

Logic Programming with Defaults and Argumentation Theories^{*}

Hui Wan¹, Benjamin Groszof², Michael Kifer¹, Paul Fodor¹, and Senlin Liang¹

¹ State University of New York at Stony Brook, USA

² Vulcan, Inc., USA

Abstract. We define logic programs with defaults and argumentation theories, a new framework that unifies most of the earlier proposals for defeasible reasoning in logic programming. We present a model-theoretic semantics and study its reducibility and well-behavior properties. We use the framework as an elegant and flexible foundation to extend and improve upon Generalized Courteous Logic Programs (GCLP) [19]—one of the popular forms of defeasible reasoning. The extensions include higher-order and object-oriented features of Hilog and F-Logic [7,21]. The improvements include much simpler, incremental reasoning algorithms and more intuitive behavior. The framework and its Courteous family instantiation were implemented as an extension to the FLORA-2 system.

Keywords: Defeasible reasoning, argumentation theory, well-founded models.

1 Introduction

Common-sense reasoning is one of the most important applications of logic programming. Surprisingly, the ability to do such reasoning within logic programming comes from a single, relatively simple device: default negation [8,15,16]. While this parsimony is convenient for theoretical studies, it is a major stumbling block for practical use of logic programming in common-sense reasoning: default negation is too low-level a concept to be safely entrusted to a knowledge engineer who is not a trained logician. Anecdotal evidence suggests that logicians are also not doing much better when it comes to modeling concrete application domains using default negation as a sole tool. These problems have been stimulating search for higher-abstraction forms of logic programming, which equip the knowledge engineer with frameworks and tools that are closer to the way humans tend to think of and describe the various application domains. These frameworks include object-oriented and higher-order concepts [21,7], inheritance and exceptions [32,22,34], and defeasible reasoning [3,5,6,10,11,12,17,19,25,27,31,33,35].

Defeasible reasoning in logic programming (LP) has been successfully used to model a broad range of application domains and tasks, including: policies, regulations, and law; actions, change, and process causality; Web services; aspects

^{*} This work is part of the SILK (Semantic Inference on Large Knowledge) project sponsored by Vulcan, Inc.

of inductive/scientific learning and natural language understanding. However, there has been a bewildering multitude of formal approaches to defeasibility based on a wide variety of intuitions about desired behavior and conceptualization. The difficulties in agreeing on what is the “right” intuition are discussed in [17,6] among others. On top of this, the formal machinery employed by the different approaches is so diverse that there is little hope that more than a tiny subset of the approaches could be directly integrated in a practical, scalable reasoning system. It is also often unclear how to extend most of these approaches to include other popular LP frameworks, such as HiLog [7] and F-logic [21].

The present paper addresses these issues. First, we introduce a new general framework for defeasible reasoning that abstracts the intuitions about defeasibility into what we call *argumentation theories*. Then we develop a simple semantics for this framework and study its properties. The semantics is based on well-founded models [15]; due to space limitations, stable models [16] will be defined in an extended version of this paper. The relationship of this framework to other proposals is discussed in Section 5. The short story is that, based on our analysis, almost all approaches to defeasible reasoning in LP that we are aware of can be simulated in our framework with a suitable choice of an argumentation theory. This makes it possible to use different such theories in one reasoning system. Second, we develop a family of useful argumentation theories one of which captures the essence of *Generalized Courteous Logic Programs* (GCLP) [19]. This formulation provides a foundation to straightforwardly extend GCLP from predicate calculus-based LP to HiLog [7] and F-Logic [21], and also to improve upon GCLP’s behavior and algorithms in several other significant ways, as detailed in Section 4. The usefulness of the HiLog and F-Logic features is well recognized in the literature and industry, e.g., for meta-reasoning; knowledge base translation and integration; modeling of agent’s beliefs, context, and modality; knowledge provenance and navigational meta-data; and Semantic Web data models. Third, we have implemented our framework and several GCLP-style argumentation theories as an extension to FLORA-2.¹ Adding other such theories is straightforward.²

The rest of this paper is organized as follows. Section 2 illustrates GCLP-style defeasible reasoning with an example. Section 3 introduces our framework for defeasible reasoning with argumentation theories. Section 4 presents a family of argumentation theories, which extend and improve GCLP in several significant ways. Section 5 discusses related work, and Section 6 concludes the paper.

2 Motivating Example

The following example illustrates the basic ideas of defeasible reasoning. It uses a stripped-down syntax of FLORA-2 and models part of a game, complete with frame axioms, where blocks are moved from square to square on a board.

¹ <http://flora.sourceforge.net>

² FLORA-2 supports only argumentation theories with the well-founded semantics.

```

// moving blk from->to, if to is free; from becomes free
@move loc(?s+1,?blk,?to) ^ neg loc(?s+1,?blk,?from) :-
    move(?s,?blk,?from,?to), loc(?s,?blk,?from), not loc(?s,?,?to).
@frax1 loc(?s+1,?blk,?pos) :- loc(?s,?blk,?pos). // frame axiom 1
@frax2 neg loc(?s+1,?blk,?pos) :- neg loc(?s,?blk,?pos). // frame axiom 2
@dloc neg loc(?s,?blk,?pos). // each location is free, by default
@state loc(0,block4,square7). // initial state
// no block can be in two places at once
opposes(loc(?s,?blk,?y),loc(?s,?blk,?z)) :- posn(?y), posn(?z), ?y != ?z.
// move-action beats frame axioms; move & init state beats default location
overrides(move,frax1). overrides(move,dloc).
overrides(move,frax2). overrides(frax1,dloc). overrides(state,dloc).
// facts
move(2,block4,square7,square3). // State 2: block4 moves square7->square3
posn(square1). posn(square2). ... .. posn(square16).

```

The example illustrates the Courteous flavor [19] of defeasible reasoning. Here, some rules are labeled (e.g., `@move`, `@frax1`), and the predicate `overrides` specifies that some rules (e.g., the ones labeled `@move`) defeat others (e.g., the ones labeled `@frax1` and `@frax2`). We distinguish between the classical-logic-like *explicit negation*, `neg`, and default negation `not`. Literals L and `neg L` are incompatible and cannot both appear in a consistent model. The predicate `opposes` specifies additional incompatibilities. In our example, `opposes` says that no block can be present in two different positions on the board in the same state.

We can now see how defeasible reasoning works. The rule labeled `@dloc` is a “catch-case” low-priority default that says that all locations on the board are free. Contrary to this default, the fact labeled `@state` says that `block4` is at `square7` in state 0. This “defeats” the `@dloc` rule (due to `overrides(state,dloc)`), so `block4` is indeed at `square7`. Other squares are free unless the “catch-all” default `@dloc` is overridden. Such overriding can occur due to the `@move` rule, which specifies the effects of the move action. The `@move` rule also defeats the frame persistence axioms, `@frax1` and `@frax2`, which simply state that block locations persist from one state to another. Thus, in states 1 and 2 our block is still at `square7` and other squares are free. However, at state 2 a move occurs, and the `@move` rule derives that `block4` must be at `square3` in state 3. Due to the priorities specified via the predicate `overrides`, this latter derivation defeats the derivations produced by the frame axioms and the default location fact `@dloc`.

3 Defeasible Reasoning with Argumentation Theories

Let \mathcal{L} be a logic language with the usual connectives \wedge for conjunction and $:-$ for rule implication; and two negation operators: `neg`, for explicit negation, and `not` for default negation. The alphabet of the language consists of: an infinite set of variables, which are shown in the examples as alphanumeric symbols prefixed with the question mark `?`; and a set of constant symbols, which can appear

as individuals, function symbols, and predicates. Constants will be shown as alphanumeric symbols that are not prefixed with a “?”.

We use the standard notion of *terms* in logic programming. **Atomic formulas**, also called **atoms**, are quite general in form: they can be the atoms used in ordinary logic programming; or the higher-order expressions of HiLog [7]; or the frames of F-logic [21]. A **literal** has one of the following forms:

- An atomic formula.
- $\text{neg } A$, where A is an atomic formula.
- $\text{not } A$, where A is an atom.
- $\text{not neg } A$, where A is an atom.
- $\text{not not } L$ and $\text{neg neg } L$, where L is a literal; these are identified with L .

Let A denote an atom. Literals of the form A or $\text{neg } A$ (or literals that reduce to these forms after elimination of double negation) are called **not-free literals**; literals that reduce to the form $\text{not } A$ are called **not-literals**.

Definition 1. A **plain rule** in a logic language \mathcal{L} is an expression of the form

$$L :- \text{Body} \tag{1}$$

where L , called the **head** of the rule, is a not-free literal in \mathcal{L} , and Body , called the **body** of the rule, is a conjunction of literals in \mathcal{L} .³ As is common in logic programming, we will often write A, B to represent the conjunction $A \wedge B$.

A **labeled rule** in \mathcal{L} is an expression of the form $@r \rho$, where ρ is a plain rule and r is a term, called the **label** of the rule. Thus, labeled rules have the form

$$@r L :- \text{Body} \tag{2}$$

A rule label is not a rule identifier: several rules can have the same label. A **formula** is a literal, a conjunction of literals, or a rule. Given a rule of the form (2), the term

$$\text{handle}(r, L) \tag{3}$$

is called the **handle** for that rule. Here **handle** is a binary function symbol specifically reserved for representing rule handles. However, we do not make further assumptions about this symbol. \square

A **logic program with defaults and argumentation theories** (an **lpda**, for short) in a logic language \mathcal{L} is a set of labeled and plain rules in \mathcal{L} .

In our theory, plain rules are considered to be definite statements about the real world. In contrast, labeled rules are *defeasible defaults*: some (or even all) instances of such rules can be “defeated” by other labeled rules in which case inferences produced by the defeated rules might be “overridden” or “canceled.”

We will be often using variable-free expressions, which we call **ground**. Thus, a **ground term** is a term that contains no variables, a **ground literal** is a variable-free literal, and a **ground rule** is a rule that has no variables.

³ This is easy to generalize to allow Lloyd-Topor extensions [23].

Lpdas are used in conjunction with *argumentation theories*. An argumentation theory is a set of rules that defines conditions under which some rule instances may be defeated or canceled by other rules.

Definition 2. Let \mathcal{L} be a logic language. An **argumentation theory** is a set, AT , of plain rules in \mathcal{L} of the form (1). We also assume that the language \mathcal{L} includes a unary predicate, $\$defeated_{AT}$, which may appear in the heads of some rules in AT .⁴ When confusion does not arise, we will omit the subscript AT .

An *lpda* \mathcal{P} is said to be **compatible** with AT if $\$defeated_{AT}$ does not appear in the rule heads in \mathcal{P} . It is often useful to consider stronger compatibility requirements, which impose additional syntactic restrictions on \mathcal{P} . If C_{AT} is such a compatibility requirement then we will speak of C_{AT} -**compatible** *lpdas*, i.e., *lpdas* that are compatible with AT and satisfy the condition C_{AT} . \square

Thus, an argumentation theory is an ordinary logic program whose rules are not labeled. The rules AT will normally contain other predicates, besides $\$defeated_{AT}$, that are used to specify how the rules in \mathcal{P} get defeated. For instance, the argumentation theories described in Section 4 include the binary predicates **opposes** and **overrides**. In our FLORA-2-based implementation, argumentation theories are *meta-programs*, as in Section 4, encoded using HiLog [7]; we anticipate this would be common for other implementations of *lpdas* as well. For the purpose of defining the semantics, we assume that the argumentation theories are ground. A grounded version of AT with respect to a compatible *lpda* \mathcal{P} is obtained by appropriately instantiating the variables and meta-predicates in AT . For instance, for the theories in Section 4 this means (i) replacing the variables $?R$ with ground rule handles (see Definition 1) followed by (ii) replacing the meta-statement **body**($?R, ?B$), **call**($?B$) in rule (12) with bodies of the rules in \mathcal{P} that have $?R$ as the handle (\mathcal{P} may have more than one rule with the same handle).

Note that the definitions and the subsequent theory permit different subsets of the overall *lpda* to have different argumentation theories AT with different $\$defeated_{AT}$ predicates.⁵

Definition 3. Let \mathcal{P} be an *lpda* and AT an argumentation theory over language \mathcal{L} .

- The **Herbrand Universe** of \mathcal{P} , denoted $\mathcal{U}_{\mathcal{L}}$, is the set of all ground terms built using the constants and function symbols that appear in \mathcal{L} . When confusion does not arise, we will simply write \mathcal{U} .
- The **Herbrand Base** of \mathcal{P} , denoted $\mathcal{B}_{\mathcal{L}}$ (or simply \mathcal{B} , when no ambiguity arises), is the set of all ground **not**-free literals that can be constructed using the predicates in \mathcal{L} . \square

⁴ We say “may” for the sake of generality. If $\$defeated$ does not occur in the head of any rule then the semantics of *lpdas* reduce to ordinary logic programming.

⁵ Our FLORA-2 extension also supports multiple argumentation theories.

3.1 The Well-Founded Semantics

In this section, we extend the well-founded semantics for default negation [15] to *lpdas*. Our development follows the general outline in [29]. The full version of this paper also provides the stable model semantics [16].

The following definition of partial interpretations is essentially from [15,29]. First, we assume that the language includes three special propositional constants, **t**, **f**, and **u**, which stand for *true*, *false*, and *undefined*, respectively. We also assume the existence of the following total order on these propositions: $\mathbf{f} < \mathbf{u} < \mathbf{t}$.

Definition 4. A *partial Herbrand interpretation*, \mathbf{I} , is simply a set of ground literals. In addition: \mathbf{I} must contain both **t** and **not f**; it may contain neither **u** nor **not u**; and $L, \text{not } L$ cannot both be in \mathbf{I} , for any literal L .

An interpretation is *inconsistent relative to an atom* A if both A and $\text{neg } A$ belong to \mathbf{I} . Otherwise, \mathbf{I} is *consistent relative to* A . An interpretation is *consistent* if it is consistent relative to every atom and *inconsistent* if it is inconsistent relative to some atom. An interpretation is *total* if, for every ground **not**-free literal L (except **u**), either L or $\text{not } L$ belongs to \mathbf{I} .

We also define $\mathbf{I}^+ = \{L \mid L \in \mathbf{I} \text{ is a not-free literal}\}$ and $\mathbf{I}^- = \{L \mid L \in \mathbf{I} \text{ is a not-literal}\}$. Thus, $\mathbf{I} = \mathbf{I}^+ \cup \mathbf{I}^-$. \square

Models. Next we define satisfaction for ground formulas and *lpdas*.

Definition 5. Let \mathbf{I} be a partial Herbrand interpretation, L a ground **not**-free literal, and F, G ground formulas. Then \mathbf{I} maps formulas to $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ as follows:

- If L is a **not**-free literal then $\mathbf{I}(L) = \mathbf{t}$ iff $L \in \mathbf{I}$, $\mathbf{I}(L) = \mathbf{f}$ iff $\text{not } L \in \mathbf{I}$, and $\mathbf{I}(L) = \mathbf{u}$, otherwise.
- $\mathbf{I}(\text{not } L) = \sim \mathbf{I}(L)$, where $\sim \mathbf{t} = \mathbf{f}$, $\sim \mathbf{f} = \mathbf{t}$, and $\sim \mathbf{u} = \mathbf{u}$.
Note that the above two items together with Definition 4 imply that $\mathbf{I}(\mathbf{t}) = \mathbf{t}$, $\mathbf{I}(\text{not } \mathbf{t}) = \mathbf{f}$; $\mathbf{I}(\mathbf{f}) = \mathbf{f}$, $\mathbf{I}(\text{not } \mathbf{f}) = \mathbf{t}$; and $\mathbf{I}(\mathbf{u}) = \mathbf{I}(\text{not } \mathbf{u}) = \mathbf{u}$.
- $\mathbf{I}(F \wedge G) = \min(\mathbf{I}(F), \mathbf{I}(G))$.
- For a plain rule $F :- G$, define $\mathbf{I}(F :- G) = \mathbf{t}$ if and only if $\mathbf{I}(F) \geq \mathbf{I}(G)$.
- For a labeled rule $@r F :- G$, we define $\mathbf{I}(@r F :- G) = \mathbf{t}$ if and only if $\mathbf{I}(F) \geq \min(\mathbf{I}(G), \mathbf{I}(\text{not } \$\text{defeated}(\text{handle}(r, F))))$.
Here $\text{handle}(r, F)$ is the handle (Definition 1) for the rule $@r F :- G$. \square

Definition 6. If $\mathbf{I}(F) = \mathbf{t}$, where \mathbf{I} is an interpretation, then we write $\mathbf{I} \models F$ and say that \mathbf{I} is a *model* of F (or that *satisfies* F). An interpretation \mathbf{I} is a model of an *lpda* \mathcal{P} if $\mathbf{I} \models R$ for every $R \in \mathcal{P}$. \square

Definition 7. Given an *lpda* \mathcal{P} , an argumentation theory AT , and an interpretation \mathbf{M} , we say that \mathbf{M} is a model of \mathcal{P} with respect to the argumentation theory AT , written as $\mathbf{M} \models (\mathcal{P}, AT)$, if $\mathbf{M} \models \mathcal{P}$ and $\mathbf{M} \models AT$. \square

Definition 8. Suppose that \mathbf{M}_1 , and \mathbf{M}_2 are interpretations. We define $\mathbf{M}_1 \preceq \mathbf{M}_2$ if $\mathbf{M}_1^+ \subseteq \mathbf{M}_2^+$ and $\mathbf{M}_1^- \supseteq \mathbf{M}_2^-$. The minimal models with respect to \preceq are called the *least* models of (\mathcal{P}, AT) . \square

Well-Founded Models. We now define a special kind of models for *lpdas*, called the well-founded models. These models are first defined as limits of transfinite sequences of partial interpretations and then we show that they correspond to the ordinary well-founded models of certain normal logic programs that are obtained by a transformation from *lpdas*.

The quotient operator is defined analogously to [29], but with changes to adapt this concept to logic programs with defaults and argumentation theories.

Definition 9. Let \mathcal{Q} be a set of rules, which can include labeled as well as plain rules, and let \mathbf{J} be a partial Herbrand interpretation for \mathcal{Q} . We define the *lpda quotient of \mathcal{Q} by \mathbf{J}* , written as $\frac{\mathcal{Q}}{\mathbf{J}}$, by the following sequence of steps:

1. Replace every **not**-literal in the body of \mathcal{Q} by its truth value in \mathbf{J} .
2. Replace every labeled rule of the form $@r L :- \text{Body}$ in \mathcal{Q} , such that $\mathbf{J}(\$defeated(handle(r, L))) = \mathbf{t}$ with the rule $L :- \text{Body}$, **f**.
(Rule handles were introduced in Definition 1.)
3. Replace every labeled rule $(@r L :- \text{Body}) \in \mathcal{Q}$ such that $\mathbf{J}(\$defeated(handle(r, L))) = \mathbf{u}$ with the rule $L :- \text{Body}$, **u**.
4. Remove all labels from the remaining labeled rules.

The resulting set of rules is the *lpda quotient* $\frac{\mathcal{Q}}{\mathbf{J}}$. □

In the next definition, $LPM(Q)$ denotes the least partial model of a **not**-free *lpda* Q . As in [29], $LPM(Q)$ is computed iteratively, by making all possible derivations using the rules in Q starting with the empty partial interpretation.

Definition 10. The *well-founded model* of an *lpda* \mathcal{P} with respect to the argumentation theory AT , written as $WFM(\mathcal{P}, AT)$, is defined as the limit of the following transfinite induction. Initially, \mathbf{I}_0 is the empty set. Suppose \mathbf{I}_m has already been defined for every $m < k$, where k is an ordinal. Then:

- $\mathbf{I}_k = LPM\left(\frac{\mathcal{P} \cup AT}{\mathbf{I}_{k-1}}\right)$, if k is a non-limit ordinal.
- $\mathbf{I}_k = \cup_{i < k} \mathbf{I}_i$, if k is a limit ordinal. □

According to the next theorem, this limit exists. The theorem also shows that *lpdas* reduce to and can be implemented using ordinary logic programming systems that support the well-founded semantics (e.g., XSB).

Theorem 1 (Reduction). The transfinite sequence $\langle \mathbf{I}_0, \mathbf{I}_1, \dots \rangle$ of interpretations in Definition 10 has a (unique) limit. It is reached for some (possibly transfinite) ordinal, α , such that $\mathbf{I}_\alpha = \mathbf{I}_{\alpha+1}$. This limit, $WFM(\mathcal{P}, AT)$, is a least model of (\mathcal{P}, AT) . Furthermore, $WFM(\mathcal{P}, AT)$ coincides with the well-founded model of the ordinary logic program $\mathcal{P}' \cup AT$, where \mathcal{P}' is obtained from \mathcal{P} by changing every labeled rule $(@r L :- \text{Body}) \in \mathcal{P}$ to the plain rule $L :- \text{Body}$, **not** $\$defeated(handle(r, L))$. □

3.2 Well-Behaved Argumentation Theories

So far, argumentation theories were defined in a very general way. However, not all such theories are practically useful. This section introduces a number of *well-behavior properties* that are useful for argumentation theories to abide. These conditions involve both the argumentation theories themselves and their associated compatibility requirements (see Definition 2).

Definition 11. *An argumentation theory AT with a compatibility requirement C_{AT} ensures consistency relative to an atom A if for every C_{AT} -compatible lpda \mathcal{P} , the well-founded model of (\mathcal{P}, AT) is consistent relative to A (see Definition 4). We say that (AT, C_{AT}) ensures consistency, if it ensures consistency for all atoms. \square*

Definition 12. *Consider an argumentation theory AT with a compatibility requirement C_{AT} . Let us further assume that AT uses a binary predicate **opposes**, which is defined on rule handles. Literals L_1 and L_2 are said to **oppose each other** in a partial interpretation \mathbf{I} of an lpda \mathcal{P} iff **opposes**(handle(r_1, L_1), handle(r_2, L_2)) is true in \mathbf{I} for all pairs of rules of the form $@r_1 L_1 :- \dots$ and $@r_2 L_2 :- \dots$ in \mathcal{P} (i.e., rules having L_1 and L_2 in the head).*

We say that AT ensures strong consistency if, for every C_{AT} -compatible lpda \mathcal{P} , the well-founded model \mathbf{M} of (\mathcal{P}, AT) has the following property:

If any pair of literals, L_1 and L_2 , oppose each other in \mathbf{M} , then L_1 and L_2 cannot both be true in \mathbf{M} .

\square

Definition 13. *Consider an argumentation theory AT with a compatibility condition C_{AT} . Let us further assume that AT uses two binary predicates, **overrides** and **opposes**, whose arguments are rule handles. We say that AT has the **overriding property** if, for every C_{AT} -compatible lpda \mathcal{P} , the following rule is true in the well-founded model of (\mathcal{P}, AT) :*

$$\begin{aligned} \text{\$defeated}(\text{handle}(r_2, L_2)) :- & \text{Body}_1 \wedge \text{Body}_2 \\ & \wedge \text{overrides}(\text{handle}(r_1, L_1), \text{handle}(r_2, L_2)) \\ & \wedge \text{opposes}(\text{handle}(r_1, L_1), \text{handle}(r_2, L_2)) \\ & \wedge \text{not } \text{\$defeated}(\text{handle}(r_1, L_1)) \end{aligned} \quad (4)$$

for all pairs of rules of the form $(@r_i L_i :- \text{Body}_i) \in \mathcal{P}$, $i = 1, 2$. \square

Next we develop a family of argumentation theories that obeys these properties.

4 Courteous Argumentation Theories

We now develop a family of particularly interesting argumentation theories, denoted AT^C , which subsumes *generalized courteous logic programs* (GCLP) [19]. Some members of this family correspond to different earlier versions of courteous

logic programs [18,19]; others improve upon these previous versions by eliminating certain cases of controversial behavior. The properties of these argumentation theories are discussed at the end of this section.

Apart from the standard predicate **\$defeated**, the argumentation theories in the \mathcal{AT}^C family use two other predicates: **opposes** and **overrides**, which are normally defined by the user. Argumentation theories might include additional axioms, such as symmetry for **opposes** or transitivity for **overrides**. The **opposes** and **overrides** predicates are expected to be specified over rule handles; they may occur as facts and in the heads of rules. Other predicates in \mathcal{AT}^C represent concepts used to argue that certain rules must or must not be defeated. The variables $?R$ and $?S$ are expected to range over rule handles.

Definition of defeasibility. These rules define what it means for a rule to be defeated or to defeat another rule. A rule is *defeated* if it is *refuted* or *rebutted* by some other rule, provided that the latter rule is not *compromised*. A rule can also be defeated if it is *disqualified* for some other reason.

$$\begin{aligned} \text{\$defeated}(?R) & \text{ :- } \text{\$defeats}(?S, ?R), \text{ not } \text{\$compromised}(?S). \\ \text{\$defeated}(?R) & \text{ :- } \text{\$disqualified}(?R). \\ \text{\$defeats}(?R, ?S) & \text{ :- } \text{\$refutes}(?R, ?S) \text{ or } \text{\$rebutts}(?R, ?S). \end{aligned} \quad (5)$$

The predicates **\$refutes** and **\$rebutts** will be defined shortly. The predicates **\$compromised** and **\$disqualified** can mean different things depending on the intended theory of argumentation. Here are some of the possibilities:

- No rule is compromised or disqualified. *Lpdas* with this argumentation theory are equivalent to the original courteous logic programs (GCLP).

$$\begin{aligned} \text{\$compromised}(?X) & \text{ :- } \text{false}. \\ \text{\$disqualified}(?X) & \text{ :- } \text{false}. \end{aligned} \quad (6)$$

- A rule is compromised if it is defeated, and it is disqualified if it transitively defeats itself.⁶ This choice has been the main one we have experimented with recently for practical use cases using our FLORA-2 extension.

$$\begin{aligned} \text{\$compromised}(?R) & \text{ :- } \text{\$refuted}(?R), \text{\$defeated}(?R). \\ \text{\$disqualified}(?X) & \text{ :- } \text{\$defeats}^*(?X, ?X). \end{aligned} \quad (7)$$

Here **\$defeats*** denotes the transitive closure of **\$defeats**.

- Another reasonable choice is

$$\begin{aligned} \text{\$compromised}(?R) & \text{ :- } \text{\$defeated}(?R). \\ \text{\$disqualified}(?X) & \text{ :- } \text{\$defeats}^*(?X, ?X). \end{aligned} \quad (8)$$

Definitions for \$refutes and \$rebutts. *Refutation* of a rule, r , means that a higher-priority rule implies a conclusion that is incompatible with the conclusion implied by r . It is defined as follows:

$$\begin{aligned} \text{\$refutes}(?R, ?S) & \text{ :- } \text{\$conflict}(?R, ?S), \text{ overrides}(?R, ?S). \\ \text{\$refuted}(?R) & \text{ :- } \text{\$refutes}(?R2, ?R). \end{aligned} \quad (9)$$

⁶ Note that we do not require that the predicate **\$defeats** is transitively closed.

Rebuttal means that a pair of rules assert conflicting conclusions, but neither derivation can be discarded or considered “more important” than the other. This intuition can be expressed in several different (not necessarily equivalent) ways. We have been experimenting with the following definitions (where (11) together with (6) defines the original GCLP):

$$\text{\$rebuts}(?R, ?S) \text{ :- } \text{\$conflict}(?R, ?S), \text{ not } \text{\$compromised}(?R). \quad (10)$$

$$\text{\$rebuts}(?R, ?S) \text{ :- } \text{\$conflict}(?R, ?S), \text{ not } \text{\$compromised}(?R), \quad (11)$$

$$\text{ not } \text{\$refuted}(?R), \text{ not } \text{\$refuted}(?S).$$

Definition of candidacy and conflict. A *candidate* rule-instance is one whose body is true in the knowledge base:

$$\text{\$candidate}(?R) \text{ :- } \text{body}(?R, ?B), \text{ call}(?B). \quad (12)$$

Here **body** is a meta-predicate that binds $?B$ to the body of a rule with handle $?R$. The **call** meta-predicate takes the body of a rule and poses it as a query to the knowledge base. We note that these meta-predicates can be represented as object-level predicates in HiLog [7]. We omit reviewing here the main aspects of HiLog for reasons of space and focus.

Conflicting rules are now defined as follows: two rule handles are in conflict if they are both candidates and are in opposition to each other.

$$\text{\$conflict}(?R, ?S) \text{ :- } \text{\$candidate}(?R), \text{\$candidate}(?S), \text{opposes}(?R, ?S). \quad (13)$$

Background theory for mutual exclusion. The predicate **opposes** is normally defined within the user knowledge base by a set of facts and rules. In addition, our argumentation theories require **opposes** to be symmetric and such that every literal must oppose its explicit negation (**neg**):

$$\text{opposes}(?R, ?S) \text{ :- } \text{opposes}(?S, ?R). \quad (14)$$

$$\text{opposes}(\text{handle}(?L1, ?H), \text{handle}(?L2, \text{neg } ?H)).$$

We say that an argumentation theory *belongs to the AT^C family* if it includes the rules (5), (9), and (12)–(14); plus either (6) or (7) or (8); and either (10) or (11). Let AT be an argumentation theory in AT^C and let the compatibility requirement be as follows. An *lpda* \mathcal{P} is compatible with AT iff:

- The set of the atoms that appear in the heads of plain (non-defeasible) rules and in the heads of labeled (defeasible) rules in \mathcal{P} are disjoint.
- The $\text{\$}$ -predicates defined by AT^C (**\\$defeated**, **\\$compromised**, **\\$refuted**, etc.) do not occur in the heads of the program rules (i.e., they are defined only by the rules in AT).

Theorem 2 (Well-behavior). *Let AT be an argumentation theory in AT^C with the above compatibility requirement. Then AT satisfies the properties of well-behaved theories of Section 3.2; namely:*

1. *AT* ensures consistency for the atoms that occur in the heads of labeled rules.
2. Suppose there is no literal A for which both A and $\text{neg } A$ appear in the heads of plain rules in \mathcal{P} . Then *AT* ensures consistency (for all atoms).
3. If $\text{opposes}(\text{handle}(\dots, L_1), \text{handle}(\dots, L_2))$ is true only when neither L_1 nor L_2 occurs in the heads of plain rules, then *AT* ensures strong consistency.
4. *AT* has the overriding property. \square

Consider an argumentation theory, denoted AT^{GCLP} , that consists of the rules (5) – (6), (9), (11) – (14) and the following compatibility requirement. An *lpda* \mathcal{P} is compatible with AT^{GCLP} iff:

- \mathcal{P} contains only labeled rules.
- The $\$$ -predicates defined by AT^{GCLP} ($\$$ defeated, $\$$ refuted, etc.) do not occur in the heads of the program rules.

Theorem 3 (GCLP as LPDA). *Consider AT^{GCLP} and a compatible *lpda* \mathcal{P} . Let \mathcal{P}' be the program obtained from \mathcal{P} using the GCLP transformation of [19].⁷ Then the restrictions of the well-founded models of (\mathcal{P}, AT^{GCLP}) and of \mathcal{P}' to the predicates mentioned in \mathcal{P} coincide. \square*

This result says that the original GCLP is essentially equivalent to LPDA with the argumentation theory AT^{GCLP} . The new formulation of GCLP has many benefits. First, it is not limited to ordinary logic programs: it extends straightforwardly to HiLog [7], F-logic [21], and other forms of logic programming. Second, *lpdas* are inherently incremental: adding new knowledge does not require changes to the already existing knowledge. In contrast, in the original approach, adding new rules or facts meant that the GCLP transformation had to be re-applied from scratch. It was substantial effort to find an equivalent incremental transformation [13]. Third, the new formulation generalizes GCLP by allowing non-defeasible rules.

Also, importantly, the new framework lets us use different argumentation theories, while the original approach had one or two built into fairly complex transformations, often making it hard to see through the complexity and to experiment. In contrast, the new approach separates the argumentation theory from program transformation, makes it much easier to see the rationale behind the different parts of the argumentation theories, greatly simplifies the implementation, and enables various optimizations and improvements.

A case in point is the following example of controversial behavior exhibited by the original GCLP in an “edge case.”

```
@a p.           @b q.           @c s.
overrides(handle(a,?), handle(c,?)). opposes(handle(? ,p), handle(? ,s)).
overrides(handle(c,?), handle(b,?)). opposes(handle(? ,q), handle(? ,s)).
```

Here GCLP sanctions the model $\{\mathbf{p}, \text{not } \mathbf{q}, \text{not } \mathbf{s}\}$. However, one might feel that the intended model should instead be $\{\mathbf{p}, \mathbf{q}, \text{not } \mathbf{s}\}$ because c is defeated

⁷ We are glossing over the minor detail that the syntax of *lpdas* is slightly different from the syntax of GCLP used in [19].

and thus should not defeat **b**. Modifying the argumentation theory AT^{GCLP} is much easier than examining and modifying the complex transformation of the original GCLP. The alternative intuition about desired defeasibility behavior in the above “edge case” example can be expressed by replacing the rules (6) with (7) in the argumentation theory AT^{GCLP} .

As further illustration, we show how GCLP can be used in conjunction with HiLog’s higher-order syntax and F-logic’s object-oriented syntax. The example also illustrates the use of rule labels with variables.

```
@perm(?t) ?p(?usr) :- ?adm[states(?t)->?p(?usr)], ?adm[controls->?p].
overrides(handle(perm(?t1),?),handle(perm(?t2),?)) :- ?t1 > ?t2.
Bob[states(2008)->neg print(A1)]. Bob[states(2009)->print(A1)].
Bob[controls->{print(?), negprint(?)}].
```

Here the first rule says that if an administrator, `?adm`, has stated at time `?t` that the user `?usr` has a privilege, `?p`, and if that administrator controls this type of privileges, then the privilege is granted. Privileges can be positive (e.g., `print`) or negative (e.g., `neg print`). This rule is defeasible and its label is non-ground. The head of the rule is a HiLog literal, because of the higher-order variable `?p`, while the body has a combination of F-logic and HiLog features. The second rule is non-defeasible. It says that later pronouncements override earlier ones. The facts on line 3 say that the administrator `Bob` has issued conflicting statements about whether the user `A1` is allowed to print or not. The last fact says that `Bob` controls the printing privilege as well as its revocation. With the AT^{GCLP} argumentation theory, the above *lpda* sanctions the conclusion `print(A1)`, as expected. It is worth pointing out here that modifying the original GCLP transformation [19] to handle this kind of programs is not a trivial matter.

5 Comparison with Other Work

The last two decades saw a great number of approaches to defeasible reasoning in logic programming. Most of these are based on Reiter’s Default Logic [30], stable models [16], and only a few [19,24] use the well-founded semantics [15]. None of the works surveyed here uses the notion of argumentation theories, but [17,10,12] have goals similar to ours. Due to the sheer size of the literature on defeasible reasoning, it is not feasible to do justice to all prior work in this section. Therefore, we will focus on the more closely related work and refer the reader to a recent survey [9] for a broader discussion of the literature, including the works that we were unable to mention.

General frameworks [17,10,12]. The closest, in spirit, to our work are the logic of prioritized defaults by Gelfond and Son [17], the meta-interpretation approach of [12], and ordered logic programs of Delgrande, Schaub, and Tompits [10].

The logic of prioritized defaults [17] does not use the notion of argumentation theories, but it is made clear that the meaning of the various theories of defaults may differ from one application domain to another. This is analogous to allowing argumentation theories to vary. However, Gelfond and Son developed their

language as a meta-theory, whose semantics is given by meta-interpreters. What we call an “argumentation theory” is built into meta-interpreters in [17], and no independent model theory is given. In contrast, our approach distills all the differences between the different default theories to the notion of an argumentation theory with a simple interface to the user-provided domain description, the predicate `$defeated`. This allows us to define model-theoretic semantics, including the well-founded and stable models, to unify the theories of Courteous Logic Programming, Defeasible Logic, Prioritized Defaults, and more. This also allows us to focus on the development of powerful argumentation theories, which have the expected behavior on all known “benchmarks” that we are aware of from the literature. The following is one such example.

```
@d1 neg flies :- penguin.      overrides(d1,d2).      bird.
@d2 flies :- bird.            overrides(d2,d3).      swims.
@d3 penguin :- bird, swims.   overrides(d1,d3).
```

This example was discussed in [5,17] as a case where a seemingly correct domain description yields the unintended model `{swims, bird, penguin, flies}` instead of the expected model `{swims, bird, penguin, negflies}`. Gelfond and Son [17] argue that this happens because the above domain description is “unclear” and requires a clarification in the form of the statement `opposes(d2,d3)`. In our opinion, however, requiring such additional domain-specific particulars is undue engineering burden. Like [5], we believe that the above domain description is sufficient by itself and, together with any of the argumentation theories of Section 4, this *lpda* has the expected behavior in our framework.

Like Gelfond and Son’s work, Leone et. al. [12] set out to unify approaches to defeasible reasoning. Specifically, they present an adaptable meta-interpreter, which can be made to simulate the approaches described in [6,33] among others.

Delgrande et. al. [10] propose a framework of ordered logic programming, which can use a variety of preference handling strategies. For each strategy, this approach devises a transformation from ordered logic programs to ordinary logic programs. Each transformation is custom-made for the particular preference-handling strategy, and the approach was illustrated by showing transformations for several strategies, including two described in earlier works [33,12].

Unlike our approach, Delgrande’s framework does not come with a unifying model-theoretic semantics. Instead, the definition of preferred answer sets differs from one preference-handling strategy to another. One of the more important conceptual differences between our work and [10] has to do with the nature of the variable parts of the two approaches. In our case, the variable part is the argumentation theory, which is a set of definitions for concepts that a human reasoner might use to argue why certain conclusions are to be defeated. In case of [10], the variable part is the transformation, which encodes a fairly low-level mechanism: the order of rule applications required to generate the preferred answer set.⁸ Finally, we note that each program transformation in [10] needs a

⁸ Note that argumentation theories can also encode rule application orderings.

compiler that contains hundreds of lines of Prolog code. Our approach requires no new software, and each argumentation theory typically contains 20-30 rules.

Defeasible Logic [25]. Defeasible Logic is related to *lpdas* in a number of ways. On one hand, [1] shows that a **not**-free subset of GCLP (which is a special case of *lpdas*) can be represented as a defeasible logic theory. On the other hand, it can be shown that Defeasible Logic programs with non-contradictory strict rules can be represented as *lpdas* with suitable argumentation theories both under the well-founded semantics [24] and under the stable model semantics [2].⁹ Apart from the ability to choose argumentation theories, *lpdas* generalize Defeasible Logic in other ways. For instance, Defeasible Logic does not deal with general conflicts, i.e., situations where the opposing rules have heads that are not negations of each other. In addition, *lpdas* can use the full power of rules to define the prioritization ordering, while Defeasible Logic requires that this ordering is specified in advance.

Other logics of prioritized defaults. Many other formalizations of prioritized defaults, including [3,5,6,11,31,33,34,35], have been developed over the years. In these formalisms, priorities can be assigned either to atoms (e.g., [31]) or to rules ([3,5,6] and others), and the details vary widely. For example, most proposals specify priorities explicitly, but some (e.g., [11]) assign them implicitly, via the notion of specificity (rule r_1 is more specific than r_2 if the body of r_1 entails the body of r_2). For yet others, the mechanism for implicit prioritization is instead derived from the structure of class hierarchies [34]. In most cases, these proposals use stable models instead of the well-founded models used in the present work. However, as mentioned earlier, stable models for *lpdas* can be defined and, based on our analysis, most of the above approaches can be simulated within our framework by choosing suitable argumentation theories. Only Sakama and Inoue's approach [31] bucks the trend. The key difficulty in capturing this formalism is the way in which it defines preferences over answer sets: in [31], preferences are derived from a priority relation over atoms, while all other approaches define priorities over rules only.

Argumentation theories. A significant body of work is dedicated to development of argumentation theories. These include papers like [4,14,26], which use this term in a different sense than we do and are not closely related,¹⁰ as well as more closely related works [27,28,20]. The focus of the latter works is development of the actual concepts that argumentation theories operate with. For instance, [27] uses Default Logic [30] to formalize the notions of defeat, defensible arguments, etc. Our work is more general in the sense that we do not stick to a particular argumentation theory and our FLORA-2-based implementation makes it easy

⁹ Stable models for *lpdas* will be defined in the full version of this paper.

¹⁰ By *arguments* these works mean proofs or sets of supporting statements, not rules that define the notion of defeasibility. The focus of [4] is non-monotonic logic in general, while [14] is a procedural approach to defeasible rules. It is unclear whether this approach can be captured as an argumentation theory in our framework.

to use different such theories. However, concrete argumentation theories used in our framework might embody notions analogous to those in [27]. For instance, the theory of Section 4 uses the notions of defeated and defensible (*rebutted*, in our terminology) arguments, although those notions are not exactly the ones developed in [27].

6 Conclusions

We presented a novel approach that unifies most of the earlier work on defeasible reasoning in logic programming (LP). The primary advantages of the approach are:

- Generalization of Courteous and other previous defeasible LP approaches to include HiLog-style higher-order and F-logic style object-oriented features.
- Much simpler implementation for Courteous and other previous defeasible LP approaches. Such an implementation is easy in a system with sufficient degree of introspection, like FLORA-2: contrast 20-30 rules per argumentation theory (e.g., Section 4) versus thousands of lines of code (e.g., [19,10]).
- Unification of almost all previous defeasible LP approaches within one theory and the ability to combine multiple such in one system.
- Improvements on original GCLP, including a direct model theory, simpler and faster incremental updating, and better control over edge case behavior.

References

1. Antoniou, G., Billington, D., Governatori, G., Maher, M.J.: Representation results for defeasible logic. *ACM Trans. Comput. Log.* 2(2), 255–287 (2001)
2. Antoniou, G., Maher, M.J.: Embedding defeasible logic into logic programs. In: *Int'l Conference on Logic Programming*, pp. 393–404 (2002)
3. Baader, F., Hollunder, B.: Priorities on defaults with prerequisites, and their application in treating specificity in terminological default logic. *Journal of Automated Reasoning* 15(1), 41–68 (1995)
4. Bondarenko, A., Dung, P.M., Kowalski, R.A., Toni, F.: An abstract, argumentation-theoretic approach to default reasoning. *AI* 93(1-2), 63–101 (1997)
5. Brewka, G., Eiter, T.: Preferred answer sets for extended logic programs. *Artificial Intelligence* 109, 297–356 (1999)
6. Brewka, G., Eiter, T.: Prioritizing default logic. In: *Intellectics and Computational Logic – Papers in Honour of Wolfgang Bibel*, pp. 27–45. Kluwer Academic Publishers, Dordrecht (2000)
7. Chen, W., Kifer, M., Warren, D.S.: HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming* 15(3), 187–230 (1993)
8. Clark, K.L.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) *Logic and Data Bases*, pp. 292–322. Plenum Press, New York (1978)
9. Delgrande, J., Schaub, T., Tompits, H., Wang, K.: A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence* 20(12), 308–334 (2004)

10. Delgrande, J.P., Schaub, T., Tompits, H.: A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming* 2, 129–187 (2003)
11. Dung, P.M., Son, T.C.: An argument-based approach to reasoning with specificity. *Artificial Intelligence* 133(1-2), 35–85 (2001)
12. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing preferred answer sets by meta-interpretation in answer set programming. *Theory and Practice of Logic Programming* 3(4), 463–498 (2003)
13. Ganjugunte, S.: Extending reasoning infrastructure for rules on the semantic web: Well-founded negation, incremental courteous logic programs, and interoperability tools in sweetrules. Master's thesis, UMBC (2005)
14. García, A.J., Simari, G.R.: Defeasible logic programming: an argumentative approach. *Theory Practice of Logic Programming* 4(2), 95–138 (2004)
15. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. *Journal of the ACM* 38, 620–650 (1991)
16. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: *Proceedings of ICLP/SLP*, pp. 1070–1080. MIT Press, Cambridge (1988)
17. Gelfond, M., Son, T.C.: Reasoning with prioritized defaults. In: Dix, J., Moniz Pereira, L., Przymusinski, T.C. (eds.) *LPKR 1997*. LNCS, vol. 1471, pp. 164–223. Springer, Heidelberg (1998)
18. Groszof, B.N.: Prioritized conflict handling for logic programs. In: *Int'l Logic Programming Symposium*, October 1997, pp. 197–211 (1997)
19. Groszof, B.N.: A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report Supplementary Update Follow-On to RC 21472, IBM (July 1999)
20. Karacapilidis, N.I., Papadias, D., Gordon, T.F.: An argumentation based framework for defeasible and qualitative reasoning. In: *XIIIth Brazilian Symposium on Artificial Intelligence, Advances in Artificial Intelligence*, pp. 1–10. Springer, Heidelberg (1996)
21. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. *Journal of ACM* 42, 741–843 (1995)
22. Lakshmanan, L.V.S., Thirunarayan, K.: Declarative frameworks for inheritance. In: Chomicki, J., Saake, G. (eds.) *Logics for Databases and Information Systems*, pp. 357–388. Kluwer Academic Publishers, Dordrecht (1998)
23. Lloyd, J.W.: *Foundations of Logic Programming*, 2nd edn. Springer, Heidelberg (1987)
24. Maier, F., Nute, D.: Relating defeasible logic to the well-founded semantics for normal logic programs. In: *Int'l Workshop on Non-monotonic Reasoning* (2006)
25. Nute, D.: Defeasible logic. In: *Handbook of logic in artificial intelligence and logic programming*, pp. 353–395. Oxford University Press, Oxford (1994)
26. Pereira, L.M., Pinto, A.M.: Reductio ad absurdum argumentation in normal logic programs. In: *ArgNMR workshop at LPNMR*, pp. 96–113 (2007)
27. Prakken, H.: An argumentation framework in default logic. *Annals of Mathematics and Artificial Intelligence* 9(1-2), 93–132 (1993)
28. Prakken, H.: A logical framework for modelling legal argument. In: *ICAIL 1993: 4th Int'l Conf. on Artificial Intelligence and Law*, pp. 1–9. ACM, New York (1993)
29. Przymusinski, T.C.: Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence* 12, 141–187 (1994)
30. Reiter, R.: A logic for default reasoning. *Artificial Intelligence* 13, 81–132 (1980)
31. Sakama, C., Inoue, K.: Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123(1-2), 185–222 (2000)

32. Touretzky, D.S.: *The Mathematics of Inheritance Systems*. Morgan Kaufmann, San Francisco (1986)
33. Wang, K., Zhou, L., Lin, F.: Alternating fixpoint theory for logic programs with priority. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) *CL 2000*. LNCS, vol. 1861, pp. 164–178. Springer, Heidelberg (2000)
34. Yang, G., Kifer, M.: Inheritance in rule-based frame systems: Semantics and inference. *Journal on Data Semantics* 2800, 69–97 (2003)
35. Zhang, Y., Wu, C.M., Bai, Y.: Implementing prioritized logic programming. *AI Communications* 14(4), 183–196 (2001)