Defeasibility in Answer Set Programs with Defaults and Argumentation Rules

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 be seen as an attempt to unify 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. 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 among others. Combining the various theories of defeasible reasoning with other advances in logic-based knowledge representation, such as HiLog and F-logic, has also been a problem.
Our earlier work \cite{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 \textit{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 application 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 \cite{40}, was developed on the basis of the well-founded models \cite{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 \cite{20}, which has a very different semantics and is not capable 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 \cite{19}.

The present work takes the idea of LPDA further and introduces ASPDA—an analogous framework for defeasibility of \textit{disjunctive} logic rules through argumentation rules 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 \cite{16}. Then we study the class of \textit{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. \cite{14}. We briefly discuss the relationship in Section \ref{5}.

A preliminary report on this work appeared in \cite{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 \ref{2} illustrates defeasible reasoning under the answer-set semantics using the well-known Turkey Shoot example \cite{32}. Section \ref{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 \ref{4} gives two examples of argumentation rule sets for ASPDA. One is an adaptation of GCLP \cite{24,40} to ASPDA, a theory that is used in all examples throughout this paper. Another is an argumentation ruleset that behaves very similarly to Defeasible Logic \cite{1} except in edge cases. Sections \ref{5} and \ref{6} discuss related work and conclude the paper.

2. Motivating Example

The example in Figure \ref{1} is adapted from the Texas Turkey Shoot game example in \cite{32}. We use the usual syntax of logic programming with the only difference that rules are tagged with \texttt{@tag} symbols and head-disjunctions are allowed. Variables are prefixed with the symbol “\texttt{?}”.

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 \texttt{g1} and \texttt{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, \texttt{shoot(g1,1)} and \texttt{shoot(g1,2)} represent the actions of shooting the gun \texttt{g1} at time points 1 and 2. In the example, some of the rules have tags, e.g., \texttt{kpld} and \texttt{sh1}, and the predicate \texttt{#overrides} specifies priorities among some of these tagged rules.

We distinguish between the classical-logic-like \textit{explicit negation} \texttt{neg} and the \textit{default negation} \texttt{naf} (which in this paper will have the answer-set semantics). Literals \texttt{L} and \texttt{neg L} are assumed to be incompatible and

---

\footnote{Earlier \cite{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. \cite{14}.}
Wan, Kifer and Grosof / Defeasibility in Answer Set Programs with Defaults and Argumentation Rules

3

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(?T+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).

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 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 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. 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: \( \text{neg} \) for explicit negation and \( \text{naf} \) 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 “?”; 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 “?”; and a set of propositional constants, \( \text{t} \) and \( \text{f} \), which stand for \textit{true} and \textit{false}, respectively. We also assume the following order on these propositions: \( \text{f} < \text{t} \).

We use the standard notion of \textit{terms} in logic programming. \textit{Atomic formulas}, also called \textit{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.
- \(\text{not} A\) and \(\text{notf} A\), where \(A\) is an atomic formula.
- \(\text{notf} \text{not} A\), where \(A\) is an atomic formula.
- \(\text{notf} \text{notf} L\) and \(\text{notf} \text{notf} L\), where \(L\) is a literal; these are identified with \(L\).

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

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

\[
@r L_1 \lor \ldots \lor L_k : \text{Body}
\]

(1)

where \(r\) is a term, called the tag of the rule; \(L_1, \ldots, L_k (k \geq 0)\) are \(\text{notf}\)-free literals in \(L\), called the head literals of the rule; and \(\text{Body}\), called the body of the rule, is a conjunction of literals in \(L\). As is common in logic programming, we will often write \(A, B\) to represent the conjunction \(A \land B\). A rule tag is not a rule identifier; several rules can have the same tag.

A constraint is a special form of rule where \(f\) 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 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 \(L\) is a set of tagged rules in \(L\), which can be strict or defeasible. Sets or rules that do not have disjunctions in the head will be called non-disjunctive ASPDAs.

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.

ASPDAs 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 \(L\) be a logic language. An argumentation ruleset is a set, \(AT\), of strict rules in \(L\) of the form \(^{\text{strict}}\). We also assume that the language \(L\) includes a binary predicate, \$defeated\(_{AT}\), which may appear in the heads of some rules in \(AT\). When confusion does not arise, we will omit the subscript \(AT\).

An aspda \(P\) is said to be compatible with \(AT\) if \$defeated\(_{AT}\) does not appear in the rule heads in \(P\).

In an argumentation ruleset all rules are strict, by definition. The rules in \(AT\) will normally contain other predicates, besides \$defeated\(_{AT}\), that are used to specify how the rules in \(P\) 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

---

\(^{2}\) This is easy to generalize to allow Lloyd-Tohor extensions \[21\].

\(^{3}\) This is easy to generalize to allow Lloyd-Tohor extensions \[21\].

\(^{4}\) 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).
prefixed with the $-sign, such as \$\text{conflict}$, and will be in normal font, except \$\text{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 \text{body}, will also be set in bold. The predicates defined and used only by user programs 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 \texttt{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 \texttt{aspda} to use different sets argumentation rules \( AT \) with different \$\text{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 \cite{23} supports multiple argumentation rule sets.

### 3.1. Interpretations and Models

**Definition 5 (Herbrand universe and Herbrand base)** Let \( \mathcal{P} \) be an \texttt{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 \texttt{naf}-free literals that can be constructed using the predicates in \( \mathcal{L} \).

**Definition 6 (Herbrand interpretation)** A **Herbrand interpretation**, \( I \), is a subset of \( \mathcal{B} \), i.e., a set of ground \texttt{naf}-free literals. In addition, \( I \) must contain \( t \) and must not contain \( f \).

An interpretation is **inconsistent relative to** an atom \( A \) if both \( A \) and \texttt{neg} \( A \) are in \( I \). Otherwise, \( I \) 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 this explicitly.

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

**Definition 7 (Truth valuation)** Let \( I \) be a Herbrand interpretation, \( L \) a ground \texttt{naf}-free literal, and let \( F, G \) be ground formulas. We define **truth valuations** that map formulas to \{t,f\} as follows:

- If \( L \in I \), \( I(L) = t \) if \( L \in I \), \( I(L) = t \) otherwise.
- \( I(\texttt{naf} L) \) = \( \sim I(L) \), where \( \sim \) is the negation of \( I(L) \). Recall that \( f < t \).
- If \( F \land G \) is a ground rule, then we write \( I(F) \land I(G) \) is satisfied in \( I \).
- If \( F \lor G \) is a ground rule, then we write \( I(F) \lor I(G) \) is satisfied in \( I \).
- For a strict rule \( @r \ R : - G \), we define \( I(R) \) is satisfied in \( I \).

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 \( @r \ L_1 \lor ... \lor L_k : - G \), we define \( I(L_1 \lor ... \lor L_k) = t \) if and only if \( I(L_1) \lor ... \lor I(L_k) \) is satisfied in \( I \).

That is, a defeasible rule of the above form is true if either (i) the rule 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, \( I \) an interpretation, and \( I(F) = t \), then we write \( I \models F \) and say that \( I \) is a model of \( F \) or that \( F \) is satisfied in \( I \).

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

We write \( I \models \mathcal{P} \) if \( I \models R \) for every \( R \in \mathcal{P} \).

**Definition 9 (Model of \texttt{aspda} w.r.t. argumentation theory)** Given an \texttt{aspda} \( \mathcal{P} \), an argumentation ruleset \( AT \), and an interpretation \( \mathcal{M} \), we say that \( \mathcal{M} \) is a model of \( \mathcal{P} \) with respect to the argumentation ruleset \( AT \) (or a model of \( \mathcal{P}, AT \), for short), written as \( \mathcal{M} \models (\mathcal{P}, AT) \), if \( \mathcal{M} \models \mathcal{P} \) and \( \mathcal{M} \models AT \).

**Definition 10 (Minimal model)** A minimal model of \( (\mathcal{P}, AT) \) is a model \( \mathcal{M} \) of \( (\mathcal{P}, AT) \) such that no proper subset of \( \mathcal{M} \) is a model of \( (\mathcal{P}, AT) \).

### 3.2. Stable Model and Answer-set Semantics

In this section, we extend the stable model semantics \cite{20} and the answer-set semantics \cite{19} to ASPDA. We start with non-disjunctive \texttt{aspdas} and stable models.
Definition 11 (ASPDA quotient, non-disjunctive case)
Let $Q$ be a non-disjunctive aspda, and let $J$ be a Herbrand interpretation for $Q$. The ASPDA quotient$^6$ of $Q$ by $J$, written as $\frac{Q}{J}$, is defined by the following sequence of steps:

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

Note that $\frac{Q}{J}$ 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 $M$ is a stable model of a non-disjunctive aspda $P$ with respect to the argumentation theory $AT$, if $M$ is a minimal Herbrand model of $\frac{P \cup AT}{M}$. Note that $\frac{P \cup AT}{M}$ 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 [19]. □

The next theorem shows that non-disjunctive aspdas can be implemented using ordinary logic programming systems that support the stable model semantics (e.g., DLV [20]).

Theorem 1 (Reduction for stable model semantics)
Let $P$ be a non-disjunctive aspda and $AT$ an argumentation ruleset. Then the following two sets coincide:

- The set of stable models of $P$ with respect to $AT$.
- The set of stable models of the ordinary logic program $P' \cup AT'$, where $P'$ is obtained from $P$ by converting every defeasible rule $(\@r L : - Body) \in P$ into the plain rule of the form $L : - Body \land$ naf $\$defeated(r, L)$ 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 $P \cup AT$. According to Definition [11] $\frac{P \cup AT}{S}$ is obtained through the following steps:

1. Delete every rule $R \in P \cup AT$ 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 $(\@r L : - Body) \in P \cup AT$ such that $\$defeated(r, L) \in S$.
3. Remove all naf-literals from the remaining rules in $P \cup AT$.
4. Remove all tags from the remaining tagged rules in $P \cup AT$.

Note that this makes $\frac{P \cup AT}{S}$ into an ordinary logic program.

According to the definition of Quotient in the ordinary stable model semantics [20], the quotient of $P' \cup AT'$ by $S$, is obtained through the following steps:

1. Delete every rule $R \in P' \cup AT'$ 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 $P' \cup AT'$.

From the above it can be safely inferred that the ASPDA quotient $\frac{P \cup AT}{S}$ is the same set of ordinary logic rules as the (ordinary) quotient of $P' \cup AT'$ by $S$. For instance, consider a defeasible rule $\@r L : - Body$ in $P$. If $\$defeated(r, L) \in S$, this rule will be deleted by the process of construction of $\frac{P \cup AT}{S}$. The corresponding rule

$L : - Body \land$ naf $\$defeated(r, L)$

will be deleted by the construction of the (ordinary) quotient of $P' \cup AT'$ by $S$.

The claim now follows from the above and the definitions of stable models in ASPDA and in the classical case (Definition [112] 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 aspdas 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:

\[
\begin{align*}
\@r_1 &\ a \lor b \lor c. \\
\@r_2 &\ d \lor e.
\end{align*}
\]
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:

\[
\#\text{opposes}(a, d).
\]

\[
\#\text{opposes}(a, e).
\]

\[
\#\text{opposes}(b, e).
\]

Suppose, in addition, that rule \(r_1\) has a higher priority than \(r_2\), which we specify using the fact

\[
\#\text{overrides}(r_1, r_2).
\]

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 \(r_1\) overrides \(r_2\), the proposition \(a\) makes \(r_1\) true, and \(a\) is incompatible with both heads in rule \(r_2\).

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:

\[
\$\text{defeated}(r_2, d) :- \\
\#\text{overrides}(r_1, r_2) \land \#\text{opposes}(a, d) \land a.
\]

\[
\$\text{defeated}(r_2, e) :- \\
\#\text{overrides}(r_1, r_2) \land \#\text{opposes}(a, e) \land a.
\]


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

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

**Proof:** Let \(S\) be a Herbrand interpretation of \(P \cup AT\). By Definition [13] \(\frac{P \cup AT}{S}\) is constructed by the following steps:

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

Note that \(\frac{P \cup AT}{S}\) is an ordinary disjunctive logic program. For future reference, let us denote it \(Q_1\).

By definition of the quotient in the ordinary answer set semantics [19], the quotient of \(P' \cup AT'\) by \(S\), is obtained from \(P' \cup AT'\) by these steps:
(i) Delete every rule $R \in \mathcal{P}' \cup \mathcal{AT}'$ that has a literal of the form $\text{naf } A$ in $R$’s body where $A \in S$;

(ii) Remove all $\text{naf}$-literals from the remaining rules.

Let us denote the resulting logic program with $Q_2$. We will call the rules in $Q_1$ and $Q_2$ the reducts of the original rules in $\mathcal{P} \cup \mathcal{AT}$ and $\mathcal{P}' \cup \mathcal{AT}'$, respectively.

Now consider a rule $R \left( \forall r \, L_1 \lor \ldots \lor L_n : - \text{Body} \right) \in \mathcal{P}$.

- If there is a literal of the form $\text{naf } A$ in $R$’s body and $A \in S$, $R$ would be deleted and its reduct will be neither in $Q_1$ nor $Q_2$.

- If no such literal $\text{naf } A$ exists in $R$ then $\text{Body}$ of the reduct of $R$ does not contain $\text{naf}$-literals. Let $K_0 = \{1, \ldots, n\}$ be a subset such that $S \models \text{naf } \text{defeated}(r, L_i)\), for all $i \in K_0$, and $S \models \text{naf } \text{defeated}(r, L_j)$, for all $j \notin K_0$. Then, if $K_0 \neq \{\}$,
  - $Q_1$ would contain the rule $\forall i \in K_0 \, L_i : - \text{Body} \leftarrow \text{the reduct of } R$.
  - $\mathcal{P}' \cup \mathcal{AT}'$ would contain a set of rules of the form

$$
\forall i \in K_0 \, L_i : - \text{Body} \land \\
\forall i \in K_0 \, \text{naf } \text{defeated}(r, L_i) \land \\
\forall j \notin K_0 \, \text{defeated}(r, L_j)
$$

for each non-empty subset $K \subseteq N = \{1, \ldots, n\}$.

During the construction of $Q_2$, after step (i) the only remaining rules will be of the form

$$
\forall i \in K \, L_i : - \text{Body} \land \\
\forall i \in K \, \text{naf } \text{defeated}(r, L_i) \land \\
\forall j \notin N-K \, \text{defeated}(r, L_j)
$$

for each $K$ such that $K \subseteq K_0$. After step (ii), the reducts of $R$ that will remain in $Q_2$ would be:

$$
\forall i \in K \, L_i : - \text{Body} \land \\
\forall j \notin N-K \, \text{defeated}(r, L_j)
$$

for each $K$ such that $K \subseteq K_0$. Among these rules, only one rule, $\forall i \in K_0 \, L_i : - \text{Body} \land \\
\forall j \notin K_0 \, \text{defeated}(r, L_j)$, can possibly have a body entailed by $S$. Furthermore, $S$ entails this rule if and only if $S$ entails $\forall i \in K_0 \, L_i : - \text{Body}$, which is a reduct of $R$ in $Q_1$.

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

- $Q_1$ has no reducts of $R$. So, the entire rule is deleted in step 2 (of ASPDA quotient).

- $\mathcal{P}' \cup \mathcal{AT}'$ must contain the rules of the form

$$
\forall i \in K \, L_i : - \text{Body} \land \\
\forall i \in K \, \text{naf } \text{defeated}(r, L_i) \land \\
\forall j \notin N-K \, \text{defeated}(r, L_j)
$$

for each non-empty subset $K \subseteq N = \{1, \ldots, n\}$. Each such rule contains at least one literal $\text{naf } \text{defeated}(r, L_i)$ in the rule body. Since $K_0 = \{\}$ implies that all such literals are false in $S$, step (i) in the construction of $Q_2$ eliminates all the above rules. So, neither $Q_1$ nor $Q_2$ will have any reducts of $R$.

It can now be seen that $S$ is a minimal Herbrand model of $Q_1$ if and only if $S$ is a minimal Herbrand model of $Q_2$. In other words, $S$ is an answer set for the aspda $\mathcal{P}$ with respect to $\mathcal{AT}$ if and only if $S$ is an answer set for the ordinary logic program $\mathcal{P}' \cup \mathcal{AT}'$. $\Box$

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 $L_1 \lor \ldots \lor L_n : - \text{Body}$ with $n$ new rules

$$
L_i : - \text{Body} \land \forall 1 \leq j \leq n, \, j \neq i \text{naf} L_j
$$

(2)

where $1 \leq i \leq n$. We will use shift($\mathcal{P}$) to denote such transformation of a (non-de defeasible) disjunctive logic program. For example, consider a program con-
sisting of one rule \( p \lor q \lor s : \rightarrow \text{body} \), the shifting of the program is
\[
\begin{align*}
p & : \rightarrow \text{body} \land \neg q \land \neg s, \\
q & : \rightarrow \text{body} \land \neg p \land \neg s, \\
s & : \rightarrow \text{body} \land \neg q \land \neg p.
\end{align*}
\]
Ben-Eliyahu and Dechter [4] have shown that the above shifting transformation is an equivalence transformation for so-called head-cycle free programs.

We reproduce that definition below adjusting it for disjunctive ASPs.

**Definition 15** [4] The dependency graph \( G_\mathcal{P} \), of a ground ASP \( \mathcal{P} \), 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 ASP 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 ASP consisting only of the rule
\[
@r \ p \lor q \lor s : \rightarrow \text{body}
\]
is head-cycle free.

Under certain restrictions, the head-cycle free property for \( \mathcal{P} \cup \mathcal{A} \mathcal{T} \) can be reduced to head-cycle freedom for \( \mathcal{P} \). For example, if the literals that appear in rule heads in \( \mathcal{A} \mathcal{T} \) do not appear in any rule body in \( \mathcal{P} \), and \( \mathcal{A} \mathcal{T} \) is non-disjunctive, then \( \mathcal{P} \cup \mathcal{A} \mathcal{T} \) is head-cycle free if and only if \( \mathcal{P} \) is head-cycle free. This is satisfied in the argumentation ruleset \( \mathcal{A} \mathcal{T} \mathcal{D} \mathcal{P} \mathcal{L} \mathcal{P} \) in Section 4.2. It is also satisfied in the argumentation ruleset \( \mathcal{A} \mathcal{T} \mathcal{D} \mathcal{P} \mathcal{L} \mathcal{P} \mathcal{S} \) 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 programing exists, and an equivalence result holds for disjunctive ASPs.

**Definition 16** Let \( \mathcal{P} \) be a disjunctive ASP. We define the t-shifting of \( \mathcal{P} \), \( t\text{-shift}(\mathcal{P}) \), as a non-disjunctive ASP obtained from \( \mathcal{P} \) by replacing each rule of the form (\( @r L_1 \lor ... \lor L_n : \rightarrow \text{body} \)) in \( \mathcal{P} \) with \( n \) new rules
\[
@r \ L_i : \rightarrow \text{body} \land \bigwedge_{1 \leq j \leq n, j \neq i} \neg \text{lit}(L_j).
\]
where \( 1 \leq i \leq n \).

Surprisingly, it turns out that \( t\text{-shift}(\mathcal{P}) \) is not equivalent to \( \mathcal{P} \) even for head-cycle free ASPs. To see this, consider the following rule set, \( \mathcal{P}^{ex} \):
\[
\begin{align*}
@r1 \ a & : \neg b \land \neg c. \\
@r2 & : d. \\
@r3 & : c.
\end{align*}
\]
Suppose that the associated argumentation rules imply \#defeated(r1, c) and does not imply any other \#defeated(...) facts involving the above rules. Then \( \mathcal{P}^{ex} \) would have the following answer sets: \{a, d, c\} and \{b, d, c\}. In contrast, the above t-shifting transformation yields the following non-disjunctive ASP, \( t\text{-shift}(\mathcal{P}^{ex}) \):
\[
\begin{align*}
@r1 \ a & : \neg b \land \neg c. \\
@r1 \ b & : \neg a \land \neg c. \\
@r1 \ c & : \neg a \land \neg b. \\
@r2 & : d. \\
@r3 & : c.
\end{align*}
\]
which has only one answer set: \{d, c\} with respect to the argumentation ruleset. This shows that \( t\text{-shift} \) is not an equivalence transformation under ASPDA.

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

**Definition 17** The ASPDA shifting of an ASP \( \mathcal{P} \), written as \( \text{aspda}\_\text{shift}(\mathcal{P}) \), is a non-disjunctive ASP obtained from \( \mathcal{P} \) by replacing each strict rule with its t-shifting and replacing each defeasible rule of the form \( @r \bigwedge_{1 \leq i \leq n, j \neq i} \text{lit}(L_i) \rightarrow \text{Body} \) in \( \mathcal{P} \) with \( n \) new defeasible rules and \( 2n \) new strict rules as follows:
\[
\begin{align*}
@r \ L_1 : & \rightarrow \text{Body} \land \bigwedge_{1 \leq i \leq n} \text{lit}(L_i). \\
\text{lit}(r, L_1) & : \rightarrow \neg \text{lit}(L_i). \\
\text{lit}(r, L_1) & : \rightarrow \#\text{defeated}(r, L_1).
\end{align*}
\]
where \( 1 \leq i \leq n \). Here \( \text{lit}(r, L_i) \), \( 1 \leq i \leq n \), are literals of the form \( \text{newsym}_i(Vars_i) \), where \( \text{newsym}_i \) is a fresh predicate name that depends only on \( r \) and \( L_i \), while \( Vars_i \), the argument vector of the literal, is a vector of variables that occur in \( r \) and \( L_i \). We omit the rule tags for strict rules here.

**Theorem 3** Let \( \mathcal{P} \) be an ASP and let \( \mathcal{A} \mathcal{T} \) be an argumentation ruleset such that \( \mathcal{P} \cup \mathcal{A} \mathcal{T} \) is head-cycle free. There is a one-to-one relationship between the answer sets of \( \mathcal{P} \) with respect to \( \mathcal{A} \mathcal{T} \) and the answer sets of \( \text{aspda}\_\text{shift}(\mathcal{P}) \) with respect to \( t\text{-shift}(\mathcal{A} \mathcal{T}) \).

---

\(^7\) The works [13]21 developed similar shifting techniques.
Namely, a Herbrand interpretation $S$ is an answer set of $P$ with respect to $AT$ if and only if $f(S)$ is an answer set of \( \text{aspda\_shift}(P) \) with respect to $t\_shift(AT)$, where $f(S) = S \cup \{ \text{lit}(r, L) \mid P \}$ contains a rule with tag $r$ and with $L$ in its head (possibly as a disjunct), so that either $\text{defeated}(r, L) \in S$ or $L \notin S$.

**Proof:** The proof consists of establishing five equivalences, which we denote $\iff_1, \ldots, \iff_5$.

- $S$ is an answer set of $P$ with respect to $AT$ $\iff_1$
- $S$ is a minimal Herbrand model of $Q_1 = P \cup AT \setminus S$ $\iff_2$
- $S$ is a minimal Herbrand model of $Q_2 = S = \text{shift}(P \cup AT) \setminus S$ $\iff_3$
- $f(S)$ is a minimal Herbrand model of $Q_4 = \text{aspda\_shift}(P) \cup t\_shift(AT) \setminus f(S)$ $\iff_4$
- $f(S)$ is an answer set of \( \text{aspda\_shift}(P) \) with respect to $t\_shift(AT)$ $\iff_5$

In proving each equivalence, we will choose an arbitrary defeasible rule $R$ of the form $\forall x \ L_1 \lor \ldots \lor L_n : -\text{Body}$ in $P$ and an arbitrary strict rule $T \in P \cup AT$, 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 \text{naf}-literals (they are evaluated away in the quotients on both sides).

Let $K_0 = \{ k \mid S \vdash \text{naf defeated}(r, L_k), 1 \leq k \leq n \}$ and let $K_1$ be $\{ l \mid L_k \in S, 1 \leq k \leq n \}$.

($\iff_1$): This follows from Definition 13.

($\iff_2$): By definition, every defeasible rule $R \in P$ gives rise to the following single rule $R_1$ in the quotient $Q_1$:

$$
\bigvee_{i \in K_0} L_i : -\text{Body}
$$

(4)

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

The strict rules $T \in P \cup AT$ give rise to $T_1$ in $Q_1$, where $T_1$ has the same head and body as $T$ but the tag is stripped off. By definition, all rules in $Q_1$ are either $R_1$s or $T_1$s and are obtained in the above way. So, $Q_1$ consists of the rules of the form (4) or of the strict rules from $P \cup AT$ that lost their tag.

$Q_2$ is constructed from $Q_1$ via shifting of ordinary (non-defeasible) disjunctive rules. A rule $R_1$ of the form (4) produces $|K_0|$ rules of the form

$$
L_1 : -\text{Body} \land \bigwedge_{j \in K_0, j \neq i} \text{naf}L_j
$$

(5)

for $i \in K_0$. The strict rule $T_1 \in Q_1$ gives rise to the rules $\text{shift}(T_1)$.

Since, by assumption, $Q_1$ does not contain \text{naf}-literals, $S$ is a minimal Herbrand model of $Q_1$ iff $S$ is an answer set of $Q_1$. Observe that:

- $Q_1$ is an ordinary (non-defeasible) disjunctive logic program,
- $P \cup AT$ is head-cycle free, so $Q_1 = P \cup AT \setminus S$ is head-cycle free.

Therefore, as shown in [4,13,21], $S$ is an answer set of $Q_1$ iff $S$ is an answer set of $Q_2 = \text{shift}(Q_1)$.

$Q_2$ contains no rules other than those in $S$ and $\text{shift}(T_1)$.

($\iff_3$): $Q_3 = \frac{S}{f(S)}$ is constructed according to Definition 13. Strict rules in $Q_3$ all have the form $\frac{\text{shift}(T_1)}{S}$ and defeasible rules are obtained as follows:

3-a. If $|K_1 \cap K_0| \geq 2$, the rules of the form (5) yield nothing in $Q_3$. 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 $\{ L_1 : -\text{Body} \mid i \in K_1 \cap K_0 \}$ in $Q_3$. 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 \text{naf}-literals in the remaining rule are deleted in Step 3.

3-c. If $|K_1 \cap K_0| = 0$, (5) yields the rules $\{ L_1 : -\text{Body} \mid i \in K_0 \}$ in $Q_3$. This is because the rules in (5) such that $i \notin K_0$ are deleted in Step 2 of Defi-
nition [13] while the naf-literals in the remaining rules are deleted in Step 3.

\((\Leftarrow_5)\): 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.

\((\Leftarrow_4)\): \(Q_3\) is divided into two subsets: \(Q_3^{df}\), which consists of rules obtained from the defeasible rules in \(Q_2\), and \(Q_3^{st}\) which consists of rules obtained from the strict rules in \(Q_2\).

\(Q_4\) is divided into \(Q_4^{df}\) and \(Q_4^{st}\) the same way. \(Q_4\) is obtained from \(asnda_{\text{shift}}(P) \cup t_{\text{shift}}(AT)\) by applying the steps in Definition [11]. Recall that each defeasible rule \(R \in P\) gives rise to a collection of rules of the form \(3\) in \(asnda_{\text{shift}}(P)\), which in \(Q_4\) become

\[
L_i : \neg \text{Body} \land \bigwedge_{1 \leq j \leq n, j \neq i} \text{lit}(r, L_j).
\]

for each \(i \in K_0\);

\[
\text{lit}(r, L_j) : = \neg \text{defeated}(r, L_j).
\]

for each \(1 \leq j \leq n\);

\[
\text{lit}(r, L_j).
\]

for each \(j \notin K_1\). All these rules constitute \(Q_4^{df}\).

Each strict rule \(T \in P' \cup AT\) gives rise to the set of rules \(t_{\text{shift}}(T)\) in \(Q_4\). These rules constitute \(Q_4^{st}\).

The difference between \(t_{\text{shift}}(T)\) used in \(Q_4\) and \(\text{shift}(T_1)\) used in \(Q_3\) is that \(T\) has a tag while \(T_1\) does not. The quotient operation removes tags, so \(t_{\text{shift}}(T) = \text{shift}(T_1)\). Because every rule in \(Q_4^{st}\) is of the form \(t_{\text{shift}}(T)\) for some \(T\) and every rule in \(Q_3^{st}\) has the form \(\text{shift}(T_1)\) for some \(T_1\) (which is obtained from \(T\) by tag removal), we have:

\[
f(S) \text{ is a Herbrand model of } Q_4^{df} \text{ iff } S \text{ is a Herbrand model of } Q_3^{st}.
\]

Now consider the defeasible rules in \(Q_4^{df}\). Since \([7]\) and \([8]\) are the only rules that define \(\text{lit}(r, L_j)\), it follows that \(f(S) \not\models \text{lit}(r, L_j), \forall j \in K_1 \cap K_0\), and \(f(S) \models \text{lit}(r, L_j), \forall j \notin K_1 \cap K_0\). Under \(f(S)\),

4-a. If \(|K_1 \cap K_0| = 1\), the rules in \([6]\) yield nothing in \(Q_4\), since for each \(i \in K_0\), every rule \([6]\) has some body literal \(\text{lit}(r, L_j)\) such that \(f(S) \not\models \text{lit}(r, L_j)\).

4-b. If \(|K_1 \cap K_0| = 1\), the rules in \([6]\) give rise to a single rule \(L_1 : \neg \text{Body}, \forall i \in K_1 \cap K_0\). Indeed, any rule in \([6]\) such that \(i \notin K_1 \cap K_0\) has some body literal \(\text{lit}(r, L_j)\) such that \(f(S) \not\models \text{lit}(r, L_j)\).

4-c. If \(|K_1 \cap K_0| = 0\), \([6]\) gives rise to the following rules \(\{L_1 : \neg \text{Body} : i \in K_0\}\), which is obtained by the same argument as before.

Every rule in \(Q_4^{df}\) comes from \([6]\) or \([7]\) or \([8]\) and all the rules of the form \([7]\) and \([8]\) are satisfied in \(f(S)\). By comparing \([3a],[3b],[3c]\) with \([4a],[4b],[4c]\), it follows that

\[
f(S) \text{ is a Herbrand model of } Q_4^{df} \text{ iff } S \text{ is a Herbrand model of } Q_3^{st}.
\]

From \([10]\) and \([9]\) we obtain

\[
f(S) \text{ is a Herbrand model of } Q_4 \text{ iff } S \text{ is a Herbrand model of } Q_3.
\]

To complete the proof for the equivalence \((\Leftarrow_4)\), it remains to show that \(f(S)\) is a minimal model of \(Q_4\) if and only if so is \(S\) for \(Q_3\).

Minimality of \(f(S)\): if \(S\) is a minimal Herbrand model of \(Q_3\), then \(\forall A \in f(S), f(S) - \{A\} \not\models \text{a rule in } Q_4\) if and only if so is \(S\) for \(Q_3\).

Minimality of \(S\): if \(f(S)\) is a minimal Herbrand model of \(Q_4\), then for any \(A \in S, S - \{A\} \not\models \text{a rule in } Q_3\). If it were a model then, by the previously established correspondence between the rules in \(Q_2\) and \(Q_4\), there is a rule \(R_2 \in Q_4\) of the form \([6]\) or \(t_{\text{shift}}(T)\) which, by construction, must be such that \(f(S) - \{A\} \not\models R_2\).

- if \(A = \text{lit}(r, L)\) for some \(r\) and \(L\), there must be some rule \(R\) of the form \([4a]\) or \([4c]\) such that \(f(S) - \{A\} \not\models R\), so \(f(S)\) also cannot be a model of \(Q_4\) in this case.

Minimality of \(S\): if \(f(S)\) is a minimal Herbrand model of \(Q_4\), then for any \(A \in S, S - \{A\} \not\models \text{a rule in } Q_3\). If it were a model then, by \([11]\), \(f(S - \{A\}) \subset f(S)\) must be a Herbrand model of \(Q_4\), contrary to the assumption that \(f(S)\) is a minimal Herbrand model of \(Q_4\).

This concludes the proof of \((\Leftarrow_4)\) and of the theorem.

\(\square\)

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 $A_{\text{GCLP}}$. It is this argumentation ruleset that was tacitly assumed in all the earlier examples in this paper.

In $A_{\text{GCLP}}$, the predicate $\text{defeated}$, which plays a key role in the semantics of aspdas, is defined in terms of the predicates $\text{opposes}$ and $\text{overrides}$. 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 $\text{opposes}$.

The $\text{defeated}$ predicate is defined as follows: A rule is defeated if it is refuted by some other unde¬feated 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.

$\text{defeated}(T, L) := \neg \text{defeats}(T', L', T, L).

The auxiliary predicate $\text{defeats}$ is defined as follows:

$\text{defeats}(T_1, L_1, T_2, L_2) :=
\neg \text{refutes}(T_1, L_1, T_2, L_2)∧\text{naf} \text{defeated}(T_1, L_1)∧\text{naf} \text{strict}(T_2, L_2).

The predicate $\text{strict}$ is used here to distinguish strict rules from the defeasible ones. The predicate $\text{refutes}$ 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:

$\text{refutes}(T_1, L_1, T_2, L_2) :=
\neg \text{conflict}(T_1, L_1, T_2, L_2)∧\text{?L}_1∧\text{#overrides}(T_1, L_1, T_2, L_2).

The definition of a conflict between two rules, re¬presented by the predicate $\text{conflict}$ above, relies in turn on the notion of a candidate. A candidate rule-instance is one whose body is true in the knowledge base:

$\text{candidate}(T, L) := \text{body}(T, L, ?B)∧?B.$

Here the meta-predicate $\text{body}$ 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:

$\text{conflict}(T_1, L_1, T_2, L_2) :=
\text{candidate}(T_1, L_1)∧\text{candidate}(T_2, L_2)∧\text{#overrides}(L_1, L_2).

Recall that the $\text{opposes}$ information is supplied by the knowledge engineer. However, argue¬mentation rules may include additional background axioms. In our case, $A_{\text{GCLP}}$ supplies the following background axioms for $\text{opposes}$:

$\text{opposes}(L_1, L_2) := \neg \text{opposes}(L_2, L_1).$

$\text{opposes}(L, \neg L).$

$\neg ?L_1 ∧ ?L_2 ∧ \text{#opposes}(L_1, L_2).$

The first is a symmetry axiom that states that opposi¬tion is a reciprocal relation. The second axiom states that literal 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 $\text{overrides}$ is also mostly defined by the knowledge engineer. However, $A_{\text{GCLP}}$ also supplies a background axiom that establishes preference for strict rules over defeasible ones:

$\text{overrides}(T_1, L_1, T_2, L_2) :=
\text{strict}(T_1, L_1)∧\text{naf} \text{strict}(T_2, L_2).

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:

$\text{overrides}(T_1, L_1, T_2, L_2) :=
\text{overrides}(T_1, L_1)∧\text{head}(T_2, L_1)∧\text{head}(T_2, L_2).

Here $\text{head}$ is a meta-predicate that relates tags to the heads of the rules labeled with those tags. The body¬occurrence of $\text{overrides}$ is the overriding relation over tags and the head occurrence is the overriding relation over tag-head pairs.

Similarly, $\text{strict}$ is also often specified over tags and the following axiom relates that to strictness at the
level of tag-head pairs.

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

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: \{a\}, {b, d}, {c, d}, {c, e}.

4.2. Defeasible Logic [1]

Our second argumentation ruleset was inspired by a version of Defeasible Logic under the stable model semantics, as defined in [1][3] and is designed to behave very similarly except in edge cases.

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 $\#\text{strict}$ and $\#\text{defeater}$. Other important restrictions in [1] are that it does not support disjunctions in the rule heads; opposition among literals is limited to $p$ and $\text{neg} \ p$, 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].

$$\$\text{defeated}(?T, ?L) : -$$  
$$\$\text{conflict}(?T, ?T') \land$$  
$$\text{head}(?T', ?L') \land \$\text{definitely}(?L').$$

$$\$\text{defeated}(?T, ?L) : - \#\text{defeater}(?T).$$

$$\$\text{defeated}(?T, ?L) : - \$\text{overruled}(?T).$$

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

The predicate $\$\text{definitely}$ is defined as follows:

$$\$\text{definitely}(?L) : -$$  
$$\#\text{strict}(?T) \land \text{head}(?T, ?L) \land$$  
$$\text{body}(?T, ?B) \land \text{each_definite}(?B).$$

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 $\$\text{definitely}(?B)$ is true or when $?B$ is bound to a conjunction, conj, and $\$\text{definitely}(c)$ is true for each conjunct $c \in \text{conj}$. 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 $\$\text{definitely}(?L)$.

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

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

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

$$\$\text{overruled}(?T) : -$$  
$$\$\text{conflict}(?T, ?T') \land \$\text{candidate}(?T') \land$$  
$$\text{naf}\$\text{refuted}(?T').$$

$$\$\text{refuted}(?T') : -$$  
$$\$\text{conflict}(?T, ?T') \land \$\text{candidate}(?T) \land$$  
$$\#\text{overrides}(?T, ?T') \land \text{naf}\#\text{defeater}(?T').$$

$$\$\text{conflict}(?T, ?T') : -$$  
$$\text{head}(?T, ?L) \land \text{head}(?T', \text{neg} ?L).$$

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 how priorities over the rules are specified. For instance, $AT^{AGCLP}$ is more general than $AT^{DL}$ in the sense that tags in user programs 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 $\$\text{defeated}$ 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\textsuperscript{AGCLP} argumentation rules, this example set has two answer sets. One is \{\texttt{neg loaded}(g1,1), \texttt{loaded}(g2,1), \texttt{neg alive}(3)\} and the other is \{\texttt{loaded}(g1,1), \texttt{neg loaded}(g2,1), \texttt{neg alive}(3)\}. Thus, AT\textsuperscript{AGCLP} yields the expected result.

As to the AT\textsuperscript{DL} argumentation rules and the logic in \cite{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\textsuperscript{DL} yields the same result as AT\textsuperscript{AGCLP}. \hfill \Box

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

Here both AT\textsuperscript{AGCLP} and AT\textsuperscript{DL} lead to the same conclusion:

\begin{verbatim}
{ fishCount(s0+1, Squamish, trout, 400),
  fishCount(s0+2, Squamish, trout, 0) }
\end{verbatim}

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 \cite{40}—a sibling of ASPDA developed for the well-founded semantics—if we use either the very same argumentation ruleset AT\textsuperscript{DL}, which we used here, or AT\textsuperscript{GCLP}, a ruleset analogous to AT\textsuperscript{AGCLP} but designed for the well-founded semantics \cite{40}.

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

Example 4 \cite{40} Figure 3 specifies part of a game where blocks are moved from square to square on a board.

The argumentation rules AT\textsuperscript{AGCLP} under ASPDA and AT\textsuperscript{GCLP} under LPDA both give the same expected result in this case:

\begin{verbatim}
{ loc(0, block4, square7),
  loc(1, block4, square7),
  loc(3, block4, square3) }\hfill \Box
\end{verbatim}

Again, AT\textsuperscript{DL} does not handle this example directly, since the syntax in \cite{11} does not include \texttt{neg}. However, \cite{3} shows that \texttt{naf} can be simulated using a transform that relies on \texttt{neg} only. Under this transform, AT\textsuperscript{DL} yields the same result as the other two theories. \hfill \Box

Example 5 Figure 4 shows a scenario where a cycle exists in the \#\texttt{overrides} relation between a pair of opponents.

Under ASPDA, AT\textsuperscript{DL} yields an answer set in which both \texttt{a} and \texttt{b} are true. Indeed, one can verify that the following literals are true:
\begin{verbatim}
$\texttt{refuted}(r1,a),$
$\texttt{refuted}(r2,b),$
$\texttt{naf}\$_{\texttt{defeated}}(r1,a),$
$\texttt{naf}\$_{\texttt{defeated}}(r2,b),$
$\texttt{naf}\$_{\texttt{defeated}}(r1,a) \text{ and}$
$\texttt{naf}\$_{\texttt{defeated}}(r2,b).$
\end{verbatim}

Hence this program has only one answer set, in which both \texttt{a} and \texttt{b} are true.

The intuition that underlies such a behavior could be explained as follows. Each of the two rules, tagged with \texttt{r1} and \texttt{r2}, has some rule that overrides it, so both rules are refuted; Since refuted rules cannot be used to override any other rule, none of the rules is overridden and none is defeated.

In contrast, AT\textsuperscript{AGCLP} does not produce this answer set. Indeed, consider an interpretation in which both \texttt{a} and \texttt{b} are true. We can infer that:
\begin{verbatim}
$\texttt{refutes}(r1, a, r2, b),$
$\texttt{refutes}(r2, b, r1, a)$
\end{verbatim}

are true both, but \texttt{naf}\$_{\texttt{defeated}}(r1, a) and \texttt{naf}\$_{\texttt{defeated}}(r2, b) cannot both be true. This shows that \texttt{a} and \texttt{b} cannot both be true. So \{\texttt{a}, \texttt{b}\} is not an answer set. Instead, there are two answer sets: in one \texttt{a} is true and in the other \texttt{b} is true.

The intuition underlying this argumentation ruleset is that \texttt{r1} and \texttt{r2} refute each other, so one cannot decide which rule should defeat the other: in one possible world \texttt{r1} defeats \texttt{r2} and \texttt{a} is true, while in the other world \texttt{r2} defeats \texttt{r1} and \texttt{b} is true.

Which intuition is more suitable depends on the application domain. For instance, if higher priority means “greater confidence” and a cycle in the \#\texttt{overrides} relation is seen as a kind of contradiction then AT\textsuperscript{DL} might be preferable as a way of discarding contradictory priorities. If, however, the intent is to make use of all statements about rule priorities and resolve the contradiction by admitting all possible
/* 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).

/* 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 neg 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).
exceptions are the works [22,11,15,6,9], which had
different frameworks for such reasoning. The notable
considered disjunctive information or tried to unify the
various theories of defeasible reasoning, only a few

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 do-
main description, the predicate $\text{defeated}$. 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 Pro-
gramming, Defeasible Logic, Prioritized Defaults, and
more.

Delgrande et al. [11] propose a framework for or-
dered 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 transfor-
mation 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 seman-
tics. Instead, the definition of preferred answer sets dif-
fers from one preference-handling strategy to another.
One of the more important conceptual differences be-
tween 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 [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. 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]. 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.

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

References


[38] D. Reeves, B. Grosf, M. Wellman, and H. Chan. Towards a declarative language for negotiating executable contracts. In


