Defeasibility in Answer Set Programs via Argumentation Theories*

Hui Wan[1] Michael Kifer[1] Benjamin Grosof[2]

[1] State University of New York at Stony Brook, USA
[2] Vulcan, Inc., USA

Abstract. Defeasible reasoning has been studied extensively in the last two decades and many different and dissimilar approaches are currently on the table. This multitude of ideas has made the field hard to navigate and the different techniques hard to compare. Our earlier work on Logic Programming with Defaults and Argumentation Theories (LPDA) introduced a degree of unification into the approaches that rely on the well-founded semantics. The present work takes this idea further and introduces ASPDA—a unifying framework for defeasibility of *disjunctive* logic programs under the Answer Set Programming (ASP). Since the well-founded and the answer set semantics underlie almost all existing approaches to defeasible reasoning in Logic Programming, LPDA and ASPDA together capture most of those approaches. In addition to ASPDA, we obtained a number of interesting and non-trivial results. First, we show that ASPDA is reducible to ordinary ASP programs, albeit at the cost of exponential blowup in the number of rules. Second, we study reducibility of ASPDA to the non-disjunctive case and show that head-cycle-free ASPDA programs reduce to the non-disjunctive case—similarly to head-cycle-free ASP programs, but through a more complex transformation. The blowup in the program size is linear in this case.

1 Introduction

Defeasible reasoning is a form of non-monotonic reasoning where logical axioms are true "by default" but their truth status may be undercut or even negated by other, conflicting axioms. This type of reasoning has been an important application of logic programming. It was successfully used to model policies, regulations, and law; actions, change, and process causality; Web services; aspects of inductive/scientific learning and natural language understanding. However, there is a bewildering multitude of dissimilar and incompatible approaches to defeasibility based on a wide variety of intuitions and techniques. The difficulties in relating and comparing the different approaches have been discussed in [13,3] and other works. Combining the various theories of defeasible reasoning with other advances in logic-based knowledge representation, such as HiLog [4] and F-logic [16], has also been a problem.

* This work is part of the SILK (Semantic Inference on Large Knowledge) project sponsored by Vulcan, Inc. It was also partially supported by the NSF grant 0964196.

Our earlier work [20] addressed some of these issues by introducing a general framework for defeasible reasoning, called LPDA, which abstracts the intuitions about defeasibility into what we call *argumentation theories*. This enabled a uniform syntax and semantics for a wide variety of defeasible theories, which could be used in harmony and simultaneously in the same knowledge base. LPDA, as defined in [20], was developed on the basis of the well-founded models [9] and was able to unify a number of approaches to defeasible reasoning that are based on the well-founded semantics. However, a large number of works on defeasible reasoning are based on the stable model semantics [11], which has very different properties and is not capturable by well-founded models. Furthermore, defeasible reasoning in the presence of disjunctive information, which to the best of our knowledge has not been considered hitherto, appears to require even more general semantics, the answer set semantics [10].

The present work takes the idea of LPDA further and introduces ASPDA—an analogous framework for defeasibility of *disjunctive* logic rules through argumentation theories based on Answer Set Programming (ASP). In this way, LPDA and ASPDA together unify and extend most of the existing theories of defeasible reasoning in Logic Programming.

Extension of the semantics of LPDA to ASP with head-disjunctions turned out to be elegant but not straightforward. The relationship between ASPDA and the regular ASP also proved to be non-obvious. First, we show that ASPDA can be expressed by regular ASP programs, albeit at the cost of exponential blowup in the number of rules. Then we study the class of *head-cycle-free* programs with disjunctive heads and show that a related notion exists for ASPDA. By analogy with the classical case, such programs can be reduced to non-disjunctive programs under the defeasible stable model semantics, although the transformation is more complicated than in the case of the regular ASP. The blowup in the program size is still linear, however.

The rest of this paper is organized as follows. Section 2 illustrates defeasible reasoning under the answer-set semantics using the well-known Turkey Shoot example [19]. Section 3 defines the syntax and semantics of defeasible disjunctive logic programs and presents a number of interesting results about reducibility to the regular logic programming and to the non-disjunctive case. Section 4 gives two examples of argumentation theories for ASPDA. One is an adaptation of GCLP [14,20] to ASPDA, a theory that is used in all examples throughout this paper. Another is an argumentation theory that captures Defeasible Logic [1]. Although Defeasible Logic (as all other theories of defeasible reasoning up until now) does not support head-disjuncts in the rules, it is an apt illustration of ASPDA as a unifying framework that is capable of capturing much of the prior work. Sections 5 and 6 discuss related work and conclude the paper.

2 Motivating Example

The following example is adapted from the Texas Turkey Shoot game example in [19]. We use the usual syntax of logic programming with the only difference that rules are tagged with `@tag` symbols and disjunctions are allowed. Variables are prefixed with the symbol `?`. Initially one of the guns is known to be loaded, but it is not known which. The objective is to find a plan to kill the turkey by shooting one or both guns assuming that the shooter can observe the effects of his actions. Let `g1` and `g2` be the constants representing the guns. Numerals are used in the example to represent time points, and the initial time point is assumed to be 1. For instance, `shoot(g1,1)` and `shoot(g1,2)` represent the actions of shooting the gun `g1` at time points 1 and 2.

```
@kpld    loaded(?Gun,?Time+1) :- loaded(?Gun,?Time).    // Frame axiom 1.
@kpunld  neg loaded(?Gun,?T+1) :- neg loaded(?Gun,?T). // Frame axiom 2.
@dd      neg alive(?Time+1) :- neg alive(?Time).       // Frame axiom 3.
@liv     alive(?Time+1) :- alive(?Time).                // Frame axiom 4.
// A gun becomes unloaded after being fired
@sht1    neg loaded(?Gun,?Time+1) :- shoot(?Gun,?Time).
// The turkey becomes dead after a loaded gun is fired at it
@sht2    neg alive(?Time+1) :- shoot(?Gun,?Time), loaded(?Gun,?Time).
//  Axioms for the initial state
         alive(1).                                 // The turkey is alive initially
@unld    neg loaded(g1,1) ∨ neg loaded(g2,1). // One gun is unloaded initially
@ld      loaded(g1,1) ∨ loaded(g2,1).         // One gun is loaded initially
         shoot(g1,1).                         // Fire g1 at time 1
         // If g1 is unloaded at time 1, fire g2 at time 2.
         shoot(g2,2) :- not loaded(g1,1).
// axioms for contradiction and rule priorities
#opposes(alive(?Time), neg alive(?Time)).
#overrides(sht1, kpld).
#overrides(sht2, liv).
```

In the above specification, some of the rules have tags, e.g., `kpld` and `sht1`, and the predicate `#overrides` specifies priorities among some of these tagged rules. We distinguish between the classical-logic-like *explicit negation* `neg` and the *default negation* `not` (which in this paper will have the answer-set semantics). Literals L and `neg` L are assumed to be incompatible and cannot both appear in a consistent model. The predicate `#opposes` specifies additional contradictions, such as the inability for the turkey to be both dead and alive at the same time.

We can now explain how defeasible reasoning works in the above game. The rule labeled `kpld` is a frame persistence axiom stating that a loaded gun stays loaded unless some other action explicitly changes this state of affairs. The rule `sht1` states that if a gun is fired then it becomes unloaded in the next state. This rule has a higher priority than the frame axiom `kpld` due to the axiom `#overrides`(sht1,kpld). The rule labeled `liv` is another frame axiom stating that a live turkey remains alive by default. This rule is defeated by the higher-priority rule labeled `sht2`, which says that if a loaded gun is fired at the turkey, then the turkey is dead in the next state. Note that our program has disjunctions in the heads of the rules labeled `unld` and `ld`), so the initial state of the game is uncertain. The problem is to infer that by firing one or both guns in succession the shooter can kill the turkey despite the uncertainty in the initial state. Note that due to the disjunctions, the other existing logic programming approaches to defeasible reasoning cannot handle the above situation, and this is precisely the motivation for our current work. We will return to this example at the end of Section 4.1 after the necessary theory is developed.

3 Defeasible Reasoning with Argumentation Theories

In this section we introduce the syntax and semantics of disjunctive logic programming where defeasibility is controlled by argumentation theories. The main syntactic difference is that rules now have *tags*, and the main semantic difference is that these rules can be *defeated*.

Let $\mathcal{L}$ be a logic language with the usual connectives $\wedge$ for conjunction, $\vee$ for disjunction, 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 assume that the language includes two special propositional constants, $\mathbf{t}$ and $\mathbf{f}$, which stand for *true* and *false*, respectively. We also assume the following order on these propositions: $\mathbf{f} < \mathbf{t}$.

We use the standard notion of *terms* in logic programming. ***Atomic formulas***, also called ***atoms***, can be quite general in form: they can be the usual atoms used in ordinary logic programming; or the higher-order expressions of HiLog [4]; or the frames of F-logic [16]. A ***literal*** has one of the following forms:

– An atomic formula.
– `neg` A and `not` A, where A is an atomic formula.
– `not neg` A, where A is an atomic formula.
– `not not` L and `neg neg` L, where L is a literal; these are identified with L.

For convenience, the literals `not not` L and `neg neg` L will be identified with L. Let A denote an atom. Literals of the form A or `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 `not` A are called **`not`-literals**.

Definition 1 (Tagged rule). *A **tagged rule** in a logic language $\mathcal{L}$ is an expression of the form*

$$@r\ L_1 \vee ... \vee L_k \texttt{ :- } Body \tag{1}$$

*where r is a term, called the **tag** of the rule; L_1, ..., L_k ($k \geq 0$) are literals in $\mathcal{L}$, called the **head literals** of the rule; 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 rule tag is not a rule identifier: several rules can have the same tag.*

*A **constraint** is a special form of rule where $\mathbf{f}$ is a single head literal. We will usually omit $\mathbf{f}$ in such rules.*

[3] This is easy to generalize to allow Lloyd-Topor extensions [18].

A ***formula*** *is a literal, a Boolean combination of literals using conjunction and disjunction, or a rule.* □

We will often omit showing rule tags when they are immaterial.

Definition 2 (Rule handle). *Given a rule of the form (1), the terms of the form*

$$\texttt{handle}(r, L_i), \text{ where } i = 1, ..., k$$

are called the ***handles*** *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.* □

Definition 3 (Ground terms and rules). *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.* □

Definition 4 (ASPDA). *An* ***answer-set program with defaults and argumentation theories*** *(an* `aspda`*, for short) in a logic language $\mathcal{L}$ is a set of tagged rules in $\mathcal{L}$, which can be* ***strict*** *or* ***defeasible****. Sets or rules that do not have disjunctions in the head will be called non-disjunctive* `aspda`*s.* □

Strict rules are used as *definite* statements about the world. In contrast, defeasible rules represent *defeasible defaults* whose instances can be "defeated" by other rules. Inferences produced by the defeated rules are "overridden."

We assume that the distinction between strict and defeasible rules is specified in some way: either syntactically or by means of a predicate. For instance, in Section 4, we use the predicate `#strict` for that purpose.

*Aspda*s are used in conjunction with *argumentation theories*, which are sets of rules that defines conditions under which some rule instances may be defeated by other rules.

Definition 5 (Argumentation theory). *Let $\mathcal{L}$ be a logic language. An* ***argumentation theory*** *is a set, AT, of* strict *rules in $\mathcal{L}$ of the form (1). We also assume that the language $\mathcal{L}$ includes a unary predicate, $\texttt{\$defeated}_{AT}$, which may appear in the heads of some rules in AT.*[4] *When confusion does not arise, we will omit the subscript AT.*
An `aspda` *$\mathcal{P}$ is said to be* ***compatible*** *with AT if $\texttt{\$defeated}_{AT}$ does not appear in the rule heads in $\mathcal{P}$.* □

In argumentation theory all rules are strict, by definition.[5] The rules in AT will normally contain other predicates, besides $\texttt{\$defeated}_{AT}$, that are used to specify how the rules in $\mathcal{P}$ get defeated.

[4] If `$defeated` does not occur in the head of any rule then the semantics of *aspda*s reduce to ordinary logic programming.

[5] In principle, we could allow argumentation theories to be defeasible, but we will not do so in this paper.

Usually argumentation theories employ the concepts of rule priority and contradictions among facts. Priorities are often specified via a predicate, such as `#overrides`, which tell that some rules (or rule instances) have higher priorities than other rules (e.g., `#overrides`$(rule\_tag1, rule\_tag2)$). Contradictions are commonly expressed via predicates such as `#opposes`, which tell that certain facts cannot be true together (e.g., `#opposes`$(price(ball1, 20), price(ball1, 30))$). The `$defeated` predicate is then defined in terms of these and other predicates. In this paper, we adopt the convention that the predicates defined only by argumentation theories will be prefixed with the \$-sign, the predicates used and/or defined both by the argumentation theories and user programs will be prefixed with the #-sign, and the predicates defined only by user programs will not be marked in any special way: they will be denoted by alphanumeric symbols.

In defining the semantics, we assume that the argumentation theories are ground. A grounded version of AT with respect to a compatible ***aspda*** $\mathcal{P}$ is obtained by appropriately instantiating the variables and meta-predicates.

Note that the theory developed here permits different subsets of the overall ***aspda*** to have different argumentation theories AT with different `$defeated`$_{AT}$ predicates. For instance, our implementation of the logic programming framework with argumentation theories for the well-founded semantics in an extended version of FLORA-2 [15] supports multiple argumentation theories.

3.1 Interpretations and Models

Definition 6 (Herbrand universe). *Let $\mathcal{P}$ be an* `aspda` *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 $\mathcal{L}$.* □

Definition 7 (Herbrand interpretation). *A **Herbrand interpretation**, $\boldsymbol{I}$, is a subset of $\mathcal{B}$. It is simply a set of ground* `not` *-free literals. In addition, $\boldsymbol{I}$ must contain* **t** *and it must not contain* **f**.
*An interpretation is **inconsistent relative to** an atom A if both A and* `neg` *A are in $\boldsymbol{I}$. Otherwise, $\boldsymbol{I}$ 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.* □

Note that all interpretations considered in this paper are Herbrand, so we will often neglect to mention Herbrandness explicitly.

Next we introduce the notion of satisfaction of defeasible rules and strict rules by interpretations.

Definition 8 (Truth valuation). *Let $\boldsymbol{I}$ be a Herbrand interpretation, L a ground* `not` *-free literal, and let F, G be ground formulas. We define **truth valuations** that map formulas to* {**t**,**f**} *as follows:*

- $\boldsymbol{I}(L) = \mathbf{t}$ *iff* $L \in \boldsymbol{I}$, $\boldsymbol{I}(L) = \mathbf{f}$ *iff otherwise.*
- $\boldsymbol{I}(\texttt{not}\, L) = \sim \boldsymbol{I}(L)$, *where* $\sim \mathbf{t} = \mathbf{f}$ *and* $\sim \mathbf{f} = \mathbf{t}$.
- $\boldsymbol{I}(F \wedge G) = \texttt{min}(\boldsymbol{I}(F), \boldsymbol{I}(G))$. *Recall that* $\mathbf{f} < \mathbf{t}$.
- $\boldsymbol{I}(F \vee G) = \texttt{max}(\boldsymbol{I}(F), \boldsymbol{I}(G))$.
- *For a strict rule* $@r\ F \texttt{ :- } G$, *we define* $\boldsymbol{I}(F \texttt{ :- } G) = \mathbf{t}$ *if and only if* $\boldsymbol{I}(F) \geq \boldsymbol{I}(G)$.
- *For a defeasible rule* $@r\ F \texttt{ :- } G$, *we define* $\boldsymbol{I}(@r\ F \texttt{ :- } G) = \mathbf{t}$ *if and only if* $\boldsymbol{I}(F) \geq \texttt{min}(\boldsymbol{I}(G), \boldsymbol{I}(\texttt{not}\,\texttt{\$defeated}(\texttt{handle}(r, F))))$.
  Here $\texttt{handle}(r, F)$ *is the handle for the rule* $@r\ F \texttt{ :- } G$ *(Definition 2).* □

Definition 9 (Model of formula). *If F is a ground formula, $\boldsymbol{I}$ an interpretation, and $\boldsymbol{I}(F) = \mathbf{t}$, then we write $\boldsymbol{I} \models F$ and say that $\boldsymbol{I}$ is a **model** of F or that F is **satisfied** in $\boldsymbol{I}$. An interpretation $\boldsymbol{I}$ is a model of an* `aspda` *$\mathcal{P}$ if all the rules in $\mathcal{P}$ are satisfied in $\boldsymbol{I}$, i.e., if $\boldsymbol{I} \models R$ for every $R \in \mathcal{P}$.* □

Definition 10 (Model of ASPDA). *Given an* `aspda` *$\mathcal{P}$, an argumentation theory AT, and an interpretation $\boldsymbol{M}$, we say that $\boldsymbol{M}$ is a model of $\mathcal{P}$ with respect to the argumentation theory AT (or a model of $(\mathcal{P}, AT)$, for short), written as $\boldsymbol{M} \models (\mathcal{P}, AT)$, if $\boldsymbol{M} \models \mathcal{P}$ and $\boldsymbol{M} \models AT$.* □

Definition 11 (Minimal model). *An interpretation $\boldsymbol{M}$ is a minimal model of $(\mathcal{P}, AT)$ iff $\boldsymbol{M}$ is a model of $(\mathcal{P}, AT)$ and no proper subset of $\boldsymbol{M}$ is a model of $(\mathcal{P}, AT)$.* □

3.2 Stable Model and Answer-set Semantics

In this section, we extend the stable model semantics [11] and the answer-set semantics [10] to ASPDA. We start with non-disjunctive *aspda*s and stable models.

Definition 12 (ASPDA quotient, non-disjunctive case). *Let $\mathcal{Q}$ be a non-disjunctive* `aspda`*, and let $\boldsymbol{J}$ be a Herbrand interpretation for $\mathcal{Q}$. The **ASPDA quotient of $\mathcal{Q}$ by $\boldsymbol{J}$**, written as $\dfrac{\mathcal{Q}}{\boldsymbol{J}}$, is defined by the following sequence of steps:*

1. *Delete every rule $R \in \mathcal{Q}$ such that there is a* `not` *-literal of the form* $\texttt{not}\, A$ *in R's body and $A \in \boldsymbol{J}$;*
2. *Delete every defeasible rule of the form $@r\ L \texttt{ :- } Body$ in $\mathcal{Q}$ such that* $\texttt{\$defeated}(\texttt{handle}(r, L)) \in \boldsymbol{J}$.
3. *Remove all* `not` *-literals from the remaining rules.*
4. *Remove all tags from the remaining tagged rules.* □

When dealing with stable models, it is often assumed that interpretations are consistent [10]. All the definitions and results in this section extend to this case straightforwardly.

Recall that the *minimality* of Herbrand models is defined in Definition 11.

Definition 13 (Stable model). *A Herbrand interpretation $\boldsymbol{M}$ is a **stable model** of a non-disjunctive* `aspda` *$\mathcal{P}$ with respect to the argumentation theory AT, if $\boldsymbol{M}$ is a minimal Herbrand model of $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$.* □

The next theorem shows that non-disjunctive ***aspda***s can be implemented using ordinary logic programming systems that support the stable model semantics (e.g., DLV [17]).

Theorem 1 (Reduction for the stable model semantics). *Let $\mathcal{P}$ be a non-disjunctive* `aspda` *and AT an argumentation theory. Then the following two sets coincide:*

– *The set of stable models of $\mathcal{P}$ with respect to AT.*
– *The set of stable models of the ordinary logic program $\mathcal{P}' \cup AT'$, where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by converting every defeasible rule* (`@r L :- Body`) $\in \mathcal{P}$ *into the plain rule of the form* `L :- Body, not $defeated(handle(r,L))` *and removing all the remaining tags; and AT' is obtained from AT by simply removing all the tags.* □

For disjunctive rules, stable models are called *answer sets* and we will now generalize the above semantics to such rules. In generalizing ***aspda***s to disjunctive rules, the main problem is to define handles for disjunctive rules, to define quotients, and to find an analog of the reduction theorem.

Example 1. Consider a disjunctive program

```
@r1 a ∨ b ∨ c.
@r2 d ∨ e.
```

The ordinary stable models of this program are {`a`, `d`}, {`a`, `e`}, {`b`, `d`}, {`b`, `e`}, {`c`, `d`}, and {`c`, `e`}. Suppose now that `a` cannot be true when either `d` or `e` holds, and that `b`, `e` are also incompatible. We express this with the following facts:

```
#opposes(a,d).      #opposes(a,e).      #opposes(b,e).
```

Suppose, in addition, that rule `r1` has a higher priority than `r2`, i.e.,

```
#overrides(r1,r2).
```

Intuitively, {`a`, `d`}, {`a`, `e`}, and {`b`, `e`} can no longer be models due to the incompatibility statements above, while the models {`b`, `d`}, {`c`, `d`}, and {`c`, `e`} are still intuitively fine. At the same time, one might feel that {`a`} should be viewed as a suitable model because `r1` overrides `r2`, `a` makes `r1` true, and `a` is incompatible with both heads of the rule `r2`.

As it turns out, {`a`} may or may not be a defeasible stable model—it all depends on the associated argumentation theory. It would be a stable model of our ***aspda*** if the argumentation theory had the following rule instances:

```
$defeated(handle(r2,d)) :- #overrides(r1,r2), #opposes(a,d), a.
$defeated(handle(r2,e)) :- #overrides(r1,r2), #opposes(a,e), a.
```
□

The following definitions generalize Definition 12 to disjunctive ***aspda***s and make the intuition behind Example 1 precise.

Definition 14 (ASPDA quotient, disjunctive case). *Let $\mathcal{Q}$ be a disjunctive* `aspda`*, and let $\boldsymbol{J}$ be a Herbrand interpretation for $\mathcal{Q}$. We define the ASPDA* ***quotient of** $\mathcal{Q}$ **by** $\boldsymbol{J}$, written as $\dfrac{\mathcal{Q}}{\boldsymbol{J}}$, by the following sequence of steps:*

1. *Delete every rule $R \in \mathcal{Q}$ such that there is a* `not` *-literal of the form* `not` A *in R's body and $A \in \boldsymbol{J}$;*
2. *For every defeasible rule of the form* $@\mathtt{r}\ \mathtt{L_1} \vee ... \vee \mathtt{L_n}$ `:- Body` *in $\mathcal{Q}$, delete every $\mathtt{L_i}$ such that* $\mathtt{\$defeated(handle(r, L_i))} \in \boldsymbol{J}$*. If all the $\mathtt{L_i}$'s are deleted, delete the entire rule.*
3. *Remove all* `not` *-literals from the remaining rules.*
4. *Remove all tags from the remaining tagged rules.* □

Definition 13 carries over in a natural way:

Definition 15 (Answer set). *A Herbrand interpretation $\boldsymbol{M}$ is an* ***answer set*** *of a disjunctive* `aspda` *$\mathcal{P}$ with respect to the argumentation theory AT, if $\boldsymbol{M}$ is a minimal Herbrand model of $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$.* □

The analog of Theorem 1 is as follows.

Theorem 2 (Reduction for the answer-set semantics). *Let $\mathcal{P}$ be a (disjunctive)* `aspda` *and AT an argumentation theory. Then the following two sets coincide:*

- *The set of answer sets for the* `aspda` *$\mathcal{P}$ with respect to AT.*
- *The set of answer sets for the ordinary logic program $\mathcal{P}' \cup AT'$, where $\mathcal{P}'$ is obtained from $\mathcal{P}$ by converting every defeasible rule* $(@\mathtt{r}\ \mathtt{L_1} \vee ... \vee \mathtt{L_n} \ \mathtt{:\text{-}\ Body}) \in \mathcal{P}$ *into a collection of plain rules of the form*

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i} \ \mathtt{:\text{-}\ Body} \wedge \wedge_{i \in K} \ \mathtt{not\ \$defeated(handle(r, L_i))}$$
$$\wedge \wedge_{j \in N-K} \ \mathtt{\$defeated(handle(r, L_j))}.$$

  for each subset $K \subseteq N = \{1, ..., n\}$ and removing all the remaining tags; and AT' is obtained from AT by simply removing all the tags. □

With the above definitions, it can now be verified that the answer sets for the ***aspda*** in Example 1 are precisely as described there.

3.3 Reduction to the Non-disjunctive Case

In ordinary answer-set programming, certain disjunctive rules can be reduced to the non-disjunctive case via the so-called *shifting* transformation. For example, this transformation would replace the rule $\mathtt{p} \vee \mathtt{q} \vee \mathtt{s}$ `:- body` with the rules

```
p :- body, not q, not s.
q :- body, not p, not s.
s :- body, not q, not p.
```

Ben-Eliyahu and Dechter [2] have shown that this is an *equivalence* transformation for so called *head-cycle free* programs.[6] We reproduce that definition below adjusting it for disjunctive ***aspda***s.

[6] The works [7,12] developed similar shifting techniques.

Definition 16. *[2] The **dependency graph** $G_{\mathcal{P}}$, of an* `aspda` *$\mathcal{P}$, is a directed graph where nodes are literals. An edge from L to L' goes iff there is a rule in which L appears positive in the body and L' is a head literal. An* `aspda` *is **head-cycle free (HCF)** iff its dependency graph does not contain directed cycles that connect literals that belong to the head of the same rule.* □

An interesting question is whether an analogous shifting transformation and an equivalence result holds for disjunctive ***aspda***s.

Definition 17. *Let $\mathcal{P}$ be a disjunctive* `aspda`. *The (ordinary) **shifting** of $\mathcal{P}$, written as $shift(\mathcal{P})$, is a non-disjunctive* `aspda` *obtained from $\mathcal{P}$ by replacing each defeasible rule of the form* $(@\mathtt{r}\ \mathtt{L}_1 \vee ... \vee \mathtt{L}_n \ \mathtt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* n *new defeasible rules*

$$\begin{array}{lll} @\mathtt{r}\ \mathtt{L}_1 & \mathtt{:-} & \mathtt{Body} \wedge \mathtt{not}\, \mathtt{L}_2 \wedge ... \wedge \mathtt{not}\, \mathtt{L}_n \\ ... & ... & ... \\ @\mathtt{r}\ \mathtt{L}_n & \mathtt{:-} & \mathtt{Body} \wedge \mathtt{not}\, \mathtt{L}_1 \wedge ... \wedge \mathtt{not}\, \mathtt{L}_{n-1} \end{array}$$ □

Surprisingly, it turns out that $shift(\mathcal{P})$ is *not* equivalent to $\mathcal{P}$ even for HCF ***aspda***s. To see this, consider the following rule set, which we will denote $\mathcal{P}_1$

```
@r1  a ∨ b ∨ c.              @r2  d.             @r3  c.
```

Suppose that the associated argumentation theory implies `$defeated(handle(r1, c))` and does not imply any other `$defeated` facts that involve the above rules. Then $\mathcal{P}_1$ would have the following answer sets: $\{\mathtt{a}, \mathtt{d}, \mathtt{c}\}$ and $\{\mathtt{b}, \mathtt{d}, \mathtt{c}\}$. In contrast, the ordinary shifting transformation yields the non-disjunctive ***aspda*** $shift(\mathcal{P}_1)$

```
@r1  a :-  not b ∧ not c.            @r2  d.
@r1  b :-  not a ∧ not c.            @r3  c.
@r1  c :-  not a ∧ not b.
```

which has only one answer set: $\{\mathtt{d}, \mathtt{c}\}$ with respect to the argumentation theory.

It turns out, however, that a result similar to Ben-Eliyahu and Dechter's holds for disjunctive ***aspda***s, but for a slightly different shifting transformation.

Definition 18. *The **ASPDA shifting** of an* `aspda` *$\mathcal{P}$, written as* `aspda`*_$shift(\mathcal{P})$, is a non-disjunctive* `aspda` *obtained from $\mathcal{P}$ by replacing each defeasible rule of the form* $(@\mathtt{r}\ \mathtt{L}_1 \vee ... \vee \mathtt{L}_n \ \mathtt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* n *new defeasible rules of the form*

$$\begin{array}{lll} @\mathtt{r}\ \mathtt{L}_1 & \mathtt{:-}\ \mathtt{Body} & \wedge\big(\mathtt{not}\, \mathtt{L}_2 \vee \mathtt{\$defeated(handle(r, L_2))}\big) \\ & & \wedge \ \ ... \\ & & \wedge\big(\mathtt{not}\, \mathtt{L}_n \vee \mathtt{\$defeated(handle(r, L_2))}\big) \\ ... & ... & ... \\ @\mathtt{r}\ \mathtt{L}_n & \mathtt{:-}\ \mathtt{Body} & \wedge\big(\mathtt{not}\, \mathtt{L}_1 \vee \mathtt{\$defeated(handle(r, L_1))}\big) \\ & & \wedge \ \ ... \\ & & \wedge\big(\mathtt{not}\, \mathtt{L}_{n-1} \vee \mathtt{\$defeated(handle(r, L_{n-1}))}\big) \end{array} \quad (2)$$

Theorem 3. *Let $\mathcal{P}$ be an HCF* `aspda` *and let AT be an argumentation theory. Then S is an answer set of $\mathcal{P}$ with respect to AT iff S is an answer set of* `aspda`*_$shift(\mathcal{P})$ with respect to AT.* □

Corollary 1. *Let $\mathcal{P}$ be an HCF* `aspda`*. Let AT be an argumentation theory such that, for each literal L, whenever* `$defeated(handle(`r, L`))` *is true for some rule tag r,* `$defeated(handle(`r', L`))` *is true for every tag r' such that there is a rule with the tag r' and L as a head-literal. Then S is an answer set of* `aspda_`$shift(\mathcal{P})$ *with respect to AT iff S is an answer set of $shift(\mathcal{P})$. In other words,* `aspda_`$shift(\mathcal{P})$ *is equivalent to $shift(\mathcal{P})$ with respect to AT.*

4 Examples of Argumentation Theories

4.1 A-GCLP [14,20]

Our first example is an ASPDA counterpart for the argumentation theory proposed in [20], which captures generalized courteous logic programs [14] (under the well-founded semantics). We will call this theory A-GCLP and will denote it by AT^{AGCLP}. It is this argumentation theory that was used in all the earlier examples in this paper.

In AT^{AGCLP}, the predicate `$defeated`, which plays a key role in the semantics of *aspdas*, is defined in terms of the predicates `#opposes` and `#overrides`. These predicates are defined by the knowledge engineer within the knowledge base via sets of facts and rules. The argumentation theory only imposes some constraints on `#opposes`.

The `$defeated` predicate is now defined as follows. A rule handle is *defeated* if it is *refuted* by some other not defeated rule or if it transitively defeats itself.

```
$defeated(?R) :- $defeats(?S, ?R).
$defeated(?R) :- $trans_defeats(?R, ?R).
```

The auxiliary predicates used above are defined as follows:

```
$defeats(?R, ?S) :- $refutes(?R, ?S), not $defeated(?R), not #strict(?S).
$trans_defeats(?X, ?Y) :- $defeats(?X, ?Y).
$trans_defeats(?X, ?Y) :- $defeats(?X, ?Z), $trans_defeats(?Z, ?Y).
```

The predicate `#strict` is used here to distinguish strict rules from the defeasible ones. The predicate `$refutes` indicates when one rule handle refutes another. *Refutation* of a rule handle, `r`, means that a higher-priority rule implies a conclusion that is incompatible with the conclusion implied by the rule with the handle `r`. This is defined as follows:

```
$refutes(?R, ?S) :-
        $conflict(?R, ?S), #overrides(?R, ?S), ?R = handle(?T, ?L), ?L.
```

The definition of the concept of a conflict between two rules, represented by the predicate `$conflict` above, relies in turn on the notion of a candidate. A *candidate* rule-instance is one whose body is true in the knowledge base:

```
$candidate(?R) :- body(?R, ?B), ?B.
```

Here the meta-predicates **body** binds $?B$ to the body of a rule with handle $?R$.

Conflicting rules are now defined as follows: two rule handles are in conflict if they are both candidates and the literals in them are incompatible:

$$\begin{array}{l}\texttt{\$conflict}(?R1, ?R2)\ \texttt{:-} \\ \qquad ?R1 = \texttt{handle}(?T1, L1),\ ?R2 = \texttt{handle}(?T2, L2), \\ \qquad \texttt{\$candidate}(?R1),\ \texttt{\$candidate}(?R2),\ \texttt{\#opposes}(?L1, ?L2).\end{array} \tag{3}$$

Finally, the argumentation theory provides the following self-explanatory background axioms for `#opposes`, and the axiom of preference for strict rules:

$$\begin{array}{l}\texttt{\#opposes}(?L1, ?L2)\ \texttt{:-}\ \texttt{\#opposes}(?L2, ?L1). \\ \texttt{\#opposes}(?L,\ \texttt{neg}\ ?L). \\ \ \texttt{:-}\ ?L1,\ ?L2,\ \texttt{\#opposes}(?L1, ?L2). \\ \texttt{\#overrides}(?R, ?S)\ \texttt{:-}\ \texttt{\#strict}(?R),\ \texttt{not}\,\texttt{\#strict}(?S).\end{array}$$

With this argumentation theory, we can now come back to the turkey-shoot example in Section 2 and to Example 1. It can be verified that the turkey-shoot example has two answer sets. In one, $\{\texttt{neg loaded}(\texttt{g1}, \texttt{1}),\ \texttt{loaded}(\texttt{g2}, \texttt{1}),\ \texttt{neg}\,\texttt{alive}(\texttt{3})\}$ is true and in another $\{\texttt{loaded}(\texttt{g1}, \texttt{1}),\ \texttt{neg loaded}(\texttt{g2}, \texttt{1}),\ \texttt{neg}\,\texttt{alive}(\texttt{3})\}$. This shows that the sequence of actions in the example produces the expected result and AT^{AGCLP} allows us to reason by cases.

As to Example 1, one can verify that this ***aspda*** has four answer sets: $\{\texttt{a}\}, \{\texttt{b}, \texttt{d}\}, \{\texttt{c}, \texttt{d}\}, \{\texttt{c}, \texttt{f}\}$, as claimed.

4.2 Defeasible Logic [1]

As yet another example, this section develops an argumentation theory that captures the reasoning in Defeasible Logic of [1].

Defeasible Logic partitions all rules into strict, defeasible, and defeaters. The defeater rules are used only to defeat other rules, but they themselves do not produce any inferences. In our terms, this means that defeater rules are *defeated* defeasible rules whose only purpose is to block inferences produced by other rules. Strict and defeater rules are specified via the predicates `#strict` and `#defeater`. Other key restrictions in that logic are that it does not support disjunctions in the rule heads; opposition among literals is limited to p and $\texttt{neg}\,p$, for each p; does not use default negation, i.e., all literals in that logic are `not`-free; and the rule tags are also rule identifiers. This implies that each tag uniquely determines the rule head and body and this restriction lets us simplify the argumentation theory by omitting `handle` from most literals.

With this, we can now formulate the argumentation theory for Defeasible Logic, denoted AT^{DL}, as follows.

$$\begin{array}{l}\texttt{\$defeated}(\texttt{handle}(?T, ?L))\ \texttt{:-}\ \texttt{\#defeated\_aux}(?T).\ \ \textit{// no handles in } AT^{DL} \\ \texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\$conflict}(?T, ?S),\ \textbf{head}(?S, ?L),\ \texttt{\$definitely}(?L). \\ \texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\#defeater}(?T). \qquad \textit{// defeaters make no inferences} \\ \texttt{\#defeated\_aux}(?T)\ \texttt{:-}\ \texttt{\$overruled}(?T).\end{array}$$

Here **head** is a meta-predicate that binds $?L$ to the head of a rule with Id $?S$.

The predicate `$definitely` is defined as follows:

$$
\begin{array}{l}
\texttt{\$definitely}(?L)\ \texttt{:-} \\
\qquad\qquad \texttt{\#strict}(?T),\ \mathbf{head}(?T,?L),\ \mathbf{body}(?T,?B),\ \mathbf{each\_definite}(?B).
\end{array}
$$

As before, **body** is a meta-predicate that binds $?B$ to the body of a rule with tag $?T$; **each_definite**$(?B)$ is a meta predicate that is true when `$definitely`$(?B)$ is true or when $?B$ is bound to a conjunction, *conj*, and `$definitely`(c) is true for every conjunct $c \in conj$.

It remains to define `$overruled`, which relies on the notion of candidacy and conflict, like in AT^{AGCLP}. The predicate `$candidate` is defined as in (3) except that the handles are dropped and only the rule tags are retained.

$$
\begin{array}{rl}
\texttt{\$overruled}(?T)\ \texttt{:-} & \texttt{\$conflict}(?T,?S),\ \texttt{\$candidate}(?S),\ \texttt{not}\,\texttt{\$refuted}(?S). \\
\texttt{\$refuted}(?S)\ \texttt{:-} & \texttt{\$conflict}(?T,?S),\ \texttt{\$candidate}(?T), \\
 & \texttt{\#overrides}(?T,?S),\ \texttt{not}\,\texttt{\#defeater}(?T). \\
\texttt{\$conflict}(?T,?S)\ \texttt{:-} & \mathbf{head}(?T,?L),\ \mathbf{head}(?S,\texttt{neg}\,?L), \\
 & \texttt{\$candidate}(?T),\ \texttt{\$candidate}(?S).
\end{array}
$$

5 Comparison with Other Work

Although a great deal of work has been devoted to various theories of defeasible reasoning, only a few dealt with unifying frameworks for such reasoning. The notable exceptions are the works [13,5,8], which had goals similar to ours. Due to the large volume of literature on defeasible reasoning, we will focus on the above works, which are related to our work most closely. We refer the reader to a recent survey [6] for a discussion of the various individual theories of defeasibility.

The logic of prioritized defaults [13] does not use the notion of argumentation theories, but it allows for multiple theories of defaults for different application domains. This is analogous to allowing argumentation theories to vary. However, defaults are defined via meta-theories and the semantics in [13] is given by meta-interpretation. What we call an "argumentation theory" is implicit in the meta-interpreters, and no independent model theory is given. In contrast, our approach abstracts all the differences between the different theories for defaults to the notion of an argumentation theory with a simple interface to the user-provided domain description, the predicate `$defeated`. Our approach is model-theoretic and it covers both the well-founded semantics [20] and answer sets (this work). It unifies the theories of Courteous Logic Programming, Defeasible Logic, Prioritized Defaults, and more.

Delgrande et. al. [5] 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 [21,8].

Unlike ASPDA, 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 [5] 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 [5], 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.[7] It is also important to note that each program transformation in [5] needs a compiler that contains hundreds of lines of Prolog code, while our approach requires no new software, and each argumentation theory typically contains 20-30 rules.

Leone et. al. [8] 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 [3,21] among others. However, this framework lacks a model theoretic semantics and is not as flexible as ASPDA.

Finally, to the best of our knowledge, the present paper is the only work that studies the semantics of defeasibility for *disjunctive* logic programs.

6 Conclusions

This paper developed a novel theory of defeasible disjunctive logic programming under the answer-set semantics. It is a companion to our earlier work which developed a general theory of defaults and defeasibility through argumentation theories and was based on the well-founded semantics. Apart from the model theoretic semantics, we have shown that head-cycle free disjunctive defeasible programs can be reduced to non-disjunctive ones, which mirrors an analogous result for non-defeasible disjunctive rules with default negation. To illustrate the power of the proposed framework, we have given two examples of argumentation theories. One is an adaptation for stable models of the generalized courteous argumentation theory, which was presented in [20] for well-founded models. This theory was used in all the examples in this paper. The other argumentation theory was intended to show how *ASPDA* can capture other approaches to defeasible reasoning; in this case the defeasible logic of [1].

References

1. G. Antoniou, D. Billington, G. Governatori, and M. J. Maher. Embedding defeasible logic into logic programming. *Theory and Practice of Logic Programming (TPLP)*, 6(6):703–735, 2006.
2. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994.
3. G. Brewka and T. Eiter. Prioritizing default logic. In *Intellectics and Computational Logic – Papers in Honour of Wolfgang Bibel*, pages 27–45. Kluwer Academic Publishers, 2000.

[7] Note that argumentation theories can also encode rule application orderings.

4. W. Chen, M. Kifer, and D. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.
5. J. Delgrande, T. Schaub, and H. Tompits. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming*, 2:129–187, 2003.
6. J. Delgrande, T. Schaub, H. Tompits, and K. Wang. A classification and survey of preference handling approaches in nonmonotonic reasoning. *Computational Intelligence*, 20(12):308–334, 2004.
7. J. Dix, G. Gottlob, and V. Marek. Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae*, XXVIII(1/2):87–100, 1996.
8. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Computing preferred answer sets by meta-interpretation in answer set programming. *Theory and Practice of Logic Programming*, 3(4):463–498, 2003.
9. A. V. Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.
10. M. Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, , and B. Porter, editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.
11. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP*, pages 1070–1080. MIT Press, 1988.
12. M. Gelfond, H. Przymusinska, V. Lifschitz, and M. Truszczynski. Disjunctive defaults. In *Proceedings of the International Conference on Knowledge Representation and Reasoning*, pages 230–237, 1991.
13. M. Gelfond and T. Son. Reasoning with prioritized defaults. In *Third International Workshop on Logic Programming and Knowledge Representation*, volume 1471 of *Lecture Notes in Computer Science*, pages 164–223. Springer, 1997.
14. B. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report Supplementary Update Follow-On to RC 21472, IBM, July 1999.
15. M. Kifer. FLORA-2: An object-oriented knowledge base language. The FLORA-2 Web Site. http://flora.sourceforge.net.
16. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.
17. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.
18. J. Lloyd. *Foundations of Logic Programming (Second Edition)*. Springer, 1987.
19. A. R. Morales, P. H. Tu, and T. C. Son. An extension to conformant planning using logic programming. In *IJCAI*, pages 1991–1996, 2007.
20. H. Wan, B. Grosof, M. Kifer, P. Fodor, and S. Liang. Logic programming with defaults and argumentation theories. In *ICLP*, pages 432–448, 2009.
21. K. Wang, L. Zhou, and F. Lin. Alternating fixpoint theory for logic programs with priority. In *First Int'l Conference on Computational Logic (CL'00)*, number 1861 in Lecture Notes in Computer Science, pages 164–178. Springer, 2000.