide Web. Workshop held in conjunction with the ACM Conference on Information and Knowledge Management, Dec. 1–2, 1995 in Baltimore, MD. Paper also available as IBM Research Report RC 20305. IBM T.J. Watson Research Center, Hawthorne, NY 10532. World Wide Web http://www.research.ibm.com .

[Grosof, 1991] Benjamin N. Grosof. Generalizing prioritization. *Proceedings of the Second International Conference on Principle of Knowledge Representation and Reasoning*, 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.

[Grosof, 1992] 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 Research Report from Stanford University Computer Science Dept. and/or IBM T.J. Watson Research Center.

[Grosof, 1995] 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.

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

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

[Lifschitz, 1987] Vladimir Lifschitz. On the declarative semantics of logic programs with negation. In Matthew L. Ginsberg, editor, *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, San Francisco, CA, 1987.

[Makinson, 1989] 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.

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

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

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

[Przymusinski, 1988] 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.

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

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

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

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