Defeasibility in Answer Set Programs with Defaults and Argumentation Rules [1]

Hui Wan [a] Michael Kifer [b] Benjamin Grosof [c]
[a] *IBM T.J Watson Research Center, USA*
E-mail: hwan@us.ibm.com
[b] *Stony Brook University, USA*
E-mail: kifer@cs.stonybrook.edu
[c] *Vulcan Inc., USA*
E-mail: BenjaminG@vulcan.com

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 (Answer Set Programs via Argumentation Rules) — 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 can closely approximate 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. 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. We also shed light on the relationship between ASPDA and some of the earlier theories such as Defeasible Logic and LPDA.

Keywords: Logic Programming, Defeasible Reasoning, Argumentation Theory, Argumentation Rules, Answer Sets, Stable Model

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 applied to model policies, regulations, and law; actions, change, and process causality; Web services; and aspects of inductive/scientific learning [38,37,33,23,2,25,26]. 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 [22,8,12,40] among others. Combining the various theories of defeasible reasoning with other advances in logic-based knowledge representation, such as HiLog [10] and F-logic [29], has also been a problem.

Our earlier work [40] 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*. In LPDA, an argumentation theory is a set of logic axioms that express the arguments for or against defeating various rules in the knowledge base. These arguments often depend on the particular appli-

[1]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.

cation domain and user intent. An argumentation theory should be viewed not as part of a knowledge base but rather of its semantics. This approach enables 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 [40], was developed on the basis of the well-founded models [18] 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 [20], which has very different properties and is not capturable by well-founded models. Furthermore, general defeasible reasoning in the presence of disjunctive information appears to require even more general semantics, the answer set semantics [19].

The present work takes the idea of LPDA further and introduces ASPDA—an analogous framework for defeasibility of *disjunctive* logic rules through argumentation rules [1] 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. A polynomial reduction has been recently given in [16]. 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.

To avoid possible confusion, we should mention from the outset that argumentation rules are related to argumentation theories of Dung et al. [14]. We briefly discuss the relationship in Section 5.

A preliminary report on this work appeared in [41]. Compared to that earlier paper, the present paper develops the main concepts to a fuller extent, provides all proofs, and includes extensive examples of instantiations of ASPDA to illustrate the inner workings of the ASPDA framework.

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 [32]. 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 rulesets for ASPDA. One is an adaptation of GCLP [24,40] to ASPDA, a theory that is used in all examples throughout this paper. Another is an argumentation ruleset that closely simulates Defeasible Logic [1]. Sections 5 and 6 discuss related work and conclude the paper.

2. Motivating Example

The example in Figure 1 is adapted from the Texas Turkey Shoot game example in [32]. We use the usual syntax of logic programming with the only difference that rules are tagged with `@tag` symbols and head-disjunctions are allowed. Variables are prefixed with the symbol "`?`".

In the scenario described in the example, one of the guns is known to be loaded initially, 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. In the example, 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* naf (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 example. The rule tagged with `kpld` is a frame persistence axiom stating that a loaded gun stays loaded unless some other action ex-

[1] Earlier [40] we used the term "argumentation theories" but renamed it to avoid possible confusion with a similar and related term used in Dung et al. [14].

Fig. 1. Turkey-shoot example

```
@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) :- naf loaded(g1,1).
// axioms for contradiction and rule priorities
#opposes(alive(?Time), neg alive(?Time)).
#overrides(sht1, kpld).
#overrides(sht2, liv).
```

plicitly 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 that has the tag `liv` is another frame axiom stating that a live turkey remains alive by default. This rule is defeated by the higher-priority rule tagged with `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, 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 in Section 4.3 after the necessary theory is developed.

3. Defeasible Reasoning with Argumentation Rules

In this section we introduce the syntax and semantics of disjunctive logic programming where defeasibility is controlled by argumentation rules —sets of axioms (or *arguments*) that say when and why any particular rule should be considered as defeated and the inference it sanctions as null and void. The main syntactic difference from non-defeasible disjunctive logic programming 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 naf 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, **t** and **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 [10]; or the frames of F-logic [29]. A ***literal*** has one of the following forms:

- An atomic formula.
- neg A and naf A, where A is an atomic formula.
- naf neg A, where A is an atomic formula.
- naf naf L and neg neg L, where L is a literal; these are identified with L.

Let A denote an atom. Literals of the form A or $\mathsf{neg}\, A$ (or literals that reduce to these forms after elimination of double negation) are called **naf-free literals**; literals that reduce to the form $\mathsf{naf}\, A$ are called **naf-literals**.

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

$$@r\ L_1 \vee ... \vee L_k\ \mathbf{:-}\ Body \quad (1)$$

*where r is a term, called the **tag** of the rule; L_1, ..., L_k ($k \geq 0$) are naf-free 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 $\mathcal{L}$.[2] 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.[3]*

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

*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 (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 3 (ASPDA) *An **answer-set program with defaults and argumentation rules** (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. Sometimes we will omit tags when they are immaterial.* □

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 rules*, which are sets of rules that define conditions under which some rule instances may be defeated by other rules.

Definition 4 (Argumentation ruleset) *Let $\mathcal{L}$ be a logic language. An **argumentation ruleset** is a set, AT, of strict rules in $\mathcal{L}$ of the form (1). We also assume that the language $\mathcal{L}$ includes a binary predicate, $\$\mathbf{defeated}_{AT}$, 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 $\$\mathbf{defeated}_{AT}$ does not appear in the rule heads in $\mathcal{P}$.* □

In an argumentation ruleset all rules are strict, by definition.[5] The rules in *AT* will normally contain other predicates, besides $\$\mathbf{defeated}_{AT}$, that are used to specify how the rules in $\mathcal{P}$ get defeated. We will see full-fledged examples of argumentation rulesets in Section 4. Note that an argumentation ruleset is also an *`aspda`*.

Usually argumentation rules employ the concepts of rule priority and contradictions among facts. Priorities are often specified via predicates, 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**$(price(ball, 20), price(ball, 30))$). The **$defeated** predicate is then defined in terms of **#overrides**, **#opposes**, and other predicates. In this paper, we adopt the convention that the predicates defined only by argumentation rules will be prefixed with the \$-sign, such as \$conflict, and will be in normal font, except **$defeated**, which will be typeset in bold. The predicates used and/or defined both by the argumentation rules and user programs will be prefixed with the #-sign and will be in bold font. Meta-predicates, such as **body**, will also be set in bold. The predicates defined and used only by user programs

[2] This is easy to generalize to allow Lloyd-Topor extensions [31].
[3] This makes it easier to specify priorities and conflicts among *groups* of rules as opposed to individual rules, as in Figure 3 (look for the tags `move` and `frame`).

[4] If **$defeated** does not occur in the head of any rule then the semantics of *`aspda`s* trivially reduce to ordinary logic programming.
[5] In principle, we could allow argumentation rules to be defeasible, but we will not do so in this paper.

will be denoted by alphanumeric symbols and will not be marked in any special way.

In defining the semantics, we assume that the argumentation rules 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 use different sets argumentation rules *AT* with different $\texttt{\$defeated}_{AT}$ predicates. For instance, our implementation of the logic programming framework with argumentation rules for the well-founded semantics in an extended version of FLORA-2 [28] supports multiple argumentation rulesets.

3.1. Interpretations and Models

Definition 5 (Herbrand universe and Herbrand base) *Let $\mathcal{P}$ be an* aspda *and AT an argumentation ruleset 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 $\mathcal{U}$, omitting the language subscript.*
- *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* naf *-free literals that can be constructed using the predicates in $\mathcal{L}$.* □

Definition 6 (Herbrand interpretation) *A **Herbrand interpretation**, $\boldsymbol{I}$, is a subset of $\mathcal{B}$, i.e., a set of ground* naf *-free literals. In addition, $\boldsymbol{I}$ must contain* **t** *and 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 7 (Truth valuation) *Let $\boldsymbol{I}$ be a Herbrand interpretation, L a ground* naf *-free literal, and let F, G be ground formulas. We define **truth valuations** that map formulas to* $\{\mathbf{t},\mathbf{f}\}$ *as follows:*

- $\boldsymbol{I}(L) = \mathbf{t}$ *if* $L \in \boldsymbol{I}$, $\boldsymbol{I}(L) = \mathbf{f}$ *otherwise.*
- $\boldsymbol{I}(\mathsf{naf}\, L) = \sim \boldsymbol{I}(L)$, *where* $\sim \mathbf{t} = \mathbf{f}$ *and* $\sim \mathbf{f} = \mathbf{t}$.
- $\boldsymbol{I}(F \wedge G) = \mathtt{min}(\boldsymbol{I}(F),\boldsymbol{I}(G))$. *Recall that* $\mathbf{f} < \mathbf{t}$.
- $\boldsymbol{I}(F \vee G) = \mathtt{max}(\boldsymbol{I}(F),\boldsymbol{I}(G))$.
- *For a strict rule* $@r\ F \ {:-}\ G$, *we define* $\boldsymbol{I}(F \ {:-}\ G) = \mathbf{t}$ *if and only if* $\boldsymbol{I}(F) \geq \boldsymbol{I}(G)$.
Intuitively, a strict rule is true if its head is "more true" than the body, i.e., either the head is true or the body is false.
- *For a defeasible rule* $@r\ L_1 \vee ... \vee L_k \ {:-}\ G$, *we define* $\boldsymbol{I}(@r\ L_1 \vee ... \vee L_k \ {:-}\ G) = \mathbf{t}$ *if and only if* $\boldsymbol{I}(L_1 \vee ... \vee L_k) \geq \mathtt{min}(\boldsymbol{I}(G), V)$ *where* $V = max_{1 \leq i \leq k}\boldsymbol{I}(\mathsf{naf}\ \texttt{\$defeated}(r, L_i))$.
That is, a defeasible rule $@r$ *is true if either (i) it is "defeated," i.e.,* $\texttt{\$defeated}(r, L_i)$ *holds for all L_i; or (ii) its body is false; or (iii) if its head is true.* □

Definition 8 (Model of formula and rule) *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 $\boldsymbol{I}$.*

*If R is a ground rule $@r\ L_1 \vee ... \vee L_k \ {:-}\ G$, $\boldsymbol{I}$ an interpretation, and $\boldsymbol{I}(R) = \mathbf{t}$, then we write $\boldsymbol{I} \models R$ and say that $\boldsymbol{I}$ is a **model** of R or that R is **satisfied** in $\boldsymbol{I}$.*

We write $\boldsymbol{I} \models \mathcal{P}$ if $\boldsymbol{I} \models R$ for every $R \in \mathcal{P}$. □

Definition 9 (Model of *aspda* w.r.t. argumentation theory) *Given an* aspda *$\mathcal{P}$, an argumentation ruleset AT, and an interpretation $\boldsymbol{M}$, we say that $\boldsymbol{M}$ is a **model** of $\mathcal{P}$ with respect to the argumentation ruleset 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 10 (Minimal model) *A **minimal model** of $(\mathcal{P}, AT)$ is a model $\boldsymbol{M}$ of $(\mathcal{P}, AT)$ such that 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 [20] and the answer-set semantics [19] to ASPDA. We start with non-disjunctive *aspda*s and stable models.

Definition 11 (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**[6] of $\mathcal{Q}$ **by** $\boldsymbol{J}$, written as $\frac{\mathcal{Q}}{\boldsymbol{J}}$, is defined by the following sequence of steps:*

[6] In regular ASP theory, the term **reduct** is normally used. We later use the term reduction in a different sense, so quotient is used here to avoid confusion.

1. *Delete every rule $R \in \mathcal{Q}$ such that there is a* naf*-literal of the form* naf A *in R's body and $A \in \boldsymbol{J}$;*
2. *Delete every defeasible rule of the form*
   ($@r\ L$:- *Body*) $\in \mathcal{Q}$
   such that $\$\mathbf{defeated}(r, L) \in \boldsymbol{J}$.
3. *Remove all* naf*-literals from the remaining rules.*
4. *Remove all tags from the remaining rules.*

Note that $\frac{\mathcal{Q}}{\boldsymbol{J}}$ *is a normal (non-defeasible) logic program without* naf. □

When dealing with stable models, it is often assumed that interpretations are consistent [19]. All the definitions and results in this section extend to this case straightforwardly.

Definition 12 (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* $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$.
Note that $\frac{\mathcal{P} \cup AT}{\boldsymbol{M}}$ *is a Horn logic program, so here* minimal models *are meant in the sense of the regular Horn logic, not in the sense of Definition 10.* □

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 [30]).

Theorem 1 (Reduction for stable model semantics) *Let* $\mathcal{P}$ *be a non-disjunctive* `aspda` *and AT an argumentation ruleset. 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 $\wedge$ naf $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L})$
  and removing all the remaining tags; and AT′ is obtained from AT by simply removing all the tags. □

Proof: Let S be a Herbrand interpretation for $\mathcal{P} \cup AT$.
According to Definition 11, $\frac{\mathcal{P} \cup AT}{S}$ is obtained through the following steps:

1. Delete every rule $R \in \mathcal{P} \cup AT$ such that there is a naf-literal of the form naf A in R's body and $A \in S$;
2. Delete every defeasible rule of the form
   (@r L :- Body) $\in \mathcal{P} \cup AT$
   such that $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L}) \in S$.
3. Remove all naf-literals from the remaining rules in $\mathcal{P} \cup AT$.
4. Remove all tags from the remaining tagged rules in $\mathcal{P} \cup AT$.

Note that this makes $\frac{\mathcal{P} \cup AT}{S}$ into an ordinary logic program.
According to the definition of *Quotient* in the ordinary stable model semantics [20], the quotient of $\mathcal{P}' \cup AT'$ by S, is obtained through the following steps:

1. Delete every rule $R \in \mathcal{P}' \cup AT'$ such that there is a naf-literal of the form naf A in R's body and $A \in S$;
2. Remove all naf-literals from the remaining rules in $\mathcal{P}' \cup AT'$.

From the above it can be safely inferred that the ASPDA quotient $\frac{\mathcal{P} \cup AT}{S}$ is the same set of ordinary logic rules as the (ordinary) quotient of $\mathcal{P}' \cup AT'$ by S. For instance, consider a defeasible rule @r L :- Body in $\mathcal{P}$. If $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L}) \in S$, this rule will be deleted by the process of construction of $\frac{\mathcal{P} \cup AT}{S}$. The corresponding rule
L :- Body $\wedge$ naf $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L})$
will be deleted by the construction of the (ordinary) quotient of $\mathcal{P}' \cup AT'$ by S.
The claim now follows from the above and the definitions of stable models in ASPDA and in the classical case (Definition 12 and the one in [20]). □

For rules with disjunctions in the head, 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 difficulty is to find an analog of the reduction theorem (Theorem 1).

Example 1 *Consider a disjunctive program that has the following defeasible rules:*

```
@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 the proposition* a *cannot be true when either* d *or* e *holds, and that* b *is also incompatible with* e. *These constraints are specified as the following facts:*

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

Suppose, in addition, that rule r1 *has a higher priority than* r2*, which we specify using the fact*

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

Intuitively, $\{a, d\}$*,* $\{a, e\}$*, and* $\{b, e\}$ *can no longer be models due to the incompatibility constraints, while the models* $\{b, d\}$*,* $\{c, d\}$*, and* $\{c, e\}$ *are still possible. At the same time, one might feel that* $\{a\}$ *is also a suitable model because* r1 *overrides* r2*, the proposition* a *makes* r1 *true, and* a *is incompatible with both heads in rule* r2*.*

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

$\$\mathbf{defeated}(r2, d)$:-
  $\#\mathbf{overrides}(r1, r2) \wedge \#\mathbf{opposes}(a, d) \wedge a.$
$\$\mathbf{defeated}(r2, e)$:-
  $\#\mathbf{overrides}(r1, r2) \wedge \#\mathbf{opposes}(a, e) \wedge a.$ □

The following definitions generalize Definition 11 to disjunctive *aspda*s and make the intuition behind Example 1 precise.

Definition 13 (ASPDA quotient, disjunctive case) *Let* $\mathcal{Q}$ *be a disjunctive* aspda*, and let* ***J*** *be a Herbrand interpretation for* $\mathcal{Q}$*. We define the ASPDA* ***quotient of*** $\mathcal{Q}$ ***by J****, written as* $\frac{\mathcal{Q}}{\boldsymbol{J}}$*, by the following sequence of steps:*

1. *Delete every rule* $R \in \mathcal{Q}$ *that has a literal of the form* naf A *in* R*'s body where* $A \in \boldsymbol{J}$*;*
2. *For every defeasible rule of the form* $@r\ L_1 \vee ... \vee L_n$:- Body *in* $\mathcal{Q}$*, delete every* L_i *such that* $\$\mathbf{defeated}(r, L_i) \in \boldsymbol{J}$*. If all the* L_i*'s are deleted, delete the entire rule.*
3. *Remove all* naf *-literals from the remaining rules.*
4. *Remove all tags from the remaining tagged rules.* □

Definition 12 is generalized in a natural way:

Definition 14 (Answer set) *A Herbrand interpretation* ***M*** *is an* ***answer set*** *of a disjunctive* aspda $\mathcal{P}$ *with respect to the argumentation ruleset AT, if* ***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* $(@r\ L_1 \vee ... \vee L_n \text{ :- Body}) \in \mathcal{P}$ *into a collection of plain rules of the form*

$$\vee_{i\in K} L_i \text{ :- Body} \wedge \bigwedge_{i\in K} \text{naf } \$\mathbf{defeated}(r, L_i) \wedge \bigwedge_{j\in N-K} \$\mathbf{defeated}(r, L_j)$$

      for each non-empty subset $K \subseteq N$*, where* $N = \{1, ..., n\}$
    - *removing all the remaining tags.*
  - AT' *is obtained from AT by simply removing all the tags.*

Proof: Let S be a Herbrand interpretation of $\mathcal{P} \cup AT$. By Definition 13, $\frac{\mathcal{P}\cup AT}{S}$ is constructed by the following steps:

1. Delete every rule $R \in \mathcal{P} \cup AT$ that has a literal of the form naf A in R's body, where $A \in S$;
2. For every defeasible rule of the form $@r\ L_1 \vee ... \vee L_n$:- Body in $\mathcal{P} \cup AT$, delete every L_i such that $\$\mathbf{defeated}(r, L_i) \in S$. If all the L_i's are deleted, delete the entire rule.
3. Remove all naf -literals from the remaining rules.
4. Remove all tags from the remaining tagged rules.

Note that $\frac{\mathcal{P}\cup AT}{S}$ is an ordinary disjunctive logic program. For future reference, let us denote it $\mathcal{Q}_1$.

By definition of the quotient in the ordinary answer set semantics [19], the quotient of $\mathcal{P}' \cup AT'$ by S, is obtained from $\mathcal{P}' \cup AT'$ by these steps:

(i) Delete every rule $R \in \mathcal{P}' \cup AT'$ that has a literal of the form naf A in R's body where $A \in S$;
(ii) Remove all naf -literals from the remaining rules.

Let us denote the resulting logic program with $\mathcal{Q}_2$. We will call the rules in $\mathcal{Q}_1$ and $\mathcal{Q}_2$ the *reducts* of the original rules in $\mathcal{P} \cup AT$ and $\mathcal{P}' \cup AT'$, respectively.
Now consider a rule R $(@r\ L_1 \vee ... \vee L_n \text{ :- Body}) \in \mathcal{P}$.

– If there is a literal of the form naf A in R's body and $A \in S$, R would be deleted and its reduct will be neither in $\mathcal{Q}_1$ nor $\mathcal{Q}_2$.
– If no such literal naf A exists in R then *Body* of the reduct of R does not contain naf-literals. Let $K_0 \subseteq \{1, ..., n\}$ be a subset such that $S \models$ naf $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i})$, for all $\mathtt{i} \in K_0$, and $S \models \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$, for all $\mathtt{j} \notin K_0$. Then, if $K_0 \neq \{\}$,

  • $\mathcal{Q}_1$ would contain the rule $\vee_{\mathtt{i} \in \mathtt{K_0}} \mathtt{L_i}$:- Body — the reduct of R.
  • $\mathcal{P}' \cup AT'$ would contain a set of rules of the form

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i} \text{ :- Body} \wedge \bigwedge_{i \in K} \text{naf } \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i}) \wedge \bigwedge_{j \in N-K} \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$$

  for each non-empty subset $K \subseteq N = \{1, ..., n\}$. During the construction of $\mathcal{Q}_2$, after step (i) the only remaining rules will be of the form

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i} \text{ :- Body} \wedge \bigwedge_{i \in K} \text{naf } \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i}) \wedge \bigwedge_{j \in N-K} \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$$

  for each K such that $K \subseteq K_0$. After step (ii), the reducts of R that will remain in $\mathcal{Q}_2$ would be:

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i} \text{ :- Body} \wedge \bigwedge_{j \in N-K} \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$$

  for each K such that $K \subseteq K_0$. Among these rules, only one rule, $\vee_{\mathtt{i} \in \mathtt{K_0}} \mathtt{L_i}$:- Body $\wedge \wedge_{\mathtt{j} \in \mathtt{N-K_0}} \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$, can possibly have a body entailed by S. Furthermore, S entails this rule if and only if S entails $\vee_{\mathtt{i} \in \mathtt{K_0}} \mathtt{L_i}$:- Body, which is a reduct of R in $\mathcal{Q}_1$.

If $K_0 = \{\}$ then, for $\mathtt{i}$=1,...,n, $S \models \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i})$. Therefore:

  • $\mathcal{Q}_1$ has no reducts of R. So, the entire rule is deleted in step 2 (of ASPDA quotient).
  • $\mathcal{P}' \cup AT'$ must contain the rules of the form

$$\vee_{\mathtt{i} \in \mathtt{K}} \mathtt{L_i} \text{ :- Body} \wedge \bigwedge_{i \in K} \text{naf } \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i}) \wedge \bigwedge_{j \in N-K} \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j})$$

  for each non-empty subset $K \subseteq N = \{1, ..., n\}$. Each such rule contains at least one literal naf $\$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i})$ in the rule body. Since $K_0 = \{\}$ implies that all such literals are false in S, step (i) in the construction of $\mathcal{Q}_2$ eliminates all the above rules. So, neither $\mathcal{Q}_1$ nor $\mathcal{Q}_2$ will have any reducts of R.

It can now be seen that S is a minimal Herbrand model of $\mathcal{Q}_1$ if and only if S is a minimal Herbrand model of $\mathcal{Q}_2$. In other words, S is an answer set for the *aspda* $\mathcal{P}$ with respect to *AT* if and only if S is an answer set for the ordinary logic program $\mathcal{P}' \cup AT'$. □

With this theorem, it is now straightforward to verify that the answer sets for the *aspda* in Example 1 are precisely as described there.

Theorem 2 shows that a reduction exists from ASPDA to ASP, but that particular reduction is exponential in size with respect to the original program. With a little more care, a polynomial reduction can be constructed, as has been recently shown by Faber [16].

3.3. Reduction to the Non-disjunctive Case

In ordinary answer-set programming, some disjunctive rules can be reduced to the non-disjunctive case via the so-called *shifting* transformation. This transformation would replace the rule $\mathtt{L_1} \vee ... \vee \mathtt{L_n}$:- Body with n new rules

$$\mathtt{L_i} \text{ :- Body} \wedge \bigwedge_{1 \leq \mathtt{j} \leq \mathtt{n},\ \mathtt{j} \neq \mathtt{i}} \text{naf } \mathtt{L_j} \qquad (2)$$

where $1 \leq i \leq n$. We will use $shift(\mathcal{P})$ to denote such transformation of a (non-defeasible) disjunctive logic program. For example, consider a program consisting of one rule $\mathtt{p} \vee \mathtt{q} \vee \mathtt{s}$:- body, the shifting of the program is

```
p :- body ∧ naf q ∧ naf s.
q :- body ∧ naf p ∧ naf s.
s :- body ∧ naf q ∧ naf p.
```

Ben-Eliyahu and Dechter [4] have shown that the above shifting transformation is an *equivalence* transformation for so called *head-cycle free* programs.[7] We reproduce that definition below adjusting it for disjunctive *aspda*s.

Definition 15 *[4] The* ***dependency graph*** $G_{\mathcal{P}}$*, of a ground* aspda $\mathcal{P}$*, is a directed graph where nodes are ground literals. An edge going from literal* L *to literal* L' *exists if and only if there is a rule in which* L *appears positively in the body and* L' *is a head literal. An* aspda *is* ***head-cycle free*** *if and only if its dependency graph does not contain directed cycles that connect literals belonging to the head of the same rule.* □

In the above example, if p, q, s have only negative occurrences (or no occurrences at all) in body then the *aspda* consisting only of the rule

$$@\mathtt{r}\ \mathtt{p} \vee \mathtt{q} \vee \mathtt{s}\ \texttt{:-}\ \mathtt{body}$$

is head-cycle free.

Under certain restrictions, the head-cycle free property for $\mathcal{P} \cup AT$ can be reduced to head-cycle freedom for $\mathcal{P}$. For example, if the literals that appear in rule heads in *AT* do not appear in any rule body in $\mathcal{P}$, and *AT* is non-disjunctive, then $\mathcal{P} \cup AT$ is head-cycle free if and only if $\mathcal{P}$ is head-cycle free. This is satisfied in the argumentation ruleset AT^{DL} in Section 4.2. It is also satisfied in the argumentation ruleset AT^{AGCLP} in Section 4.1 if #**overrides** and #**opposes** do not appear in rule bodies in $\mathcal{P}$ (which normally is the case).

An interesting question is whether a shifting transformation analogous to ordinary answer-set programming exists, and an equivalence result holds for disjunctive *aspda*s.

Definition 16 *Let* $\mathcal{P}$ *be a disjunctive* aspda*. We define* ***t-shifting*** *of* $\mathcal{P}$*,* $t\_shift(\mathcal{P})$*, as a non-disjunctive* aspda *obtained from* $\mathcal{P}$ *by replacing each rule of the form* $(@\mathtt{r}\ \mathtt{L_1} \vee ... \vee \mathtt{L_n}\ \texttt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* n *new rules*

$$@\mathtt{r}\ \ \mathtt{L_i}\ \texttt{:-}\ \mathtt{Body} \wedge \bigwedge_{1 \le \mathtt{j} \le \mathtt{n},\ \mathtt{j} \ne \mathtt{i}} \mathsf{naf}\, \mathtt{L_j}$$

where $1 \le i \le n$. □

Surprisingly, it turns out that $t\_shift(\mathcal{P})$ is *not* equivalent to $\mathcal{P}$ even for head-cycle free *aspda*s. To see this, consider the following rule set, $\mathcal{P}^{ex}$:

```
@r1 a ∨ b ∨ c.
@r2 d.
@r3 c.
```

Suppose that the associated argumentation rules imply $\$\mathbf{defeated}(\mathtt{r1}, \mathtt{c})$ and does not imply any other $\$\mathbf{defeated}(...)$ facts involving the above rules. Then $\mathcal{P}^{ex}$ would have the following answer sets: $\{\mathtt{a}, \mathtt{d}, \mathtt{c}\}$ and $\{\mathtt{b}, \mathtt{d}, \mathtt{c}\}$. In contrast, the above t-shifting transformation yields the following non-disjunctive *aspda*, $t\_shift(\mathcal{P}^{ex})$:

```
@r1 a :- naf b ∧ naf c.
@r1 b :- naf a ∧ naf c.
@r1 c :- naf a ∧ naf b.
@r2 d.
@r3 c.
```

which has only one answer set: $\{\mathtt{d}, \mathtt{c}\}$ with respect to the argumentation ruleset. This shows that t_shift is not an equivalence transformation under ASPDA.

Fortunately, a result similar to Ben-Eliyahu and Dechter's does hold for disjunctive *aspda*s, but for a slightly different shifting transformation.

Definition 17 *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 strict rule with its t-shifting and replacing each defeasible rule of the form* $(@\mathtt{r}\ \mathtt{L_1} \vee ... \vee \mathtt{L_n}\ \texttt{:-}\ \mathtt{Body}) \in \mathcal{P}$ *with* n *new defeasible rules and* $2n$ *new strict rules as follows:*

$$\begin{array}{ll} @\mathtt{r} & \mathtt{L_i}\ \texttt{:-}\ \mathtt{Body} \wedge \bigwedge_{1 \le \mathtt{j} \le \mathtt{n},\ \mathtt{j} \ne \mathtt{i}} \mathsf{lit}(\mathtt{r}, \mathtt{L_j}). \\ & \mathsf{lit}(\mathtt{r}, \mathtt{L_i})\ \texttt{:-}\ \mathsf{naf}\, \mathtt{L_i}. \\ & \mathsf{lit}(\mathtt{r}, \mathtt{L_i})\ \texttt{:-}\ \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_i}). \end{array} \quad (3)$$

where $1 \le i \le n$*. Here* $\mathsf{lit}(\mathtt{r}, \mathtt{L_i})$*,* $1 \le i \le n$*, are literals of the form* $\mathtt{newsym_i}(\mathtt{Vars_i})$*, where* $\mathtt{newsym_i}$ *is a fresh predicate name that depends only on* r *and* $\mathtt{L_i}$*, while* $\mathtt{Vars_i}$*, the argument vector of the literal, is a vector of variables that occur in* r *and* $\mathtt{L_i}$*. We omit the rule tags for strict rules here.* □

Theorem 3 *Let* $\mathcal{P}$ *be an* aspda *and let AT be an argumentation ruleset such that* $\mathcal{P} \cup AT$ *is head-cycle free. There is a one-to-one relationship between the answer sets of* $\mathcal{P}$ *with respect to AT and the answer sets of* aspda$_shift(\mathcal{P})$ *with respect to* $t\_shift(AT)$*. Namely, a Herbrand interpretation* S *is an answer set of* $\mathcal{P}$ *with respect to AT if and only if* $f(S)$ *is an answer set of* aspda$_shift(\mathcal{P})$ *with respect to* $t\_shift(AT)$*, where* $f(S) = S \cup \{\mathsf{lit}(r, L) \mid \mathcal{P}$ *contains a rule with*

[7] The works [13,21] developed similar shifting techniques.

tag r and with L in its head (possibly as a disjunct), so that either $\$\mathbf{defeated}(r, L) \in S$ *or* $L \notin S\}$.

Proof: The proof consists of establishing five equivalences, which we denote $\Leftrightarrow_1$, ..., $\Leftrightarrow_5$.

S is an answer set of $\mathcal{P}$ with respect to AT

$\Leftrightarrow_1$

S is a minimal Herbrand model of $\mathcal{Q}_1 = \dfrac{\mathcal{P} \cup AT}{S}$

$\Leftrightarrow_2$

S is an answer set of

$\mathcal{Q}_2 = shift(\mathcal{Q}_1) = shift(\dfrac{\mathcal{P} \cup AT}{S})$

$\Leftrightarrow_3$

S is a minimal Herbrand model of

$\mathcal{Q}_3 = \dfrac{\mathcal{Q}_2}{S} = \dfrac{shift(\frac{\mathcal{P}\cup AT}{S})}{S}$

$\Leftrightarrow_4$

$f(S)$ is a minimal Herbrand model of

$\mathcal{Q}_4 = \dfrac{aspda\_shift(\mathcal{P}) \cup t\_shift(AT)}{f(S)}$

$\Leftrightarrow_5$

$f(S)$ is an answer set of $aspda\_shift(\mathcal{P})$ with respect to $t\_shift(AT)$.

In proving each equivalence, we will choose an arbitrary defeasible rule R of the form $@\mathtt{r}\ \mathtt{L_1} \vee ... \vee \mathtt{L_n} :- \mathtt{Body}$ in $\mathcal{P}$ and an arbitrary strict rule $T \in \mathcal{P} \cup AT$, and then look at what happens to these rules after applying the quotient and shifting transformations to them. As in the proof of Theorem 2, we can assume that the bodies of R and T do not contain naf-literals (they are evaluated away in the quotients on both sides).

Let K_0 be $\{k \mid S \models \mathsf{naf}\ \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_k}),\ 1 \le \mathtt{k} \le \mathtt{n}\}$ and let K_1 be $\{k \mid L_k \in S,\ 1 \le k \le n\}$.

($\Leftrightarrow_1$): This follows from Definition 14.

($\Leftrightarrow_2$): By definition, every defeasible rule $R \in \mathcal{P}$ gives rise to the following single rule R_1 in the quotient $\mathcal{Q}_1$:

$$\bigvee_{\mathtt{i} \in \mathtt{K_0}} \mathtt{L_i} :- \mathtt{Body} \qquad (4)$$

If $|K_0| = 0$, R gives rise to no rule.

The strict rules $T \in \mathcal{P} \cup AT$ give rise to T_1 in $\mathcal{Q}_1$ where T_1 has the same head and body as T but the tag is stripped off. By definition, all rules in $\mathcal{Q}_1$ are either R_1s or T_1s and are obtained in the above way. So, $\mathcal{Q}_1$ consists of the rules of the form (4) or of the strict rules from $\mathcal{P} \cup AT$ that lost their tag.

$\mathcal{Q}_2$ is constructed from $\mathcal{Q}_1$ via shifting of ordinary (non-defeasible) disjunctive rules. A rule R_1 of the form (4) produces $|K_0|$ rules of the form

$$\mathtt{L_i} :- \mathtt{Body} \wedge \bigwedge_{\mathtt{j} \in \mathtt{K_0}, \mathtt{j} \neq \mathtt{i}} \mathsf{naf}\, \mathtt{L_j} \qquad (5)$$

for $i \in K_0$. The strict rule $T_1 \in \mathcal{Q}_1$ gives rise to the rules $shift(T_1)$.

Since, by assumption, $\mathcal{Q}_1$ does not contain naf-literals, S is a minimal Herbrand model of $\mathcal{Q}_1$ iff S is an answer set of $\mathcal{Q}_1$. Observe that:

- $\mathcal{Q}_1$ is an ordinary (non-defeasible) disjunctive logic program,
- $\mathcal{P} \cup AT$ is head-cycle free, so $\mathcal{Q}_1 = \frac{\mathcal{P}\cup AT}{S}$ is head-cycle free,

Therefore, as shown in [4,13,21], S is an answer set of $\mathcal{Q}_1$ iff S is an answer set of $\mathcal{Q}_2 = shift(\mathcal{Q}_1)$.

$\mathcal{Q}_2$ contains no rules other than those in (5) and $shift(T_1)$.

($\Leftrightarrow_3$): $\mathcal{Q}_3 = \frac{\mathcal{Q}_2}{S}$ is constructed according to Definition 13. Strict rules in $\mathcal{Q}_3$ all have the form $\frac{shift(T_1)}{S}$ and defeasible rules are obtained as follows:

3-a. If $|K_1 \cap K_0| \ge 2$, the rules of the form (5) yield nothing in $\mathcal{Q}_3$. Indeed, for each rule in (5), there must exist at least one j satisfying $j \in K_0, j \neq i$, and $L_j \in S$, so every such rule will be deleted after Step 1 in Definition 13.

3-b. If $|K_1 \cap K_0| = 1$, (5) yields $\{\mathtt{L_i} :- \mathtt{Body} \mid$ where $\mathtt{i} \in \mathtt{K_1} \cap \mathtt{K_0}\}$ in $\mathcal{Q}_3$. This is because every rule in (5) such that $i \notin K_1$ is deleted in Step 1 in Definition 13, and the rules such that $i \notin K_0$ are deleted in Step 2. The naf-literals in the remaining rule are deleted in Step 3.

3-c. If $|K_1 \cap K_0| = 0$, (5) yields the rules $\{\mathtt{L_i} :- \mathtt{Body} \mid \mathtt{i} \in \mathtt{K_0}\}$ in $\mathcal{Q}_3$. This is because the rules in (5) such that $i \notin K_0$ are deleted in Step 2 of Definition 13 while the naf-literals in the remaining rules are deleted in Step 3.

($\Leftrightarrow_5$): The fifth equivalence in the proof of the theorem is a direct consequence of Definition 14, so we dispense with this case before the fourth equivalence.

($\Leftrightarrow_4$): $\mathcal{Q}_3$ is divided into two subsets: $\mathcal{Q}_3^{df}$, which consists of rules obtained from the defeasible rules in $\mathcal{Q}_2$, and $\mathcal{Q}_3^{st}$ which consists of rules obtained from the strict rules in $\mathcal{Q}_2$.

$\mathcal{Q}_4$ is divided into $\mathcal{Q}_4^{df}$ and $\mathcal{Q}_4^{st}$ the same way. $\mathcal{Q}_4$ is obtained from $aspda\_shift(\mathcal{P}) \cup t\_shift(AT)$ by applying the steps in Definition 11. Recall that each defeasible rule $R \in \mathcal{P}$ gives rise to a collection of rules of the form (3) in $aspda\_shift(\mathcal{P})$, which in $\mathcal{Q}_4$ become

$$\mathtt{L_i} \ \mathtt{:-}\ \mathtt{Body} \wedge \bigwedge_{\mathtt{1\leq j\leq n, j\neq i}} \mathsf{lit}(\mathtt{r}, \mathtt{L_j}). \quad (6)$$

for each $i \in K_0$;

$$\mathsf{lit}(\mathtt{r}, \mathtt{L_j}) \ \mathtt{:-}\ \$\mathbf{defeated}(\mathtt{r}, \mathtt{L_j}). \quad (7)$$

for each $1 \leq j \leq n$;

$$\mathsf{lit}(\mathtt{r}, \mathtt{L_j}). \quad (8)$$

for each $j \notin K_1$. All these rules constitute $\mathcal{Q}_4^{df}$.
Each strict rule $T \in \mathcal{P} \cup AT$ gives rise to the set of rules $\frac{t\_shift(T)}{S}$ in $\mathcal{Q}_4$. These rules constitute $\mathcal{Q}_4^{st}$.

The difference between $t\_shift(T)$ used in $\mathcal{Q}_4$ and $shift(T_1)$ used in $\mathcal{Q}_3$ is just that T has a tag while T_1 does not. The quotient operation removes tags, so $\frac{t\_shift(T)}{S} = \frac{shift(T_1)}{S}$. Because every rule in $\mathcal{Q}_4^{st}$ is of the form $\frac{t\_shift(T)}{S}$ for some T and every rule in $\mathcal{Q}_3^{st}$ has the form $\frac{shift(T_1)}{S}$ for some T_1 (which is obtained from T by tag removal), we have:

$f(S)$ is a Herbrand model of $\mathcal{Q}_4^{st}$ iff
S is a Herbrand model of $\mathcal{Q}_3^{st}$. (9)

Now consider the defeasible rules in $\mathcal{Q}_4^{df}$. Since (7) and (8) are the only rules that define $\mathsf{lit}(r, L_j)$, it follows that $f(S) \not\models \mathsf{lit}(r, L_j)$, $\forall j \in K_1 \cap K_0$, and $f(S) \models \mathsf{lit}(r, L_j)$, $\forall j \notin K_1 \cap K_0$. Under $f(S)$,

4-a. If $|K_1 \cap K_0| \geq 2$, the rules in (6) yield nothing in $\mathcal{Q}_4$, since for each $i \in K_0$, every rule (6) has some body literal $\mathsf{lit}(r, L_j)$ such that $f(S) \not\models \mathsf{lit}(r, L_j)$.

4-b. If $|K_1 \cap K_0| = 1$, (6) gives rise to a single rule $\mathtt{L_i} \ \mathtt{:-}\ \mathtt{Body}$, where $i \in K_1 \cap K_0$. Indeed, any rule in (6) such that $i \notin K_1 \cap K_0$ has some body literal $\mathsf{lit}(r, L_j)$ such that $f(S) \not\models \mathsf{lit}(r, L_j)$.

4-c. If $|K_1 \cap K_0| = 0$, (6) gives rise to the following rules $\{\mathtt{L_i} \ \mathtt{:-}\ \mathtt{Body} \mid \mathtt{i} \in \mathtt{K_0}\}$, which is obtained by the same argument as before.

Every rule in $\mathcal{Q}_4^{df}$ comes from (6) or (7) or (8) and all the rules of the form (7) and (8) are satisfied in $f(S)$. By comparing (3-a),(3-b),(3-c) with (4-a), (4-b), (4-c), it follows that

$f(S)$ is a Herbrand model of $\mathcal{Q}_4^{df}$ iff
S is a Herbrand model of $\mathcal{Q}_3^{df}$. (10)

From (10) and (9) we obtain

$f(S)$ is a Herbrand model of $\mathcal{Q}_4$ iff
S is a Herbrand model of $\mathcal{Q}_3$. (11)

To complete the proof for the equivalence ($\Leftrightarrow_4$), it remains to show that $f(S)$ is a minimal model of $\mathcal{Q}_4$ if and only if so is S for $\mathcal{Q}_3$.

Minimality of $f(S)$: if S is a minimal Herbrand model of $\mathcal{Q}_3$, then $\forall A \in f(S)$, $f(S) - \{A\}$ cannot be a Herbrand model of $\mathcal{Q}_4$ because:

- if $A \in S$, the minimality of S for $\mathcal{Q}_3$ implies that there must be a rule R_1 of the form (5) or $\frac{shift(T')}{S}$ such that $S - \{A\} \not\models R_1$. By the previously established correspondence between the rules in $\mathcal{Q}_3$ and $\mathcal{Q}_4$, there is a rule $R_2 \in \mathcal{Q}_4$ of the form (6) or $\frac{t\_shift(T)}{S}$ which, by construction, must be such that $f(S) - \{A\} \not\models R_2$.
- if $A = \mathsf{lit}(r, L)$ for some r and L, there must be some rule R of the form (4-b) or (4-c) such that $f(S) - \{A\} \not\models R$, so $f(S)$ also cannot be a model of $\mathcal{Q}_4$ in this case.

Minimality of S: if $f(S)$ is a minimal Herbrand model of $\mathcal{Q}_4$, then for any $A \in S$, $S - \{A\}$ cannot be a Herbrand model of $\mathcal{Q}_3$. If it were a model then, by (11), $f(S - \{A\}) \subset f(S)$ must be a Herbrand model of $\mathcal{Q}_4$, contrary to the assumption that $f(S)$ is a minimal Herbrand model of $\mathcal{Q}_4$.

This concludes the proof of ($\Leftrightarrow_4$) and of the theorem. □

4. Examples of Argumentation Rulesets

We will now introduce two very different sets of argumentation rules and then discuss how the choice of an argumentation ruleset affects the semantics on a number of simple knowledge bases.

4.1. A-GCLP [24,40]

Our first example is an ASPDA counterpart for the argumentation theory proposed in [40], which captures generalized courteous logic programs [24] under the well-founded semantics [18]. We will call this theory A-GCLP and will denote it by AT^{AGCLP}. It is this argumentation ruleset that was tacitly assumed 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 rules only impose some constraints on #**opposes**.

The $\$$**defeated** predicate is defined as follows: A rule is *defeated* if it is *refuted* by some other undefeated rule. In the ATs below, *aspda* rules are represented by pairs of variables $?T$, $?L$ (possibly with subscripts or primes) where $?T$ ranges over rule tags and $?L$ over rule heads.

$$\$\textbf{defeated}(?T, ?L) \;\texttt{:-}\; \$\mathsf{defeats}(?T', ?L', ?T, ?L).$$

The auxiliary predicate $\$$defeats is defined as follows:

$$\begin{aligned}\$\mathsf{defeats}(?T_1, ?L_1, ?T_2, ?L_2) \;\texttt{:-}\; & \\ & \$\mathsf{refutes}(?T_1, ?L_1, ?T_2, ?L_2) \wedge \\ & \mathsf{naf}\ \$\textbf{defeated}(?T_1, ?L_1) \wedge \\ & \mathsf{naf}\ \#\textbf{strict}(?T_2, ?L_2).\end{aligned}$$

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

$$\begin{aligned}\$\mathsf{refutes}(?T_1, ?L_1, ?T_2, ?L_2) \;\texttt{:-}\; & \\ & \$\mathsf{conflict}(?T_1, ?L_1, ?T_2, ?L_2) \wedge ?L_1 \\ & \wedge\ \#\textbf{overrides}(?T_1, ?L_1, ?T_2, ?L_2).\end{aligned}$$

The definition 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:

$$\$\mathsf{candidate}(?T, ?L) \;\texttt{:-}\; \textbf{body}(?T, ?L, ?B) \wedge ?B.$$

Here the meta-predicate **body** binds $?B$ to the body of a rule with the tag $?T$ and head $?L$.

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

$$\begin{aligned}\$\mathsf{conflict}(?T_1, ?L_1, ?T_2, ?L_2) \;\texttt{:-}\; & \\ & \$\mathsf{candidate}(?T_1, ?L_1) \wedge \$\mathsf{candidate}(?T_2, ?L_2) \\ & \wedge\ \#\textbf{opposes}(?L_1, ?L_2).\end{aligned}$$

Recall that the #**opposes** information is supplied by the knowledge engineer. However, argumentation rules may include additional background axioms. In our case, AT^{AGCLP} supplies the following background axioms for #**opposes**:

$$\begin{aligned}& \#\textbf{opposes}(?L_1, ?L_2) \;\texttt{:-}\; \#\textbf{opposes}(?L_2, ?L_1). \\ & \#\textbf{opposes}(?L, \mathsf{neg}\ ?L). \\ & \texttt{:-}\; ?L_1 \wedge ?L_2 \wedge \#\textbf{opposes}(?L_1, ?L_2).\end{aligned}$$

The first is a symmetry axiom that states that opposition is a reciprocal relation. The second axiom states that literals and their negations are in opposition to each other. The third axiom is a constraint that says that opposing literals cannot be both true in the same possible world.

The relation #**overrides** is also mostly defined by the knowledge engineer. However, AT^{AGCLP} also supplies a background axiom that establishes preference for strict rules over defeasible ones:

$$\begin{aligned}\#\textbf{overrides}(?T_1, ?L_1, ?T_2, ?L_2) \;\texttt{:-}\; & \\ & \#\textbf{strict}(?T_1, ?L_1) \wedge \mathsf{naf}\ \#\textbf{strict}(?T_2, ?L_2).\end{aligned}$$

Overriding is often specified via tags instead of tag-head pairs, and this was the form of overriding that we mostly used in the examples. The relationship between overriding through tag-head pairs and overriding via tags is defined by the following rule:

$$\begin{aligned}\#\textbf{overrides}(?T_1, ?L_1, ?T_2, ?L_2) \;\texttt{:-}\; & \\ & \#\textbf{overrides}(?T_1, ?T_2) \wedge \\ & \textbf{head}(?T_1, ?L_1) \wedge \textbf{head}(?T_2, ?L_2).\end{aligned}$$

Here **head** is a meta-predicate that relates tags to the heads of the rules labeled with those tags. The body-occurrence of #**overrides** is the overriding relation over tags and the head occurrence is the overriding relation over tag-head pairs.

Similarly, #**strict** is also often specified over tags and the following axiom relates that to strictness at the level of tag-head pairs.

$$\#\mathtt{strict}(?T, ?L) \text{ :- } \#\mathtt{strict}(?T) \wedge \mathbf{head}(?T, ?L).$$

Having defined this argumentation ruleset precisely, we can now come back to Example 1 and verify that the *aspda* there has four answer sets as claimed: $\{\mathtt{a}\}, \{\mathtt{b}, \mathtt{d}\}, \{\mathtt{c}, \mathtt{d}\}, \{\mathtt{c}, \mathtt{e}\}$.

4.2. *Defeasible Logic [1]*

Our second argumentation ruleset is intended to closely approximate Defeasible Logic under the stable model semantics as defined in [1]. [8]

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 important restrictions [9] in [1] are that it does not support disjunctions in the rule heads; opposition among literals is limited to p and neg p, for each p; it does *not* use default negation, so all literals are naf-free; and the rule tags are also rule identifiers, so no two rules have the same tag. This implies that rule tags uniquely determine rule's head and body and lets us simplify the argumentation rules by considering tags only and ignoring rule heads in most cases.

We can now formulate the argumentation rules, which we denote as AT^{DL}, for Defeasible Logic under the stable model semantics as defined in [1].

$$
\begin{aligned}
&\$\mathbf{defeated}(?T, ?L) \text{ :- } \$\mathsf{conflict}(?T, ?T') \wedge \\
&\qquad \mathbf{head}(?T', ?L') \wedge \$\mathsf{definitely}(?L'). \\
&\$\mathbf{defeated}(?T, ?L) \text{ :- } \#\mathbf{defeater}(?T). \\
&\$\mathbf{defeated}(?T, ?L) \text{ :- } \$\mathsf{overruled}(?T).
\end{aligned}
$$

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

[8] [1] uses two kinds of semantics for meta rules: the Kunen semantics and the stable model semantics, they yield different results for programs with cycle in the dependency graph.

[9] There are also other variants of Defeasible Logic, e.g., [5], which include rules with head disjunctions and other features.

The predicate $definitely is defined as follows:

$$
\begin{aligned}
&\$\mathsf{definitely}(?L) \text{ :- } \\
&\qquad \#\mathbf{strict}(?T) \wedge \mathbf{head}(?T, ?L) \wedge \\
&\qquad \mathbf{body}(?T, ?B) \wedge \mathbf{each\_definite}(?B).
\end{aligned}
$$

As in A-GCLP, **body** is a meta-predicate that binds $?B$ to the body of a rule with tag $?T$; **each_definite**$(?B)$ is a meta predicate; it is true when $\$\mathsf{definitely}(?B)$ is true or when $?B$ is bound to a conjunction, $conj$, and $\$\mathsf{definitely}(c)$ is true for each conjunct $c \in conj$. If t is a tag corresponding to a fact, then we assume that this is a rule whose body is an empty conjunct (i.e., (), which is commonly identified with **true** in logic), so **body**(t,()) holds and **each_definite**(()) is thus true. In this way, facts provide the base case for the recursive definition of $\$\mathsf{definitely}(?L)$.

The predicate $candidate is defined as before except that it now depends only on rule tags rather than tags and heads:

$$\$\mathsf{candidate}(?T) \text{ :- } \mathbf{body}(?T, ?B) \wedge ?B.$$

It remains to define $overruled, which relies on the notion of candidacy and conflict, as in AT^{AGCLP}.

$$
\begin{aligned}
&\$\mathsf{overruled}(?T) \text{ :- } \\
&\quad \$\mathsf{conflict}(?T, ?T') \wedge \$\mathsf{candidate}(?T') \wedge \\
&\quad \mathsf{naf}\ \$\mathsf{refuted}(?T'). \\
&\$\mathsf{refuted}(?T') \text{ :- } \\
&\quad \$\mathsf{conflict}(?T, ?T') \wedge \$\mathsf{candidate}(?T) \wedge \\
&\quad \#\mathbf{overrides}(?T, ?T') \wedge \mathsf{naf}\ \#\mathbf{defeater}(?T). \\
&\$\mathsf{conflict}(?T, ?T') \text{ :- } \\
&\quad \mathbf{head}(?T, ?L) \wedge \mathbf{head}(?T', \mathsf{neg}\ ?L).
\end{aligned}
$$

At this point it is instructive to retrospect on the differences between the two sets of argumentation rules presented here. First, there are differences in syntax and in how priorities over the rules are specified:

1. AT^{DL} does not support naf or disjunction in rule heads;
2. AT^{AGCLP} is more general in that tags are not required to be distinct and inclusion of variables in the tags provides one more level of differentiation among *rule instances*.

The other main difference is in the way **$defeated** is defined. In AT^{AGCLP}, a rule $?S$ is defeated if it is overridden by another rule $?R$ such that that $?R$ conflicts with $?S$. In contrast, in AT^{DL}, a rule $?T$ is defeated if it conflicts with a rule that is not overridden.

This leads to significant differences in the behavior of the two argumentation rulesets for the examples discussed in Section 4.3.

4.3. Examples

We now discuss a number of examples to help better understand the ASPDA semantics and the differences between the argumentation rulesets presented earlier. In all the examples, rules that have explicit tags are assumed to be defeasible and the rules without the tags are assumed strict.

Example 2 *Consider again the turkey-shoot example presented in Section 2.*

Under the AT^{AGCLP} argumentation rules, this example set has two answer sets. One is
{neg `loaded(g1, 1)`, `loaded(g2, 1)`, neg `alive(3)`}
and the other is
{`loaded(g1, 1)`, neg `loaded(g2, 1)`, neg `alive(3)`}.
Thus, AT^{AGCLP} yields the expected result.

As to the AT^{DL} argumentation rules and the logic in [1], this theory does not support disjunctions in rule heads directly. However, we can work around this issue by applying the shifting transformation. Shifting is applicable here because head-disjunctions in the turkey-shoot example are head-cycle-free. Under this transform, AT^{DL} yields the same result as AT^{AGCLP}. □

Example 3 *Figure 2 describes a scenario where a toxic discharge into a river caused massive reduction in fish population.*

Here both AT^{AGCLP} and AT^{DL} lead to the same conclusion:
`{ fishCount(s0+1,Squamish,trout,400),`
`fishCount(s0+2,Squamish,trout,0) }`
This is the expected result, meaning that up to the moment of the toxic discharge, the Squamish river had 400 trouts and then all of them died.

Interestingly, the same conclusion would be reached under LPDA [40]—a sibling of ASPDA developed for the well-founded semantics—if we use either the very same argumentation ruleset AT^{DL}, which we used here, or AT^{GCLP}, a ruleset analogous to AT^{AGCLP} but designed for the well-founded semantics [40].

Thus, in this example, both AT^{AGCLP} and AT^{DL} yield the same result and this is also true under the well-founded semantics. □

Example 4 *[40] Figure 3 specifies part of a game where blocks are moved from square to square on a board.*

The argumentation rules AT^{AGCLP} under ASPDA and AT^{GCLP} under LPDA both give the same expected result in this case:
`{loc(0,block4,square7),`
`loc(1,block4,square7),`
`loc(2,block4,square7),`
`loc(3,block4,square3)}`. □

Again, AT^{DL} does not handle this example directly, since the syntax in [1] does not include naf*. However, [3] shows that* `naf` *can be simulated using a transform that relies on* `neg` *only. Under this transform, AT^{DL} yields the same result as the other two theories.* □

Example 5 *Figure 4 shows a scenario where a cycle exists in the* `#`**`overrides`** *relation between a pair of opponents.*

Under ASPDA, AT^{DL} yields an answer set in which both `a` *and* `b` *are true. Indeed, one can verify that the following literals are true*
`$`refuted`(r1,a)`,
`$`refuted`(r2,b)`,
naf `$`overruled`(r1,a)`,
naf `$`overruled`(r2,b)`,
naf `$`**`defeated`**`(r1,a)`, *and*
naf `$`**`defeated`**`(r2,b)`.
Hence this program has only one answer set, in which both `a` *and* `b` *are true.*

The intuition is: Each of the two rules `r1` *and* `r2` *has some rule that overrides it, so both of them are refuted; Since refuted rules cannot be used to overrule any other rule, there is no rule overruled, and there is no rule defeated.*

However, AT^{AGCLP} does not produce this answer set. Indeed, consider an interpretation in which both `a` *and* `b` *are true. We can infer that*
`$`refutes`(r1,a, r2,b)`, `$`refutes`(r2,b, r1,a)`
are true, but
naf `$`**`defeated`**`(r1,a)`, naf `$`**`defeated`**`(r2,b)`
cannot both be true. This shows that `a` *and* `b` *cannot both be true. So* {a,b} *is not an answer set. Instead, there are two answer sets: in one* `a` *is true and in the other* `b` *is true.*

The intuition is: `r1` *and* `r2` *refutes each other, it cannot be decided which defeats the other, in one possible world* `r1` *defeats* `r2` *and* `a` *is true, and in the other* `r2` *defeats* `r1` *and* `b` *is true.*

For the reader who is familiar with LPDA, which is based on the well-founded semantics, we will go through the same example under the argumentation rules AT^{GCLP}, a sibling of AT^{AGCLP} mentioned earlier. The rules comprising AT^{GCLP} let us draw the

Fig. 2. Fish die-off example

```
/* Initial facts, and an “exclusion” constraint that fish count has a unique value */
occupies(trout,Squamish).
fishCount(s0,Squamish,trout,400).
#opposes(fishCount(?s,?r,?f,?C1), fishCount(?s,?r,?f,?C2)) :- ?C1 != ?C2.
/* Action/event description that specifies causal change, i.e., effect on next state */
@event fishCount(?s+1,?r,?f,0) :- occurs(?s,toxicDischarge,?r) ∧ occupies(?f,?r).
/* Persistence (“frame”) axiom */
@frame fishCount(?s+1,?r,?f,?C) :- fishCount(?s,?r,?f,?C).
/* Action axiom has higher priority than frame axiom */
#overrides(event,frame).
/* An action instance occurs */
occurs(s0+1,toxicDischarge,Squamish).
```

Fig. 3. Block moving example

```
/* moving a block from ?from to ?to, if ?to is free; after the move, ?from becomes free */
@move loc(?s+1,?blk,?to) :-
          move(?s,?blk,?from,?to) ∧ loc(?s,?blk,?from) ∧ naf loc(?s,?,?to).
@move neg loc(?s+1,?blk,?from) :-
          move(?s,?blk,?from,?to) ∧ loc(?s,?blk,?from) ∧ naf loc(?s,?,?to).
/* frame axioms: location of a block keeps the same */
@frame loc(?s+1,?blk,?pos) :- loc(?s,?blk,?pos).
@frame neg loc(?s+1,?blk,?pos) :- neg loc(?s,?blk,?pos).
/* each location is free, by default */
@dloc neg loc(?s,?blk,?pos).
/* 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 & initial state beats default location */
#overrides(move,frame).
#overrides(move,dloc).
#overrides(frame,dloc).
/* Facts: 16 squares. */
posn(square1). posn(square2). ... ... ... posn(square16).

/* initial state */
@state loc(0,block4,square7).
#overrides(state,dloc).
/* State 2: block4 moves from square7 to square3 */
move(2,block4,square7,square3).
```

Fig. 4. Cycle of **#overrides**

```
@r1 a.
@r2 b.
#opposes(a,b).
#overrides(r1,r2).
#overrides(r2,r1).
```

following conclusions:
$refutes(r1,a,r2,b),
$refutes(r2,b,r1,a),
$refuted(r1,a), *and*
$refuted(r2,b).
It now follows that:
$defeats(r1,a,r2,b),
$defeats(r2,b,r1,a)
are true. Consequently, the rules with tag-head pairs r1,a *and* r2,b *are both defeated, so both* a *and* b *are false in LPDA. This is somewhat in line with the behavior we saw from* AT^{AGCLP} *under ASPDA, but differences should have been expected, since there is always a unique well-founded model under LPDA.* □

Our last example illustrates the semantics of ASPDA on a number of simple "edge" cases, which are unlikely to be found in practice. The example shows that our semantics is quite reasonable even for such unusual *aspda*s.

Example 6 *Let an* aspda *consist of only one rule:*
@r a.
We will look at this aspda *under several different argumentation rulesets.*
With respect to the argumentation ruleset
$**defeated**(r, a).
our aspda *has one answer set where* $**defeated**(r, a) *is true and* a *false.*
For the argumentation ruleset
$**defeated**(r, a) :- a.
the above aspda *has no answer sets.*
Finally, if the argumentation ruleset is
$**defeated**(r, a) :- naf a.
then there are two answer sets:

- $**defeated**(r, a) *is true and* a *is false.*
- a *is true and* $**defeated**(r, a) *is false.* □

5. Comparison with Other Work

Although a great deal of work has been devoted to various theories of defeasible reasoning, only a few considered disjunctive information or tried to unify the different frameworks for such reasoning. The notable exceptions are the works [22,11,15,6,9], which had goals similar to ours. Due to the large volume of literature on defeasible reasoning, we will focus on the above works, since they are related to our work most closely. We refer the reader to a survey [12] for a discussion of the various individual theories of defeasibility.

Defeasible reasoning with disjunctive information in the propositional case was studied in [6]. Buccafurri et al. [9] introduced a variant of disjunctive logic programming with inheritance, called $DLP^{<}$. A key feature in such inheritance systems is overriding of the inherited information by more specific information, which can be viewed as a specialized form of defeasible reasoning. Nonmonotonic inheritance can be represented by means of argumentation rules, although we have not studied the extent to which this is possible in $DLP^{<}$.

The logic of prioritized defaults [22] also does not use the notion of argumentation rules, but it allows for multiple theories of defaults for different application domains. This is analogous to allowing argumentation rulesets to vary. However, defaults are defined via meta-theories and the semantics in [22] is given by meta-interpretation. What we call an "argumentation ruleset" is implicit in the meta-interpreters, and no independent model theory is given. In contrast, our approach abstracts all the differences between the various theories for defaults to the notion of an argumentation ruleset 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 [40] and answer sets (the present paper). It unifies the theories of Courteous Logic Programming, Defeasible Logic, Prioritized Defaults, and more.

Delgrande et al. [11] propose a framework for 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 [42,15].

Unlike ASPDA, the framework of Delgrande et al. 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 [11] has to do with the nature of the variable parts of the two approaches. In our case, the variable part is the argumentation ruleset, 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 [11], 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.[10] It is also important to note that each program transformation in [11] needs a compiler that contains hundreds of lines of Prolog code, while our approach requires no new software, and each argumentation ruleset typically contains 20-30 rules.

Eiter et. al. [15] set out to unify approaches to defeasible reasoning. Specifically, they present an adaptable meta-interpreter, which can be designed to simulate the approaches described in [8,42] among others. This framework is not as flexible as ASPDA and is fundamentally different from it: while ASPDA captures the essence of other approaches via argumentation rules, [15] captures these approaches in a less direct way, with the help of meta-interpretation.

The term "argumentation theory" was used to denote concepts that are related but significantly different from those studied in the present paper [7,17,34]. In these works, argumentation theories refer to proofs or sets of supporting premises rather than to rules that specify the notion of defeasibility. The focus of [7] is non-monotonic logic in general, while [17] is a procedural approach to defeasible reasoning. It is unclear whether these approaches can be captured as argumentation rules in our framework.

Argumentation theories were also used in a number of more closely related papers [35,36,27,14]. The focus of these works is development of the actual concepts that argumentation theories operate with. For instance, [35] uses Default Logic [39] to formalize the notions of defeat, defensible arguments, etc. Our work has a different focus in that we develop a general semantics for defeasible reasoning rather than dwelling on particular approaches to argumentation. The different argumentation rulesets (such as those in Section 4) are *examples* of the application of our general theory of defeasibility. These examples rely on some of the concepts that are analogous to those developed in [35,14]. For instance, the rulesets presented in Section 4 rely on the notion of defeated arguments, although those notions are not exactly the ones in [35,14].

Although defeasibility for disjunctive logic programs has been considered in restricted settings before [6,9], to the best of our knowledge, the present paper is the only work that studies the semantics of such logic programs in a general way. Defeasible disjunctive rules should not be confused with disjunctive logic programs under the answer-set semantics, as the latter does not explicitly represent defeasibility as a high-level concept but rather encodes it via default negation, not unlike the reduction described in Theorem 2.

[10] Note that argumentation rules can also encode rule application orderings.

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 rules but was based on the well-founded semantics. Apart from the model theoretic semantics, and the reduction theorems, 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 gave two examples of argumentation rulesets. One is an adaptation for stable models of generalized courteous argumentation rules given in [40] for well-founded models. This theory was used in most of the examples in this paper. The second argumentation ruleset was intended to show how ASPDA simulates other approaches to defeasible reasoning; in this case the defeasible logic of [1]. We gave a detailed analysis of the behavior of the two argumentation rulesets on a number of interesting examples and compared the results with the behavior that would have resulted if we used defeasibility under the well-founded semantics of [40].

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] G. Antoniou, D. Billington, and M. Maher. On the analysis of regulations using defeasible rules. In *Proc. 32nd Hawaii International Conference on Systems Science*, 1999.

[3] G. Antoniou, M. J. Maher, and D. Billington. Defeasible logic versus logic programming without negation as failure. *J. Log. Program.*, 42(1):47–57, 2000.

[4] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Ann. Math. Artif. Intell.*, 12(1-2):53–87, 1994.

[5] D. Billington, G. Antoniou, G. Governatori, and M. J. Maher. An inclusion theorem for defeasible logics. *ACM Trans. Comput. Log.*, 12(1):6, 2010.

[6] D. Billington and A. Rock. Propositional plausible logic: Introduction and implementation. *Studia Logica*, 67(2):243–269, 2001.

[7] A. Bondarenko, P. Dung., R. Kowalski, and F. Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93(1-2):63–101, 1997.

[8] 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.

[9] F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. *Theory and Practice of Logic Programming (TPLP)*, 2(3):293–321, 2002.

[10] 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.

[11] 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.

[12] 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.

[13] J. Dix, G. Gottlob, and V. Marek. Reducing disjunctive to non-disjunctive semantics by shift-operations. *Fundamenta Informaticae*, XXVIII(1/2):87–100, 1996.

[14] P. Dung, R. Kowalski, and F. Toni. Assumption-based argumentation. In I. Rahwan and G. Simari, editors, *Argumentation in Artificial Intelligence*, pages 199–218. Springer, Nyew York, 2009.

[15] 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.

[16] W. Faber. A polynomial reduction from aspda to asp. In M. Krötzsch and U. Straccia, editors, *RR*, volume 7497 of *Lecture Notes in Computer Science*, pages 213–216. Springer, 2012.

[17] A. García and G. Simari. Defeasible logic programming: an argumentative approach. *Theory Practice of Logic Programming*, 4(2):95–138, 2004.

[18] A. V. Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38:620–650, 1991.

[19] M. Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, pages 285–316. Elsevier, 2008.

[20] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[21] 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.

[22] 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.

[23] G. Governatori, A. ter Hofstede, and P. Oaks. Defeasible logic for automated negotiation. In *Proc. Fifth CollECTeR Conference on Electronic Commerce*, 2000.

[24] 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.

[25] B. Grosof, Y. Labrou, and H. Chan. A declarative approach to business rules in contracts: Courteous logic programs in xml. In *Proc. 1st ACM Conference on Electronic Commerce (EC-99)*. ACM Press, 1999.

[26] B. Grosof and S. Russell. Shift of bias as non-monotonic reasoning. In P. Brazdil and K. Konolige, editors, *Machine Learning, Meta-Reasoning, and Logics*. Kluwer Academic Publishers, Norwell, MA, 1990.

[27] N. Karacapilidis, D. Papadias, and T. Gordon. An argumentation based framework for defeasible and qualitative reasoning. In *Advances in Artificial Intelligence. XIIIth Brazilian Symposium on Artificial Intelligence*, pages 1–10. Springer, 1996.

[28] M. Kifer. FLORA-2: An object-oriented knowledge base language. The FLORA-2 Web Site. http://flora.sourceforge.net.

[29] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[30] 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.

[31] J. W. Lloyd and R. W. Topor. Making prolog more expressive. *J. Log. Program.*, 1(3):225–240, 1984.

[32] A. R. Morales, P. H. Tu, and T. C. Son. An extension to conformant planning using logic programming. In *IJCAI*, pages 1991–1996, 2007.

[33] L. Morgenstern. Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence*, 103:1–34, 1998.

[34] L. Pereira and A. Pinto. Reductio ad absurdum argumentation in normal logic programs. In *ArgNMR workshop at LPNMR*, pages 96–113, 2007.

[35] H. Prakken. An argumentation framework in default logic. *Annals of Mathematics and Artificial Intelligence*, 9(1-2):93–132, 1993.

[36] H. Prakken. A logical framework for modelling legal argument. In *ICAIL '93: 4th Int'l Conf. on Artificial Intelligence and Law*, pages 1–9. ACM, 1993.

[37] H. Prakken. *Logical Tools for Modelling Legal Argument. A Study of Defeasible Reasoning in Law*. Kluwer Academic Publishers, 1997.

[38] D. Reeves, B. Grosof, M. Wellman, and H. Chan. Towards a declarative language for negotiating executable contracts. In *Proceedings of the AAAI-99 Workshop on Arti cial Intelligence in Electronic Commerce (AIEC-99)*. MIT Press, 1999.

[39] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[40] H. Wan, B. Grosof, M. Kifer, P. Fodor, and S. Liang. Logic programming with defaults and argumentation theories. In *ICLP*, pages 432–448, 2009.

[41] H. Wan, M. Kifer, and B. Grosof. Defeasibility in answer set programs via argumentation theories. In *4th International Conference on Web Reasoning and Rule Systems (RR-2010)*, Brixen, Italy, September 2010. Springer.

[42] 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.