Conflict Resolution in Structured Argumentation

Martin Baláž, Jozef Frtús and Martin Homola

Faculty of Mathematics, Physics and Informatics
Comenius University in Bratislava, Slovakia
{balaz,frtus,homola}@fmph.uniba.sk

Abstract

While several interesting argumentation-based semantics for defeasible logic programs have been proposed, to our best knowledge, none of these approaches is able to fully handle the closure under strict rules in a sufficient manner: they are either not closed, or they use workarounds such as transposition of rules which violates desired directionality of logic programming rules.

We propose a novel argumentation-based semantics, in which the status of arguments is determined by attacks between newly introduced conflict resolutions instead of attacks between arguments. We show that the semantics is closed w.r.t. strict rules and respects directionality of rules, as well as other desired properties previously published in the literature.

1 Introduction

1.1 Background

Defeasible Logic Programs (DeLPs), introduced by García and Simari [3], extend the usual logic programming (LP) setting. In DeLPs, strict rules ($\rightarrow$) have to be always satisfied, so whenever we accept preconditions we also accept the consequent of these rules. On the other hand, defeasible rules ($\Rightarrow$) express tentative knowledge that is not universally valid, but instead it applies only if there is no evidence against it.

Obtaining a semantics for DeLPs is not straight forward, as it is not always immediately apparent which of the defeasible rules should be upheld and which should be defeated. Existing proposals largely rely on argumentation theory. Specifically, García and Simari [3] propose a procedural semantics based on argumentation dialogue games; while the ASPIC$^+$ framework, introduced by Prakken [5] and later refined by Modgil and Prakken [4], instantiates Dung's argumentation frameworks (AFs) [2].

1.2 Motivation

While these works are important as they show us that argumentation is a powerful tool useful in resolving conflicts in DeLPs, we will see that in some cases they may not be on par with some of the basic intuitions. For instance, strict rules are like normal LP rules, and hence they should always be satisfied. We see that this is not always the case.

Example 1. Consider the following defeasible logic program $\mathcal{P}$:

$$\begin{array}{llllll} \Rightarrow & a & \Rightarrow & c & a, b & \rightarrow & h \\ \Rightarrow & b & \Rightarrow & d & c, d & \rightarrow & \neg\, h \end{array}$$

This work appeared as short paper at the 19$^{\text{th}}$ International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-19), Stellenbosch, South Africa, 15$^{\text{th}}$–19$^{\text{th}}$ December 2013.

According to García and Simari [3] the conflict between the argument for h and the argument for $\neg h$ is resolved entirely by mutual defeat between them. That is, no pairwise conflict between the arguments for a, b, c, and d (i.e., structurally, the subarguments of the arguments for h and $\neg h$) is considered. Consequently we get $\{a, b, c, d\}$ as the meaning of the program. However, we see that the two strict rules are not satisfied – the body of each rule is true, but not their mutually inconsistent heads. Hence the semantics is incomplete in this respect.

In ASPIC$^+$ [5, 4] the same problem is resolved by extending $\mathcal{P}$ by transposition of strict rules, i.e., an operation that twists the rules around, and obtains $\neg b \to \neg a$ from $a \to b$. In Example 1 this leads into adding new rules, and consequently adding arguments for $\neg a$, $\neg b$, $\neg c$ and $\neg d$. As a result, defeats against arguments for a, b, c, d indirectly resolve the conflict between h and $\neg h$. Note that in such a case every of the strict rules is satisfied. However, the next example shows that transposition of rules is not applicable if rules are to be strictly treated in the LP manner.

Example 2. Consider the following defeasible logic program $\mathcal{P}$

$$\to \quad a \qquad \neg b \quad \to \quad \neg a \qquad b \quad \Rightarrow \quad c$$

In ASPIC$^+$ [5, 4], only one argument can be constructed: $A_1 = [\to a]$. After extending $\mathcal{P}$ with transposed strict rule $a \to b$, two new arguments can be constructed:

$$A_2 = [A_1 \to b] \quad A_3 = [A_2 \Rightarrow c]$$

Although there are no arguments for literals b or c in the original program, we get $\{a, b, c\}$ as the meaning of the program extended with transposition. The problem is that rules in logic programming are one-way directional and transposition does not respect this. We can see that the atom b does not follow from the program $\mathcal{P}$, but it does when $\mathcal{P}$ is transposed. That is, transposition significantly changes the semantics of the program and therefore it is not applicable in case of logic programming.

Such discrepancies were noticed by other researchers as well. Prakken and Sartor [6] and Caminada and Amgoud [1] formulated postulates which should be satisfied:

Compositionality: If an argument is accepted, all its subarguments must be also accepted [6].

Closure: The semantic output is closed under the set of strict rules [1].

Direct consistency: The semantic output is consistent [1].

Indirect consistency: The closure of the semantic output under the set of strict rules is consistent [1].

Apart from ASPIC$^+$, we are not aware of any proposal with an argumentation-based semantics that would satisfy all these principles. However, as we noted above, ASPIC$^+$ makes heavy use of transposition which, if rules are to be treated like in LP, yields counterintuitive results as we showed in Example 2.

1.3 Proposed Solution and Results

To satisfy the four postulates mentioned above without resorting to transposition, we propose a novel approach to DeLP based on AFs. In our approach, conflicts between arguments are

not resolved directly by defeating an argument. They are resolved by defeating either a default assumption or a defeasible rule from which one of the conflicting arguments is constructed. Since a strict rule can not be a resolution of any conflict, the closure postulate is satisfied. Thanks to entirely avoiding transposition, no new rules are introduced into programs, and directionality of rules is strictly respected. This makes our approach much more aligned with LP.

We formalize a notion of conflict resolution which captures a possible way how to resolve each conflict. We instantiate Dung's AFs with conflict resolutions instead of arguments, what allows us to define clear relationship between arguments and underlaying LP constructs (i.e. rules and default literals) on top of which they are constructed. We can thus identify which particular defeasible rule or default literal was defeated in order to resolve each conflict. We can also characterize the semantics of the program as conclusions of arguments whose defeasible rules and default literals were not defeated. None of the existing formalisms [6, 3, 5, 4] provides such a relationship.

There may be more ways how to resolve each conflict and the choice of a particular conflict resolution strategy is always a domain specific task. Therefore we allow any conflict resolution strategy to be formally encoded and used with our framework. Such an approach is general enough to compare and study different conflict resolution strategies known from the existing literature [6, 3, 5, 4], and to always select the best one for each particular application.

2 Preliminaries

We first introduce defeasible logic programs [3]. An *atom* is a propositional variable. A *classical literal* is either an atom or an atom preceded by classical negation $\neg$. A *default literal* is a classical literal preceded by default negation $\sim$. A *literal* is either a classical or a default literal. By definition $\neg\neg A$ equals to A and $\sim\sim L$ equals to L, for an atom A and a classical literal L. By $\mathcal{B}$ we will denote the set of all classical literals, by $\mathcal{D}$ we will denote the set of all default literals, and by $\mathcal{L}$ we will denote the set of all literals. A set of literals S is *consistent* if $\{A, \neg A\} \not\subseteq S$ for any atom A and $\{L, \sim L\} \not\subseteq S$ for any literal L.

A *strict rule* is an expression of the form $L_1, \ldots, L_n \rightarrow L_0$ where $0 \leq n$, each L_i, $1 \leq i \leq n$, is a literal, and L_0 is a classical literal. A *defeasible rule* is an expression of the form $L_1, \ldots, L_n \Rightarrow L_0$ where $0 \leq n$, each L_i, $1 \leq i \leq n$, is a literal, and L_0 is a classical literal. A *defeasible logic program* is a pair $\mathcal{P} = (\Pi, \Delta)$ of strict rules Π and defeasible rules Δ. In the following text we use the symbol $\rightsquigarrow$ to denote either strict or defeasible rule.

Abstract Argumentation Frameworks were introduced by Dung [2]. They were successfully applied to define semantics for DeLP and we will also make use of them in this respect.

Definition 1 (Abstract Argumentation Framework [2])**.** An *abstract argumentation framework* is a pair $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ where

1. $\mathcal{A}$ is a set of *arguments*, and
2. $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ is an *attack* relation on $\mathcal{A}$.

An argument A *attacks* an argument B if $(A, B) \in \mathcal{R}$. A set of arguments S *attacks* an argument A if an argument in S attacks A. A set of arguments S is *attack-free*[1] if S does not attack an argument in S. A set of arguments S *defends* an argument A if each argument attacking A is attacked by S. An attack-free set of arguments S is *admissible* iff S defends each argument in S.

[1] Note that we will use the original term "conflict-free" in slightly different context.

Definition 2 (Extension [2])**.** An admissible set of arguments S is

1. a *complete extension* iff S contains each argument defended by S.
2. the *grounded extension* iff S is the least complete extension.
3. a *preferred extension* iff S is a maximal complete extension.
4. a *stable extension* iff S attacks each argument which does not belong to S.

An *operator on* a set S is any mapping $F\colon 2^S \mapsto 2^S$. The closure of a set $X \subseteq S$ under the operator F (denoted $Cl_F(X)$) is the smallest set $Y \subseteq S$ such that $X \subseteq Y$ and $F(Y) \subseteq Y$.

3 Building Semantics of DeLPs

Existing argumentation formalisms [5, 3, 6] are usually defined through five steps. At the beginning, some underlying logical *language* is chosen for describing knowledge. The notion of an *argument* is then defined within this language. Then *conflicts* between arguments are identified. The resolution of conflicts is captured by an *attack* relation among conflicting arguments. The *status* of an argument is then determined by the attack relation.

In our approach, we diverge from this methodology. Instead of attacking a conflicting argument, we attack one of the building blocks used to construct the argument. Specifically, the resolution of a conflict is either a default assumption or a defeasible rule. The status of an argument does not depend on attack relation between arguments but on attack relation between conflict resolutions.

Our approach for argumentation-based semantics for the DeLP consists of five steps:

1. Construction of arguments on top of the language of defeasible logic programs.
2. Identification of conflicts between arguments.
3. Proposing a conflict resolution strategy.
4. Instantiation of Dung's AFs with conflict resolutions.
5. Determination of the status of default assumptions, defeasible rules, and arguments with respect to successful conflict resolutions.

The first three steps are described in the following subsection, the other two steps, i.e., the semantics, follow in Subsection 3.3.

3.1 Arguments, Conflicts, and Conflict Resolutions

A vulnerability is a part of an argument that may be defeated to resolve a conflict. It is either a defeasible rule or a default literal.

Definition 3 (Vulnerability)**.** Let $\mathcal{P}$ be a defeasible logic program. A *vulnerability* is a defeasible rule in $\mathcal{P}$ or a default literal in $\mathcal{D}$. By $\mathcal{V}_\mathcal{P}$ we will denote the set of all vulnerabilities of $\mathcal{P}$.

Two kinds of arguments can be constructed in the language of defeasible logic programs. Default arguments correspond to default literals. Deductive arguments are constructed by chaining of rules. Formalization of the structure of arguments was inspired by the work of Vreeswijk [8] which was later refined by Prakken [5].

For both types of arguments we define several functions Conc, SubArgs, Vuls denoting the conclusion, direct subarguments and vulnerabilities of an argument. Note that SubArgs defines only direct subarguments and the general term of subargument will be denoted with its transitive closure $\sqsubseteq$.

Definition 4 (Argument)**.** Let $\mathcal{P} = (\Pi, \Delta)$ be a defeasible logic program. An argument A for a literal L is

1. a *default argument* L where L is a default literal in $\mathcal{D}$

$$\begin{aligned} \textsc{Conc}(A) &= L & \textsc{AllDef}(A) &= \emptyset \\ \textsc{SubArgs}(A) &= \emptyset & \textsc{LastDef}(A) &= \emptyset \\ \textsc{Vuls}(A) &= \{L\} \end{aligned}$$

2. a *deductive argument* $[A_1, \ldots, A_n \rightsquigarrow L]$ where each A_i, $1 \leq i \leq n$, is an argument for a literal L_i, $r\colon L_1, \ldots, L_n \rightsquigarrow L$ is a rule in $\mathcal{P}$.

$$\begin{aligned} \textsc{Conc}(A) &= L \\ \textsc{SubArgs}(A) &= \{A_1, \ldots, A_n\} \\ \textsc{Vuls}(A) &= \textsc{Vuls}(A_1) \cup \cdots \cup \textsc{Vuls}(A_n) \cup (\{r\} \cap \Delta) \\ \textsc{AllDef}(A) &= \textsc{AllDef}(A_1) \cup \cdots \cup \textsc{AllDef}(A_n) \cup (\{r\} \cap \Delta) \\ \textsc{LastDef}(A) &= \textsc{LastDef}(A_1) \cup \cdots \cup \textsc{LastDef}(A_n) \text{ if } r \in \Pi;\ \{r\} \text{ otherwise} \end{aligned}$$

By $\mathcal{A}_\mathcal{P}$ we will denote the set of all arguments of $\mathcal{P}$. For an argument A, we define

$$\textsc{SupArgs}(A) = \{B \in \mathcal{A}_\mathcal{P} \mid A \in \textsc{SubArgs}(B)\}$$

For a set of arguments $S \subseteq \mathcal{A}_\mathcal{P}$ and a function $F\colon \mathcal{A}_\mathcal{P} \mapsto 2^{\mathcal{A}_\mathcal{P}}$, we define

$$F(S) = \bigcup_{A \in S} F(A)$$

An argument $A \in \mathcal{A}_\mathcal{P}$ is a *subargument* resp. a *superargument* of an argument $B \in \mathcal{A}_\mathcal{P}$ (denoted $A \sqsubseteq B$ resp. $A \sqsupseteq B$) iff $A \in Cl_{\textsc{SubArgs}}(\{B\})$ resp. $A \in Cl_{\textsc{SupArgs}}(\{B\})$.

The immediate consequence operator [7] is used in logic programming in order to characterize a single step of computations for a given logic program. We extend this operator so that it works on arguments (originally it works on sets of literals) which will be later useful to define argument closures.

Definition 5 (Immediate Consequence Operator)**.** Let $\mathcal{P}$ be a defeasible logic program and $S \subseteq \mathcal{A}_\mathcal{P}$ be a set of arguments. By $F_\mathcal{P}(S)$ we denote the set of all deductive arguments $[A_1, \ldots, A_n \rightsquigarrow L]$ such that each A_i, $1 \leq i \leq n$, is an argument in S for a literal L_i and $L_1, \ldots, L_n \rightsquigarrow L$ is a rule in $\mathcal{P}$.

To simplify the notation, we will denote the closure $Cl_{F_\mathcal{P}}$ also as $Cl_\mathcal{P}$ hereafter. The set of all arguments can be created in a bottom-up fashion by iterating the immediate consequence operator on the set of default literals.

Two kinds of conflicts among arguments may arise, each corresponds to one type of negation. Rebutting is a conflict between conclusion of two deductive arguments, while undercutting is a conflict between a default assumption and a conclusion of a deductive argument.

Definition 6 (Conflict)**.** Let $\mathcal{P}$ be a defeasible logic program. Arguments $A, B \in \mathcal{A}_\mathcal{P}$ are *conflicting* iff A rebuts or undercuts B where

1. A *rebuts* B iff A and B are deductive arguments and $\textsc{Conc}(A) = \neg\,\textsc{Conc}(B)$,
2. A *undercuts* B iff A is a deductive argument, B is a default argument, and $\textsc{Conc}(A) = \sim\textsc{Conc}(B)$.

The set $C = \{A, B\}$ is called a *conflict*. The first kind is called a *rebutting* conflict and the second kind is called an *undercutting* conflict. By $\mathcal{C}_\mathcal{P}$ we will denote the set of all conflicts of $\mathcal{P}$.

Conflicts are resolved by defeating one of the building blocks of conflicting arguments. Each default assumption or defeasible rule used to construct a conflicting argument is a possible resolution. Strict rules can not be used as a resolution of any conflict because they have to be always satisfied.

Definition 7 (Conflict Resolution)**.** Let $\mathcal{P}$ be a defeasible logic program. A vulnerability $V \in \mathcal{V}_\mathcal{P}$ is a *resolution of* a conflict $C \in \mathcal{C}_\mathcal{P}$ if $V \in \textsc{Vuls}(C)$. The pair $R = (C, V)$ is called a *conflict resolution*. By $\mathcal{R}_\mathcal{P}$ we will denote the set of all conflict resolutions of $\mathcal{P}$.

In general, each conflict may have more resolutions. Some of them may be more preferred than others. The choice of preferred conflict resolutions is domain dependent. Some vulnerabilities can be defeated in one domain, but they may as well stay undefeated in another. Therefore we allow the user to choose any conflict resolution strategy she might prefer.

Formally, a conflict resolution strategy is a set of preferred conflict resolutions. Total conflict resolution strategies resolve all conflicts.

Definition 8 (Conflict Resolution Strategy)**.** Let $\mathcal{P}$ be a defeasible logic program. A *conflict resolution strategy* is a subset σ of $\mathcal{R}_\mathcal{P}$. We say that a vulnerability $V \in \mathcal{V}_\mathcal{P}$ is a σ-resolution of a conflict $C \in \mathcal{C}_\mathcal{P}$ if $(C, V) \in \sigma$. A conflict resolution strategy σ is *total* iff for each conflict $C \in \mathcal{C}_\mathcal{P}$ there exists a σ-resolution of C.

3.2 Examples of Conflict Resolution Strategies

Although we do not restrict which vulnerabilities may resolve specific kind of conflicts, usually undercutting conflicts are resolved by defeating the default assumption and rebutting conflicts are resolved by defeating a defeasible rule. In this subsection we show three examples of conflict resolution strategies inspired by the existing work.

Default reasoning strategy resolves all – and only – undercutting conflicts by defeating the default assumption. This strategy captures the main idea of default reasoning – we can assume that a classical literal L is false unless we have an evidence for L. So if there exists a deductive argument for L, we have to withdraw the default argument $[\sim L]$.

Definition 9 (Default Reasoning Strategy)**.** The *default reasoning strategy* is a conflict resolution strategy σ_{def} where $(C, V) \in \sigma_{def}$ iff

- C is an undercutting conflict
- V is a default literal in C

Example 3. Consider the defeasible logic program $\{\sim a \rightarrow b\}$. Following arguments can be constructed:

$$\begin{array}{llllllll} A_1 & = & [\sim a] & A_3 & = & [\sim b] & A_5 & = & [A_1 \Rightarrow b] \\ A_2 & = & [\sim \neg a] & A_4 & = & [\sim \neg b] & & & \end{array}$$

Then $\sigma_{def} = \{(\{A_3, A_5\}, \sim b)\}$ is the default reasoning strategy.

In the existing approaches [5, 3], the resolution of rebutting conflict is usually determined by a partial order preference relation on defeasible rules $\prec$. The last link strategy is based on defeating a minimally preferred last used defeasible rule. This strategy is inspired by the last link principle in ASPIC$^+$ [5], where, however, arguments instead of defeasible rules are defeated.

Definition 10 (Last Link Strategy)**.** Given a defeasible logic program $\mathcal{P}$ and a partial order $\prec$ on defeasible rules, the *last link strategy* is a conflict resolution strategy σ_{last} where $(C, V) \in \sigma_{last}$ iff

- C is a rebutting conflict
- V is a $\prec$-minimal defeasible rule in $\textsc{LastDef}(C)$

The following example demonstates the last link strategy. Note that from now on, we will omit default arguments from examples to improve the legibility.

Example 4. Given the defeasible logic program

$$\begin{array}{lllll} r_1\colon & & \Rightarrow & b \\ r_2\colon & b & \Rightarrow & a \\ r_3\colon & & \Rightarrow & \neg a \end{array}$$

and order $r_1 \prec r_3 \prec r_2$, deductive arguments are

$$A_1 = [\Rightarrow b] \qquad A_2 = [A_1 \Rightarrow a] \qquad A_3 = [\Rightarrow \neg a]$$

Then $\sigma_{last} = \{(\{A_2, A_3\}, r_3)\}$ is the last link strategy.

On the other hand, the weakest link strategy resolves all and only rebutting conflicts by defeating a minimally preferred defeasible rule.

Definition 11 (Weakest Link Strategy)**.** Given a defeasible logic program $\mathcal{P}$ and a partial order $\prec$ on defeasible rules, the *weakest link strategy* is a conflict resolution strategy $\sigma_{weakest}$ where $(C, V) \in \sigma_{weakest}$ iff

- C is a rebutting conflict
- V is a $\prec$-minimal defeasible rule in $\textsc{AllDef}(C)$

Note that although the weakest link strategy is inspired by the weakest link principle in ASPIC$^+$ [5], there is one notable difference. In ASPIC$^+$, the rebutting conflict is, according to weakest link principle, always resolved by defeating an argument containing a minimally preferred defeasible rule r. That is, not directly an argument with toprule r is defeated. On the other hand, in our approach, conflicts may be resolved not only by defeating a last-used defeasible rule, but also by one occuring in arbitrary subargument. So, in this respect, our formalism is more flexible and general than existing works [6, 3, 5, 4].

Example 5. Consider the defeasible logic program and the order on rules from Example 4. The weakest link strategy is $\sigma_{weakest} = \{(\{A_2, A_3\}, r_1)\}$.

3.3 Semantics

In this subsection we derive the semantics of a defeasible logic program by an instantiation of Dung's argumentation frameworks, i.e., we implement the last two steps of our approach as described at the beginning of Section 3. Dung's AF is however instantiated with conflict resolutions rather than with arguments and the intuitive meaning of a conflict resolution (C, V) is "the conflict C will be resolved by defeating the vulnerability V". The semantics is built on three levels of attacks: attacks on the vulnerabilities, attacks on the arguments, and attacks on the conflict resolutions. Such an approach is necessary: if a vulnerability is defeated, so should be all arguments built on it, and consequently all conflict resolutions respective to the argument. In the definition below, we specify these three levels of attacks, step by step.

Definition 12 (Attack)**.** A conflict resolution $R = (C, V)$ *attacks*

- a vulnerability V' iff $V' = V$.
- an argument A iff R attacks a vulnerability in $\textsc{Vuls}(A)$.
- a conflict resolution $R' = (C', V')$ iff either
  1. $V \neq V'$ and R attacks an argument in C' or
  2. $V = V'$ and R attacks all arguments in C'.

A set of conflict resolutions $S \subseteq \mathcal{R}_\mathcal{P}$ attacks a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \mathcal{R}_\mathcal{P}$) iff a conflict resolution in S attacks V (resp. A or R).

Intuitively, it should not happen that both a conflict resolution $R = (C, V)$ and a vulnerability V are accepted. Therefore, if R is accepted, V and all arguments constructed on top of it should be defeated. The notion of attack between conflict resolutions formalizes the ideas that there may be more alternatives how to resolve a conflict and a conflict resolution may resolve other conflicts as well, thus causing other conflict resolutions to be irrelevant. This is addressed by the first option of the third case in Definition 12: a conflict C' is already resolved by the conflict resolution $R = (C, V)$ because $V \in \textsc{Vuls}(C')$ (see Examples 6–7). The second option takes care of self-attacking conflict resolutions, as we can see in the Example 8.

Example 6. Consider the conflict $C = \{[[\ \Rightarrow a] \rightarrow c], [[\ \Rightarrow b] \rightarrow \neg c]\}$ and two different resolutions $R_1 = (C, \Rightarrow a)$, $R_2 = (C, \Rightarrow b)$. Since to resolve the conflict C it is sufficient to defeat only one defeasible rule, R_1 and R_2 are alternatives and hence attack each other.

Example 7. Let $C_1 = \{[\rightarrow a], [\sim a]\}$, $C_2 = \{[\Rightarrow \neg b], [[\sim a] \rightarrow b]\}$ be conflicts with resolutions $R_1 = (C_1, \sim a)$, $R_2 = (C_2, \Rightarrow \neg b)$. Observe that to resolve both conflicts C_1, C_2 it is sufficient to defeat only vulnerability $\sim a$. Therefore, if R_1 is accepted, R_2 is not (i.e. it is not the case that conflict C_2 will be resolved by defeating vulnerability V_2) and consequently R_1 attacks R_2.

Example 8. Let $C_1 = \{[\sim a], [[\sim a] \rightarrow a]\}$ be a conflict with a resolution $V_1 = \sim a$. Following the intuitions in logic programming, both arguments in C_1 are intended to be undecided. Thus (C_1, V_1) should be self-attacking. On the other side, we do not want let $C_2 = \{[\sim a], [\ \rightarrow a]\}$ attack itself.

We now define how a Dung's argumentation framework is constructed on top of the conflict resolutions and the attacks between them.

Definition 13 (Instantiation)**.** The instantiation for a conflict resolution strategy σ is an abstract argumentation framework $\mathcal{F} = (\mathcal{A}, \mathcal{R})$ where

- $\mathcal{A} = \sigma$
- $\mathcal{R}$ is the attack relation on σ from the Definition 12.

Now thanks to the instantiation we can use the Dung's semantics in order to compute which vulnerabilities (resp. arguments, conflict resolutions) are undefeated (status IN), defeated (status OUT), or undecided (status UNDEC).

Definition 14 (Defense)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$. A set of conflict resolutions $S \subseteq \sigma$ *defends* a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \sigma$) iff each conflict resolution in σ attacking V (resp. A or R) is attacked by S.

Definition 15 (Status)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. The *status of* a vulnerability $V \in \mathcal{V}_\mathcal{P}$ (resp. an argument $A \in \mathcal{A}_\mathcal{P}$ or a conflict resolution $R \in \sigma$) with respect to $\mathcal{E}$ is

- IN if $\mathcal{E}$ defends V (resp. A or R),
- OUT if V (resp. A or R) is attacked by $\mathcal{E}$,
- UNDEC otherwise.

Let $S \subseteq \{\text{IN}, \text{UNDEC}, \text{OUT}\}$. By $\mathcal{V}^S_\mathcal{P}(\mathcal{E})$ (resp. $\mathcal{A}^S_\mathcal{P}(\mathcal{E})$ or $\mathcal{R}^S_\mathcal{P}(\mathcal{E})$) we denote the set of all vulnerabilities (resp. arguments or conflict resolutions) with the status in S. If S contains only one element s, we will write only $\mathcal{V}^s_\mathcal{P}(\mathcal{E})$ (resp. $\mathcal{A}^s_\mathcal{P}(\mathcal{E})$ or $\mathcal{R}^s_\mathcal{P}(\mathcal{E})$).

Given the status of conflict resolutions, computed from the Dung's semantics, we can now easily derive also the status of vulnerabilities and arguments. This is important in order to tell which rules and defaults are actually defeated and which are undefeated.

Proposition 1. *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P} = (\Pi, \Delta)$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. Let $\mathcal{P}^S(\mathcal{E}) = \Pi \cup (\mathcal{V}^S_\mathcal{P}(\mathcal{E}) \cap \Delta)$ and $\mathcal{D}^S(\mathcal{E}) = \mathcal{V}^S_\mathcal{P}(\mathcal{E}) \cap \mathcal{D}$. Then for any $S \in \{\{\text{IN}\}, \{\text{IN}, \text{UNDEC}\}, \{\text{IN}, \text{UNDEC}, \text{OUT}\}\}$ we have $\mathcal{A}^S_\mathcal{P}(\mathcal{E}) = Cl_{\mathcal{P}^S(\mathcal{E})}(\mathcal{D}^S(\mathcal{E}))$.*

Proof. We will demonstrate the proof only for the labels $S = \{\text{IN}\}$, since for the other cases it will be similar. Two inclusions have to considered:

($\subseteq$) Assume some argument $A \in \mathcal{A}^S_\mathcal{P}(\mathcal{E})$. Then $\mathcal{E}$ defends A and consequently $\text{VULS}(A) \subseteq \mathcal{P}^S(\mathcal{E}) \cup \mathcal{D}^S(\mathcal{E})$. $A \in Cl_{\mathcal{P}^S(\mathcal{E})}(\mathcal{D}^S(\mathcal{E}))$ can be shown by induction on A's structure.

($\supseteq$) Assume some argument $A \in Cl_{\mathcal{P}^S(\mathcal{E})}(\mathcal{D}^S(\mathcal{E}))$. Then from the construction of A we have $\text{VULS}(A) \subseteq \mathcal{P}^S(\mathcal{E}) \cup \mathcal{D}^S(\mathcal{E})$. Therefore $\mathcal{E}$ defends each vulnerability in $\text{VULS}(A)$ and consequently $\mathcal{E}$ defends also A. So the status of A w.r.t. $\mathcal{E}$ is IN.

□

We now derive the actual semantics of the input DeLP program by taking the literals supported by the arguments that are undefeated.

Definition 16 (Output)**.** Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. The *output* of $\mathcal{E}$ is a set of literals $\text{OUTPUT}_\mathcal{P}(\mathcal{E}) = \{L \in \mathcal{L} \mid \mathcal{A}^{\text{IN}}_\mathcal{P}(\mathcal{E}) \text{ contains an argument for } L\}$.

3.4 Examples Revisited

We will now revisit the examples from the introduction to show how the semantics is able to cope with the problems that were described there. Note that we will omit default arguments and also default literals in output to improve the legibility.

Example 9 (Revisiting Example 1). Six deductive arguments can be constructed from the original program

$$\begin{array}{lllllllll} A_1 & = & [\Rightarrow a] & & A_3 & = & [\Rightarrow c] & & A_5 & = & [A_1, A_2 \to h] \\ A_2 & = & [\Rightarrow b] & & A_4 & = & [\Rightarrow d] & & A_6 & = & [A_3, A_4 \to \neg h] \end{array}$$

The conflict $C = \{A_5, A_6\}$ is the only one. Suppose conflict C can be resolved by defeating any of the defeasible rules. That is, conflict resolution strategy σ contains four conflict resolutions:

$$R_1 = (C, \Rightarrow a) \qquad R_3 = (C, \Rightarrow c) \qquad R_2 = (C, \Rightarrow b) \qquad R_4 = (C, \Rightarrow d)$$

All conflict resolutions are now exclusive, since to resolve the conflict C, it is sufficient to reject only one of the defeasible rules. The instantiation for σ is on the Figure 1.

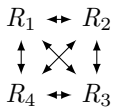

Figure 1: The instantiation for the conflict resolution strategy σ.

There are five complete extensions $\{R_1\}$, $\{R_2\}$, $\{R_3\}$, $\{R_4\}$, $\{\}$ of the instantiation and each of them determine one program output $\{b, c, d, \neg h\}$, $\{a, c, d, \neg h\}$, $\{a, b, d, h\}$, $\{a, b, c, h\}$, $\{\}$. We can see that each of these outputs satisfies both strict rules.

Example 10 (Revisiting Example 2). Recall that only one deductive argument $A_1 = [\to a]$ can be constructed from the original program. Therefore there is no conflict at all. Consequently the conflict resolution strategy σ and its only complete extension is the empty set. The output of the program is $\{a\}$ as intended.

4 Properties

Our main goal with this work was to obtain a semantics for DeLP which would satisfy the desired properties and intuitions behind DeLP. First of such properties is direct consistency [1], requiring that each output of the program is consistent. In our setting we refer to this property as conflict-freeness, as we formulate it for conflicts rather then for outputs. Given an arbitrary conflict resolution strategy σ (i.e., not necessarily a total one), if a conflict has a resolution in σ then this conflict will be resolved.

Definition 17 (Conflict-Freeness)**.** Let $\mathcal{P}$ be a defeasible logic program. A set of arguments $S \subseteq \mathcal{A}_\mathcal{P}$ is *conflict-free* iff there does not exist a conflict $C \in \mathcal{C}_\mathcal{P}$ such that $C \subseteq S$.

Proposition 2 (Conflict-Freeness Principle)**.** *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. Let $C \in \mathcal{C}_\mathcal{P}$ be a conflict s.t. $(C, V) \in \sigma$ for some vulnerability V. Then $C \not\subseteq \mathcal{A}^{\text{IN}}_\mathcal{P}(\mathcal{E})$.*

Proof. Let $R = (C, V) \in \sigma$ and $C = \{A, B\}$. Without loss of generality suppose $V \in \textsc{Vuls}(A)$. There are three cases to consider:

- $R \in \mathcal{E}$. Then status of A w.r.t. $\mathcal{E}$ is $\textsc{Out}$.
- $R \notin \mathcal{E}$ and $\mathcal{E}$ attacks R. So there is a conflict resolution $R_2 = (C_2, V_2) \in \mathcal{E}$ attacking R. If $\mathcal{E}$ attacks A, the status of A w.r.t. $\mathcal{E}$ is $\textsc{Out}$. Therefore suppose that $\mathcal{E}$ does not attack A. From previous we have $V_2 \notin \textsc{Vuls}(A)$, $V_2 \neq V$ and consequently also R_2 attacks $B \in C$. So the status of B w.r.t. $\mathcal{E}$ is $\textsc{Out}$.
- $R \notin \mathcal{E}$ and $\mathcal{E}$ does not attack R. Then status of A w.r.t $\mathcal{E}$ cannot be $\textsc{In}$.

In all cases we have $C \not\subseteq \mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})$. □

As a straight forward consequence, if σ is total, all conflicts will be resolved.

Corollary 1. *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. If σ is total then $\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})$ is conflict-free.*

We also get the original property as a direct consequence.

Corollary 2 (Direct Consistency)**.** *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. If σ is total then $\textsc{Output}_{\mathcal{P}}(\mathcal{E})$ is consistent.*

Another desired property is the closure under strict rules [1]. We generalize this property in our setting in that respect that the closure is not required only under strict rules but in addition also under all defeasible rules and default assumptions (i.e., vulnerabilities) which do not appear in any conflict in the given conflict resolution strategy.

Proposition 3 (Closure Principle)**.** *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. Let $\mathcal{P}' \subseteq \mathcal{P}$ be the set of all rules and $\mathcal{D}' \subseteq \mathcal{D}$ be the set of all default literals which are not σ-resolutions of any conflict. Then $Cl_{\mathcal{P}'}(\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E}) \cup \mathcal{D}') \subseteq \mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})$.*

Proof. Assume $A \in Cl_{\mathcal{P}'}(\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E}) \cup \mathcal{D}')$. From definition of A each attack against it is also attack against some argument in $\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})$. Therefore $\mathcal{E}$ defends A and the status of A w.r.t. $\mathcal{E}$ is always $\textsc{In}$. □

The original property is a consequent from the Proposition 3 and Corollary 2.

Corollary 3 (Indirect Consistency)**.** *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P} = (\Pi, \Delta)$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. If σ is total then $\textsc{Conc}(Cl_{\Pi}(\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})))$ is consistent.*

The compositionality property [6] states that an argument should not be accepted (status $\textsc{In}$) unless all its subarguments are accepted. We generalize this property in that respect that for each argument that is defeated (status $\textsc{Out}$) we require that all its superarguments are defeated as well. Note that this is not a direct consequence because arguments can also be in the $\textsc{Undec}$ state.

Proposition 4 (Compositionality Principle)**.** *Let σ be a conflict resolution strategy for a defeasible logic program $\mathcal{P}$ and $\mathcal{E}$ be a complete extension of the instantiation for σ. Then $Cl_{\textsc{SubArgs}}(\mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})) \subseteq \mathcal{A}^{\textsc{In}}_{\mathcal{P}}(\mathcal{E})$ and $Cl_{\textsc{SupArgs}}(\mathcal{A}^{\textsc{Out}}_{\mathcal{P}}(\mathcal{E})) \subseteq \mathcal{A}^{\textsc{Out}}_{\mathcal{P}}(\mathcal{E})$.*

Proof. Let $A \in \mathcal{A}_{\mathcal{P}}^{\text{IN}}(\mathcal{E})$ and $B \sqsubseteq A$. Each conflict resolution attacking B attacks also A. Since $\mathcal{E}$ defends A, $\mathcal{E}$ defends also B.

Let $A \in \mathcal{A}_{\mathcal{P}}^{\text{OUT}}(\mathcal{E})$ and $B \sqsupseteq A$. Since each superargument of A is attacked by $\mathcal{E}$, $B \in \mathcal{A}_{\mathcal{P}}^{\text{OUT}}(\mathcal{E})$. □

5 Conclusions

We have proposed a novel semantics for DeLPs based on AFs. In this semantics, conflicts between arguments are not resolved by defeating arguments but instead by defeating one of the defeasible rules or default literals from which the arguments are constructed. Such conflict resolutions are formally treated and used as the base for instantiation of Dung's AFs, which is one of the key features of our approach. The main contribution is as follows:

- Formalization of conflict resolutions and their use in the instantiation yields a more flexible and more general approach. Notably, we can resolve conflicts not only by defeating one of the last-used defeasible rules.
- The semantics tracks not only the status of arguments, but also the status of vulnerabilities. We can thus always say which default assumptions or defeasible rule was defeated in order to resolve each conflict.
- Since transposition of rules is avoided, directionality of inference rules is strictly respected which is in line with LP intuitions.
- The semantics satisfies desired properties specified by previous research, specifically it deals with strict rules in a satisfactory fashion.

Acknowledgements

The authors would like to thank to Stefan Woltran, Johannes Wallner, and Sylwia Polberg from TU Vienna, to Martin Caminada and Srdjan Vesic from the University of Luxembourg, and as well as to Giorgos Flouris, and Theodore Patkos from the FORTH foundation in Heraklion for fruitful discussions concerning this research. This work is supported from the VEGA project 1/1333/12, the SRDA mobility project SK-GR-0070-11, and the SAIA project 2012-03-15-0001. Martin Baláž and Martin Homola are also partially supported from the SRDA project APVV-0513-10.

References

[1] Martin Caminada and Leila Amgoud. On the evaluation of argumentation formalisms. *Artificial Intelligence*, 171(5-6):286–310, April 2007.

[2] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–357, September 1995.

[3] Alejandro Javier García and Guillermo Ricardo Simari. Defeasible logic programming: an argumentative approach. *Theory and Practice of Logic Programming*, 4(2):95–138, January 2004.

[4] Sanjay Modgil and Henry Prakken. Revisiting Preferences and Argumentation. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, pages 1021–1026. AAAI Press, 2011.

[5] Henry Prakken. An abstract framework for argumentation with structured arguments. *Argument & Computation*, 1(2):93–124, June 2010.

[6] Henry Prakken and Giovanni Sartor. Argument-based extended logic programming with defeasible priorities. *Journal of Applied Nonclassical Logics*, 7(1):25–75, 1997.

[7] Maarten H. van Emden and Robert Anthony Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, October 1976.

[8] Gerard A. W. Vreeswijk. Abstract argumentation systems. *Artificial Intelligence*, 90(1-2):225–279, February 1997.