Embedding Defeasible Logic Programs into Generalized Logic Programs

Martin Baláž[1], Jozef Frtús[1], Martin Homola[1], Ján Šefránek[1], and Giorgos Flouris[2]

[1] Comenius Unversity in Bratislava, Slovakia
[2] FORTH-ICS, Greece

Abstract. A novel argumentation semantics of defeasible logic programs (DeLP) is presented. Our goal is to build a semantics, which respects existing semantics and intuitions of "classical" logic programming. Generalized logic programs (GLP) are selected as an appropriate formalism for studying both undermining and rebutting. Our argumentation semantics is based on a notion of conflict resolution strategy (CRS), in order to achieve an extended flexibility and generality. Our argumentation semantics is defined in the frame of assumption-based framework (ABF), which enables a unified view on different non-monotonic formalisms. We present an embedding of DeLP into an instance of ABF. Consequently, argumentation semantics defined for ABF are applicable to DeLP. Finally, DeLP with CRS is embedded into GLP. This transformation enables to commute argumentation semantics of a DeLP via semantics of the corresponding GLP.

1 Introduction

Defeasible Logic Programs (DeLPs) [8] combine ideas from Defeasible Logic [13,12] and Logic Programming. While classically, logic programs (LPs) feature default negation, which enables to express default assumptions (i.e., propositions which are supposed to hold unless we have some hard evidence against them), DeLPs additionally introduce defeasible rules (i.e., rules which are supposedly applicable unless we have some hard evidence opposing them). Strict rules (i.e., regular LP rules) are denoted by $\rightarrow$ and defeasible rules by $\Rightarrow$. Let us illustrate this with an example.

Example 1. Brazil is the home team, and has a key player injured. Home teams tend to work the hardest, and who works the hardest usually wins. The team who has a key player injured does not usually win. The program is formalized into the DeLP:

$$\rightarrow home \qquad r_1 : home \Rightarrow works\_hard \qquad r_2 : works\_hard \Rightarrow wins$$
$$\rightarrow key\_player\_injured \qquad r_3 : key\_player\_injured \Rightarrow \text{not } wins$$

In the program from Example 1 there are two strict and three defeasible rules. The strict rules are facts, hence *home* and *key_player_injured* should always be true. Based on the first fact we are able to derive that Brazil should win, using the two defeasible rules r_1 and r_2, while based on the second fact we are able to derive that Brazil should not win, again relying on a defeasible rule, in this case r_3. Hence there is a conflict which somehow should be resolved.

Various approaches to DeLP typically rely on argumentation theory in order to determine which rules should be upheld and which should be defeated. However, as it can be perceived from Example 1, it is not always immediately apparent how this should be decided.

According to García and Simari [8], both rules immediately causing the conflict (r_2 and r_3) would be undecided, accepting *home*, *key_player_injured* and *works_hard* as valid derivations while taking both *wins* and not *wins* as undecided. ASPIC$^+$ [14,11], on the other hand, allows two additional solutions, one with r_2 undefeated, r_3 defeated, and *wins* valid; and the other one with r_2 defeated, r_3 undefeated, and not *wins* valid.

Some approaches, like ASPIC$^+$, allow to specify a preference relation on rules. In such a case conflict resolution may take this into account. Specifically, ASPIC$^+$ has two built-in conflict resolution strategies, *weakest-link principle* by which the rule with smallest preference is defeated among those involved in each conflict, and *last-link principle* by which only the rules immediately causing the conflict are considered and the least preferred is defeated.

We observe that more ways to resolve conflicts may be needed. This is due to the fact that defeasible rules are *domain specific*, a different conflict resolution strategy may be needed for a different domain, or in distinct application. We therefore argue that the conflict resolution strategy should be a user-specified parameter of the framework, and any DeLP framework should allow a generic way how to specify it (alongside some predefined strategies).

Some of the semantics proposed for DeLP satisfy the well accepted rationality properties for defeasible reasoning, such as *consistency* (extensions should be conflict-free) and closure (extensions should be closed w.r.t. the strict rules), as defined by Caminada and Amgoud [4]. While these properties are important, DeLP is an extension of LP, and some attention should be also devoted to keeping it in line with it. Specifically, we would like to have the semantics backward-compatible with the underlying language of logic programs – if no defeasible rules are present, the extensions should be in line with the respective class of models.

In our previous work [2] we have formalized the notion of conflict resolution strategy (CRS) and we have proposed a DeLP framework which allows to use any such strategy. The relationship with the underlying class of LPs was not investigated though. In the current paper we extend this work as follows:

– We rebuild the argumentation semantics (including the notion of conflict resolution strategy) using the Assumption Based Framework (ABF), an argumentation formalism very close in spirit to logic programming.

– We show that the semantics satisfies the closure and consistency properties [4], and we also show two additional properties which govern the handling of defeasible rules.
– We provide an alternative transformational semantics, which translates the DeLP and the given CRS into a regular logic program. We show that both semantics are equivalent. Thanks to the transformational semantics we also show full backward compatibility with the underlying class of generalized logic programs. What is more, the semantics of DeLP can now be computed using existing LP solvers.

All proofs can be found in a technical report which appears at `http://kedrigern.dcs.fmph.uniba.sk/reports/download.php?id=58`.

2 Preliminaries

Generalized logic programs and assumption-based frameworks provide a background for our investigation. We are aiming at a computation of our argumentation semantics of DeLP in the frame of classical logic programs. Generalized logic programs (with default negations in the heads of rules) are selected as a simplest LP-formalism, which enables to consider both undermining and rebutting.

Assumption-based frameworks are used in our paper as a basis for building a semantics of DeLP. ABF is a general and powerful formalism providing a unified view on different non-monotonic formalisms using argumentation semantics.

2.1 Generalized Logic Programs

We will consider propositional generalized logic programs (GLPs) in this paper.

Let At be a set of *atoms* and not $At = \{\text{not } A \mid A \in At\}$ be a set of *default literals*. A *literal* is an atom or a default literal. The set of all literals is denoted by $\mathcal{L}_{At}$. If $L = \text{not } A$ and $A \in At$, then by not L we denote A. If $S \subseteq \mathcal{L}_{At}$, then not $S = \{\text{not } A \mid A \in S\}$.

A *rule* over $\mathcal{L}_{At}$ is an expression r of the form $L_1, \ldots, L_n \to L_0$ where $0 \leq n$ and $L_i \in \mathcal{L}_{At}$ for each $0 \leq i \leq n$. The literal $head(r) = L_0$ is called the *head* of r and the set of literals $body(r) = \{L_1, \ldots, L_n\}$ is called the *body* of r.

A *generalized logic program* is a finite set of rules. We will often use only the term program. If At is the set of all atoms used in a program $\mathcal{P}$, it is said that $\mathcal{P}$ is over At. If heads of all rules of a program $\mathcal{P}$ are atoms, it is said that $\mathcal{P}$ is *normal*. A program $\mathcal{P}$ is called *positive* if the head of every rule is an atom and the body of every rule is a set of atoms or propositional constants $\mathbf{t}, \mathbf{u}, \mathbf{f}$.

Note that a GLP $\mathcal{P}$ can be viewed as consisting of two parts, a normal logic program $\mathcal{P}^+ = \{r \in \mathcal{P} \mid head(r) \in At\}$ (also called the positive part of $\mathcal{P}$) and a set of "constraints" $\mathcal{P}^- = \mathcal{P} \setminus \mathcal{P}^+$ (also called the negative part of $\mathcal{P}$).

Our definitions of some basic semantic notions follow the ideas of Przymusinski [17] (see also [6]); however, an adaptation to the case of rules with default negations in head is needed. In our approach we will use the positive part P^+ of

the program as a generator of a broad set of candidate models and consecutively we will use the negative part P^- to filter out some of the models.

Definition 1 (Partial and Total Interpretation). *A set of literals S is consistent, if it does not contain a pair $A, \text{not } A$ where $A \in At$. A partial* interpretation *is a consistent set of literals. A total interpretation is a partial interpretation I such that for every $A \in At$ either $A \in I$ or* $\text{not } A \in I$.

Each interpretation can be viewed as a mapping $I\colon At \mapsto \{0, \frac{1}{2}, 1\}$ where $I(A) = 0$ if $\text{not } A \in I$, $I(A) = \frac{1}{2}$ if $A \notin I$ and $\text{not } A \notin I$, and $I(A) = 1$ if $A \in I$. A *valuation* given by an interpretation I is a mapping $\hat{I}\colon \mathcal{L}_{At} \mapsto \{0, \frac{1}{2}, 1\}$ where $\hat{I}(A) = I(A)$ and $\hat{I}(\text{not } A) = 1 - I(A)$ for each atom $A \in At$, and $\hat{I}(\mathbf{t}) = 1$, $\hat{I}(\mathbf{u}) = \frac{1}{2}$, $\hat{I}(\mathbf{f}) = 0$. An interpretation I satisfies a rule r (denoted $I \models r$) iff $\hat{I}(head(r)) \geq \hat{I}(body(r)) = \min\{\hat{I}(L) \mid L \in body(r)\}$.

Definition 2 (Model). *An interpretation I is a model of a generalized logic program $\mathcal{P}$ iff I satisfies each rule in $\mathcal{P}$.*

As usual in logic programming, "classical" model, as defined above, are too broad and a number of more fine-grained semantics, based on certain notion of minimality are used. We proceed by defining these semantics summarily for GLPs. Not all of them were thoroughly investigated in literature, however we use analogy with other classes of logic programs, especially normal logic programs.

The notions of truth ordering and knowledge ordering on partial interpretations will be needed. For a partial interpretation I, let $T(I) = \{A \in At \mid I(A) = 1\}$ and $F(I) = \{A \in At \mid I(A) = 0\}$.

Definition 3 (Truth and Knowledge Ordering). *If I, J are partial interpretations, then*

- $I \leq_t J$ *iff* $T(I) \subseteq T(J)$ *and* $F(I) \supseteq F(J)$,
- $I \leq_k J$ *iff* $T(I) \subseteq T(J)$ *and* $F(I) \subseteq F(J)$.

Definition 4 (Program Reduct). *Let I be an interpretation. The* reduct *of a normal logic program $\mathcal{P}$ is a positive logic program $\mathcal{P}^I$ obtained from $\mathcal{P}$ by replacing in every rule of $\mathcal{P}$ all default literals which are true (resp. unknown, resp. false) in I by propositional constant* $\mathbf{t}$ *(resp.* $\mathbf{u}$*, resp.* $\mathbf{f}$*).*

Finally a fixed-point condition is expressed on a reduced program, which is formally captured by the operator $\Gamma_{\mathcal{P}}$.

Definition 5 (Operator $\Gamma_{\mathcal{P}}$). *Let $\mathcal{P}$ be a normal logic program and I be an interpretation. By $\Gamma_{\mathcal{P}}(I)$ we denote the t-least model of $\mathcal{P}^I$.*

Definition 6 (Semantics Family for GLPs). *Let $\mathcal{P}$ be a generalized logic program and I be a model of $\mathcal{P}$. Then*

- *I is a* partial stable model *of $\mathcal{P}$ iff $\Gamma_{\mathcal{P}^+}(I) = I$*
- *I is a* well-founded model *of $\mathcal{P}$ iff I is a k-minimal partial stable model of $\mathcal{P}$*

- *I is a* maximal stable model *of $\mathcal{P}$ iff I is a k-maximal partial stable model of $\mathcal{P}$*
- *I is a* least-undefined stable model *of $\mathcal{P}$ iff I is a partial stable model of $\mathcal{P}$ with subset-minimal $\{A \in At \mid I(A) = \frac{1}{2}\}$*
- *I is a* total stable model *of $\mathcal{P}$ iff I is a partial stable model of $\mathcal{P}$ which is total*

The produced semantics properly generalize existing semantics for normal logic programs.

Proposition 1. *If $\mathcal{P}$ is a normal logic program, the notion of partial stable model in Definition 6 coincides with the definition of partial stable models in [17], the notion of total stable model in Definition 6 coincides with the definition of stable models in [10], the notion of well-founded model in Definition 6 coincides with the definition of well-founded model in [9], and the notions of maximal and least-undefined stable model in Definition 6 coincides with the definition of maximal and least-undefined stable models in [18].*

If $\mathcal{P}$ is a generalized logic program, the definition of stable models in Definition 6 coincides with the definition of stable models in [1].

2.2 Assumption-based Framework

Assumption-based frameworks (ABF) [3] enable to view non-monotonic reasoning as a deduction from assumptions. Argumentation semantics of [7,5] were applied to sets of assumptions. As a consequence, a variety of semantic characterizations of non-monotonic reasoning has been provided.

An ABF is constructed over a deductive system. A *deductive system* is a pair $(\mathcal{L}, \mathcal{R})$ where $\mathcal{L}$ is a language and $\mathcal{R}$ is a set of inference rules over $\mathcal{L}$. A *language* is a set $\mathcal{L}$ of all well-formed sentences. Each *inference rule* r over $\mathcal{L}$ is of the form $\varphi_1, \ldots, \varphi_n \rightarrow \varphi_0$ where $0 \leq n$ and $\varphi_i \in \mathcal{L}$ for each $0 \leq i \leq n$. The sentence $head(r) = \varphi_0$ is called the *head* of r and the set of sentences $body(r) = \{\varphi_1, \ldots, \varphi_n\}$ is called the *body* of r.

A *theory* is a set $S \subseteq \mathcal{L}$ of sentences. A sentence φ is an *immediate consequence* of a theory S iff there exists an inference rule $r \in \mathcal{R}$ with $head(r) = \varphi$ and $body(r) \subseteq S$. A sentence φ is a *consequence* of a theory S iff there is a sequence $\varphi_1, \ldots, \varphi_n$, $0 < n$, of sentences such that $\varphi = \varphi_n$ and for each $0 < i \leq n$ holds $\varphi_i \in S$ or φ_i is an immediate consequence of $\{\varphi_1, \ldots, \varphi_{i-1}\}$. By $Cn_{\mathcal{R}}(S)$ we denote the set of all consequences of S.

An *assumption-based framework* is a tuple $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, {}^-)$ where $(\mathcal{L}, \mathcal{R})$ is a deductive system, $\mathcal{A} \subseteq \mathcal{L}$ is a set of *assumptions*, and ${}^-\colon \mathcal{A} \mapsto \mathcal{L}$ is a mapping called *contrariness function*. We say that $\overline{\alpha}$ is the *contrary* of an assumption α.

A *context* is a set $\Delta \subseteq \mathcal{A}$ of assumptions. We say that Δ is *conflict-free* iff $\{\alpha, \overline{\alpha}\} \not\subseteq Cn_{\mathcal{R}}(\Delta)$ for each assumption α. A context Δ is closed iff $\Delta = Cn_{\mathcal{R}}(\Delta) \cap \mathcal{A}$, i.e., only such assumptions, which are members of Δ, are derivable from Δ. A context Δ *attacks* an assumption α iff $\overline{\alpha} \in Cn_{\mathcal{R}}(\Delta)$. A context Δ *defends* an assumption α iff each closed context attacking α contains an assumption attacked by Δ.

ABFs enable to apply argumentation semantics to sets of assumptions and, consequently, subtle and rich semantic characterizations of sets of assumptions (and of their consequences) can be specified. A closed context Δ is

- *attack-free* iff Δ does not attack an assumption in Δ;
- *admissible* iff Δ is attack-free and defends each assumption in Δ;
- *complete* iff Δ is admissible and contains all assumptions defended by Δ;
- *grounded* iff Δ is a subset-minimal complete context;
- *preferred* iff Δ is a subset-maximal admissible context;
- *semi-stable* iff Δ is a complete context such that $\Delta \cup \{\alpha \in \mathcal{A} \mid \Delta \text{ attacks } \alpha\}$ is subset-maximal;
- *stable* iff Δ is attack-free and attacks each assumption which does not belong to Δ.

3 Defeasible Logic Programs

Our knowledge can be divided according to its epistemological status into two categories: on the one hand, one that is gained by deductively valid reasoning and on the other hand, knowledge that is reached by *defeasible reasoning* [13]. Defeasible logic programs (DeLPs) [15,8,14,11] consider two kinds of rules: *strict* and *defeasible*. Strict rules represent deductive reasoning: whenever their preconditions hold, we accept the conclusion. Defeasible rules formalize tentative knowledge that can be defeated and validity of preconditions of a defeasible rule does not necessarily imply the conclusion. Given a set of literals $\mathcal{L}_{At}$, a strict (resp. defeasible) rule is an expression $L_1, \ldots, L_n \to L_0$ (resp. $L_1, \ldots, L_n \Rightarrow L_0$) where $0 \leq n$ and $L_i \in \mathcal{L}_{At}$ for each $0 \leq i \leq n$. We will use $\rightsquigarrow$ to denote either a strict or a defeasible rule. Each defeasible rule r has assigned its name $name(r)$. The name of r is a default literal from separate language $\mathcal{L}_{\mathcal{N}}$. The intuitive meaning of $name(r) = \text{not}\, A$ is "by default, the defeasible rule r can be used", and consequently, the intuitive meaning of $\text{not}\, name(r) = A$ is "the defeasible rule r can not be used". In the following, we will denote the defeasible rule $r = L_1, \ldots, L_n \Rightarrow L_0$ with $name(r) = \text{not}\, A$ as $A\colon L_1, \ldots, L_n \Rightarrow L_0$.

Definition 7 (Defeasible Logic Program). *Let At be a set of atoms, $\mathcal{N}$ be a set of names, and $At \cap \mathcal{N} = \emptyset$. A* defeasible logic program *is a tuple $\mathcal{P} = (\mathcal{S}, \mathcal{D}, name)$ where $\mathcal{S}$ is a set of* strict rules *over $\mathcal{L}_{At}$, $\mathcal{D}$ is a set of* defeasible rules *over $\mathcal{L}_{At}$, and $name\colon \mathcal{D} \mapsto \text{not}\, \mathcal{N}$ is an injective* naming function.

3.1 From Arguments to Conflict Resolutions

The argumentation process usually consists of five steps [15,16,8,14,11]. At the beginning, a knowledge base is described in some logical language. 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 (also called "defeat relation" in some literature) among conflicting arguments. The status of an argument is then determined by the attack relation. In this

paper, conflicts are not resolved by attacking some of the conflicting arguments, but by attacking some of the weak parts of an argument called *vulnerabilities*. This helps us to satisfy argumentation rationality postulates [4] and to keep the semantics aligned with LP intuitions.

Two kinds of arguments can usually be constructed in the language of defeasible logic programs. Default arguments correspond to default literals. Deductive arguments are constructed by chaining of rules.

We define several functions *prems*, *rules* and *vuls* denoting premises (i.e. default literals) and rules occurring in an argument. Intended meaning of $vuls(A)$ is a set of vulnerabilities of an argument A (i.e., weak parts which can be defeated) consisting of premises and names of defeasible rules of an argument A.

Definition 8 (Argument). *Let $\mathcal{P} = (\mathcal{S}, \mathcal{D}, name)$ be a defeasible logic program. An argument A for a literal L is*

1. *a* default argument *L where L is a default literal.*

$$prems(A) = \{L\}$$
$$rules(A) = \emptyset$$

2. *a* deductive argument *$[A_1, \ldots, A_n \rightsquigarrow L]$ where $0 \leq n$, each A_i, $0 < i \leq n$, is an argument for a literal L_i, and $r = L_1, \ldots, L_n \rightsquigarrow L$ is a rule in $\mathcal{P}$.*

$$prems(A) = prems(A_1) \cup \cdots \cup prems(A_n)$$
$$rules(A) = rules(A_1) \cup \cdots \cup rules(A_n) \cup \{r\}$$

For both kinds of an argument A,

$$vuls(A) = prems(A) \cup name(rules(A) \cap \mathcal{D})$$

Example 2. Consider a defeasible logic program consisting of the only defeasible rule $r\colon \operatorname{not} b \Rightarrow a$. Two default arguments $A_1 = [\operatorname{not} a]$, $A_2 = [\operatorname{not} b]$ and one deductive argument $A_3 = [A_2 \Rightarrow a]$ can be constructed. We can see that $vuls(A_1) = \{\operatorname{not} a\}$, $vuls(A_2) = \{\operatorname{not} b\}$, $vuls(A_3) = \{\operatorname{not} b, \operatorname{not} r\}$.

The difference between a default and a deductive argument for a literal $\operatorname{not} A$ is in the policy of how the conflict is resolved. Syntactical conflict between arguments is formalized in the following definition. As usual in the literature [14], we distinguish two kinds of conflicts: *undermining*[3] and *rebutting*. While an undermining conflict is about a falsification of a hypothesis (assumed by default), a rebutting conflict identifies a situation where opposite claims are derived.

Definition 9 (Conflict). *Let $\mathcal{P}$ be a defeasible logic program. The pair of arguments $C = (A, B)$ is called a* conflict *iff*

- *A is a deductive argument for a default literal $\operatorname{not} L$ and B is a deductive argument for the literal L; or*

[3] Also called *undercutting* in [15].

– *A is a default argument for a default literal* not L *and B is a deductive argument for the literal L.*

The first kind is called a rebutting *conflict and the second kind is called an* undermining *conflict.*

The previous definition just identifies the conflict, but does not say how to resolve it; the notion of *conflict resolution* (to be formalized below) captures a possible way to do so. In our paper, conflicts are not resolved through attack between arguments as in [8,15,14,11], but by attacking some of the vulnerabilities in the conflicting arguments. Since our goal is to define semantics for DeLP respecting existing semantics and intuitions in LP, we assume that all undermining conflicts are resolved in a fixed way as in LP: by attacking the default argument. On the other hand, rebutting conflict is resolved by attacking some defeasible rule. Since, in general, there can be more reasonable ways how to choose which defeasible rules to attack, resolving of all rebutting conflicts is left as domain dependent for the user as an input. Note, that an attack on a defeasible rule r is formalized as an attack on the default literal $name(r)$ which is interpreted as "a defeasible rule r can be used".

Definition 10 (Conflict Resolution). *Let $\mathcal{P}$ be a defeasible logic program. A* conflict resolution *is a tuple $\rho = (A, B, V)$ where $C = (A, B)$ is a conflict, A is an argument for* not L*, and V is a default literal*

– not L *if C is an undermining conflict; or*
– $name(r)$ *where r is a defeasible rule in $rules(A) \cup rules(B)$ if C is a rebutting conflict.*

A conflict resolution strategy *of $\mathcal{P}$ is a set R of conflict resolutions.*

Let $\rho = (A, B, V)$ be a conflict resolution. The contrary of V is called the *resolution* of ρ, and denoted by $res(\rho)$. The set of vulnerabilities of ρ, denoted by $vuls(\rho)$, is defined as:

$$vuls(\rho) = \begin{cases} (vuls(A) \cup vuls(B)) & \text{whenever } V \in vuls(A) \cap vuls(B) \\ (vuls(A) \cup vuls(B)) \setminus \{V\} & \text{otherwise} \end{cases}$$

Usually, there may be more ways how to resolve a conflict and a conflict resolution may resolve other conflicts as well, thus causing other conflict resolutions to be irrelevant or inapplicable. Intuitively, if all vulnerabilities in $vuls(\rho)$ are undefeated (i.e. true), then in order to resolve the conflict in ρ, the contrary $res(\rho)$ of the chosen vulnerability in ρ should be concluded (i.e. true). Notions of $vuls(\rho)$ and $res(\rho)$ will be used for definition of the argumentation semantics in the next subsection.

Example 3. Consider the defeasible logic program $\mathcal{P} = \{\text{not } a \to a\}$ and undercutting arguments $A = [\text{not } a]$ and $B = [[\text{not } a] \to a]$. Then $\rho = (A, B, \text{not } a)$ is a conflict resolution with $res(\rho) = a$ and $vuls(\rho) = \{\text{not } a\}$. Please note that although not a has to be removed to resolve conflict between A and B, it remains a vulnerability of ρ since not a is self-attacking.

Example 4. Consider the following defeasible logic program $\mathcal{P}$

$$r_1 : \Rightarrow a \qquad r_2 : \Rightarrow b \qquad a \to \text{not } c \qquad b \to c$$

and arguments $A = [[\Rightarrow a] \to \text{not } c]$ and $B = [[\Rightarrow b] \to c]$. The rebutting conflict (A, B) can be resolved in two different ways, namely $\rho_1 = (A, B, \text{not } r_1)$ is a conflict resolution with $res(\rho) = r_1$ and $vuls(\rho) = \{\text{not } r_2\}$, and $\rho_2 = (A, B, \text{not } r_2)$ is another conflict resolution with $res(\rho) = r_2$ and $vuls(\rho) = \{\text{not } r_1\}$.

The previous example shows that there are more reasonable ways how to resolve rebutting conflicts. We show two examples of different conflict resolution strategies – the weakest-link and the last-link strategy inspired by ASPIC$^+$ [14]. In both strategies, a user-specified preference order $\prec$ on defeasible rules is assumed. In the last-link strategy, all last-used defeasible rules of conflicting arguments are compared and $\prec$-minimal defeasible rules are chosen as resolutions of the conflict. In the weakest-link strategy, each $\prec$-minimal defeasible rule of conflicting arguments is a resolution of the conflict.

Example 5. Given the defeasible logic program

$$\begin{array}{ll} r_1 : & \Rightarrow b \\ r_2 : & b \Rightarrow a \\ r_3 : & \Rightarrow \text{not } a \end{array}$$

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

$$A_1 = [\Rightarrow b] \qquad A_2 = [A_1 \Rightarrow a] \qquad A_3 = [\Rightarrow \text{not } a]$$

Then $R_1 = \{(A_3, A_2, \text{not } r_3)\}$ is the last-link strategy and $R_2 = \{(A_3, A_2, \text{not } r_1)\}$ is the weakest-link strategy.

In the weakest-link strategy from Example 5, a non-last defeasible rule r_1 is used as a resolution of the conflict. Please note that in [15,8,14,11], such conflict resolutions are not possible, which makes our approach more flexible and general.

3.2 Argumentation Semantics

In the previous subsection, definition of an argument structure, conflicts identification and examples of various conflict resolutions were discussed. However, the status of literals and the actual semantics has not been stated.

Argumentation semantics for defeasible logic programs will be formalized within ABF – a general framework, where several existing non-monotonic formalisms have been embedded [3]. In order to use some of the existing argumentation semantics, we need to specify ABF's language $\mathcal{L}$, set of inference rules $\mathcal{R}$, set of assumptions $\mathcal{A}$, and the contrariness function $^-$. Since ABF provides only one kind of inference rules (i.e. strict), we need to transform defeasible rules into strict. We transform defeasible rule r by adding a new assumption $name(r)$

into the preconditions of r. Furthermore, chosen conflict resolutions R determining how rebutting conflicts will be resolved are transformed into new inference rules. Intuitively, given a conflict resolution ρ, if all assumptions in $vuls(\rho)$ are accepted, then, in order to resolve the conflict in ρ, the atom $res(\rho)$ should be concluded. To achieve this, an inference rule $vuls(\rho) \to res(\rho)$ for each conflict resolution $\rho \in R$ is added to the set of inference rules $\mathcal{R}$.

Definition 11 (Instantiation). *Let $\mathcal{P} = (\mathcal{S}, \mathcal{D}, name)$ be a defeasible logic program built over the language $\mathcal{L}_{At}$ and R be a set of conflict resolutions. An assumption based framework respective to $\mathcal{P}$ and R is $(\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ where*

- $\mathcal{L} = \mathcal{L}_{At} \cup \mathcal{L}_{\mathcal{N}}$,
- $\mathcal{R} = \mathcal{S} \cup \{body(r) \cup \{name(r)\} \to head(r) \mid r \in \mathcal{D}\} \cup \{vuls(\rho) \to res(\rho) \mid \rho \in R\}$,
- $\mathcal{A} = \text{not } At \cup \text{not } \mathcal{N}$,
- $\overline{\text{not } A} = A$ *for each atom* $A \in At \cup \mathcal{N}$.

Example 6. Consider the defeasible logic program $\mathcal{P}$ and the conflict resolution strategy $R = \{\rho_1, \rho_2\}$ from Example 4. Assumption-based framework respective to $\mathcal{P}$ and R is following:

- $\mathcal{L} = \{a, \text{not } a, b, \text{not } b, c, \text{not } c\} \cup \{r_1, \text{not } r_1, r_2, \text{not } r_2\}$
- $R = \{b \to \text{not } c, a \to c\} \cup \{\text{not } r_1 \to b, \text{not } r_2 \to a\} \cup \{\text{not } r_1 \to r_2, \text{not } r_2 \to r_1\}$
- $\mathcal{A} = \{\text{not } a, \text{not } b, \text{not } c\} \cup \{\text{not } r_1, \text{not } r_2\}$
- $\overline{\text{not } A} = A$ for each $A \in \{a, b, c\} \cup \{r_1, r_2\}$

Now we define the actual semantics for defeasible logic programs. Given an ABF $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$, by $\mathcal{F}^+$ we denote its flattening – that is, $\mathcal{F}^+$ is the ABF $(\mathcal{L}, \{r \in \mathcal{R} \mid head(r) \notin \mathcal{A}\}, \mathcal{A}, ^-)$.

Definition 12 (Extension). *Let $\mathcal{P} = (\mathcal{S}, \mathcal{D}, name)$ be a defeasible logic program, R be a set of conflict resolutions, and $\mathcal{F} = (\mathcal{L}, \mathcal{R}, \mathcal{A}, ^-)$ an assumption-based framework respective to $\mathcal{P}$ and R. A set of literals $E \subseteq \mathcal{L}$ is*

1. *a* complete extension *of $\mathcal{P}$ with respect to R iff E is a complete extension of $\mathcal{F}^+$ with $Cn_{\mathcal{R}}(E) \subseteq E$ and $Cn_{\mathcal{R}}(E') \subseteq E'$;*
2. *a* grounded extension *of $\mathcal{P}$ with respect to R iff E is a subset-minimal complete extension of $\mathcal{P}$ with respect to R;*
3. *a* preferred extension *of $\mathcal{P}$ with respect to R iff E is a subset-maximal complete extension of $\mathcal{P}$ with respect to R;*
4. *a* semi-stable extension *of $\mathcal{P}$ with respect to R iff E is a complete extension of $\mathcal{P}$ with respect to R with subset-minimal $E' \setminus E$;*
5. *a* stable extension *of $\mathcal{P}$ with respect to R iff E is a complete extension of $\mathcal{P}$ with respect to R and $E' = E$.*

where $E' = \mathcal{L} \setminus \text{not } E$.

Example 7. Consider the assumption-based framework from Example 6. Then $E_1 = \emptyset$, $E_2 = \{\text{not } r_1, r_2, \text{not } a, b, \text{not } c\}$, and $E_3 = \{r_1, \text{not } r_2, a, \text{not } b, c\}$ are complete extensions of $\mathcal{P}$ with respect to R. Furthermore, E_1 is the grounded extension and E_2, E_3 are preferred, semi-stable and stable extensions of $\mathcal{P}$ with respect to R.

3.3 Transformational Semantics

The argumentation semantics defined above allows us to deal with conflicting rules and to identify the extensions of a DeLP, given a CRS, and hence it constitutes a reference semantics. This semantics is comparable to existing argumentation-based semantics for DeLP, and, as we show below, it satisfies the expected desired properties of defeasible reasoning. In this section we investigate on the relation of the argumentation-based semantics and classical logic programming. As we show, an equivalent semantics can be obtained by transforming the DeLP and the given CRS into a classical logic program, and computing the respective class of models.

In fact the transformation that is required is essentially the same which we used to embed DeLPs with CRS into ABFs. The names of rules become new literals in the language, intuitively if $name(r)$ becomes true it means that the respective defeasible rule is defeated. By default $name(r)$ holds and so all defeasible rules can be used unless the program proves otherwise. The conflict resolution strategy R to be used is encoded by adding rules of the form $vuls(\rho) \rightarrow res(\rho)$ for each conflict resolution $\rho \in R$, where the head of such rules is always an atom and the body is a set of default literals.

Formally the transformation is defined as follows:

Definition 13 (Transformation). *Let $\mathcal{P} = (\mathcal{S}, \mathcal{D}, name)$ be a defeasible logic program and R be a set of conflict resolutions.* Transformation *of $\mathcal{P}$ with respect to R into a generalized logic program, denoted as $T(\mathcal{P}, R)$, is defined as*

$$T(\mathcal{P}, R) = \mathcal{S} \cup \{body(r) \cup \{name(r)\} \rightarrow head(r) \mid r \in \mathcal{D}\} \cup \{vuls(\rho) \rightarrow res(\rho) \mid \rho \in R\}$$

Thanks to the transformation, we can now compute the semantics of each DeLP, relying on the semantics of generalized logic programs. Given a DeLP $\mathcal{P}$ and the assumed CRS R, the extensions of $\mathcal{P}$ w.r.t. R corresponds to the respective class of models. Complete extensions correspond to partial stable models, the grounded extension to the well-founded model, preferred extensions to maximal stable models, semi-stable extensions to least-undefined stable models, and stable extensions to total stable models.

Proposition 2. *Let $\mathcal{P}$ be a defeasible logic program and R be a set of conflict resolutions. Then*

1. *E is a complete extension of $\mathcal{P}$ with respect to R iff E is a partial stable model of $T(\mathcal{P}, R)$.*

2. *E is a grounded extension of $\mathcal{P}$ with respect to R iff E is a well-founded model of $T(\mathcal{P}, R)$.*
3. *E is a preferred extension of $\mathcal{P}$ with respect to R iff E is a maximal partial stable model of $T(\mathcal{P}, R)$.*
4. *E is a semi-stable extension of $\mathcal{P}$ with respect to R iff E is a least-undefined stable model of $T(\mathcal{P}, R)$.*
5. *E is a stable extension of $\mathcal{P}$ with respect to R iff E is a total stable model of $T(\mathcal{P}, R)$.*

A remarkable special case happens when the input program $\mathcal{P}$ does not contain defeasible rules, and hence it is a regular GLP. In such a case our argumentation-based semantics exactly corresponds to the respective class of models for the GLP. This shows complete backward compatibility of our semantics with the underlying class of logic programs.

Proposition 3. *Let $\mathcal{S}$ be a generalized logic program and $\mathcal{P} = (\mathcal{S}, \emptyset, \emptyset)$ a defeasible logic program with the empty set of conflict resolutions. Then*

1. *E is a complete extension of $\mathcal{P}$ iff E is a partial stable model of $\mathcal{S}$.*
2. *E is a grounded extension of $\mathcal{P}$ iff E is a well-founded model of $\mathcal{S}$.*
3. *E is a preferred extension of $\mathcal{P}$ iff E is a maximal partial stable model of $\mathcal{S}$.*
4. *E is a semi-stable extension of $\mathcal{P}$ iff E is a least-undefined partial stable model of $\mathcal{S}$.*
5. *E is a stable extension of $\mathcal{P}$ iff E is a total stable model of $\mathcal{S}$.*

4 Properties

In this section we will have a look on desired properties for defeasible reasoning, and show that our semantics satisfies these properties. The properties are formulated in general, that is, they should be satisfied for any a defeasible logic program $\mathcal{P}$, any set of conflict resolutions R, and any extension E of $\mathcal{P}$ w.r.t. R.

The first two properties formulated below are well known, they were proposed by Caminada and Amgoud [4]. The closure property originally requires that nothing new can be derived from the extension using strict rules. We use a slightly more general formulation, nothing should be derived using the consequence operator Cn which applies all the strict rules and in addition also all the defeasible rules which are not defeated according to R. The original property [4] is a straightforward consequence of this.

Property 1 (Closure). Let $\mathcal{R}' = \mathcal{S} \cup \{body(r) \cup \{name(r)\} \rightarrow head(r) \mid r \in \mathcal{D}\}$. Then $Cn_{\mathcal{R}'}(E) \subseteq E$.

The consistency property [4] formally requires that all conflicts must be resolved in any extension.

Property 2 (Consistency). E is consistent.

In addition we propose two new desired properties which are concerned with handling of the default assumptions. Reasoning with default assumptions is a fixed part of the semantics of GLPs (and most other classes of logic programs), and therefore in DeLPs it should be governed by similar principles. The first property (dubbed *positive defeat*) requires that for each default literal not A, this literal should be always defeated in any extension E such that there is a conflict resolution $\rho \in R$ whose assumptions are all upheld by E; and, in such a case the opposite literal A should be part of the extension E.

Property 3 (Positive Defeat). For each atom $A \in \mathcal{L}_{\mathcal{N}}$, if there exists a conflict resolution $\rho \in R$ with $res(\rho) = A$ and $vuls(\rho) \subseteq E$ then $A \in E$.

The previous property handles all cases when there is an undefeated evidence against not A and requires that A should hold. The next property (dubbed *negative defeat*) handles the opposite case. If there is no undefeated evidence against not A, in terms of a conflict resolution $\rho \in R$ whose assumptions are all upheld by E, then not A should be part of the extension E.

Property 4 (Negative Defeat). For each default literal not $A \in \mathcal{L}_{\mathcal{N}}$, if for each conflict resolution $\rho \in R$ with $res(\rho) = A$ holds not $vuls(\rho) \cap E \neq \emptyset$ then not $A \in E$.

Closure and consistency trivially hold for our semantics, as the semantics was designed with these properties in mind. They are assured by the definition of complete extension of a DeLP (Definition 12).

Proposition 4. *Each complete extension E of a defeasible logic program $\mathcal{P}$ with respect to a set of conflict resolutions R has the property of closure.*

Proposition 5. *Each complete extension E of a defeasible logic program $\mathcal{P}$ with respect to a set of conflict resolutions R is consistent.*

Satisfaction of a positive and a negative defeat properties follow from the instantiation of an ABF (Definition 11), where an inference rule $vuls(\rho) \rightarrow res(\rho)$ is added for every conflict resolution $\rho \in R$.

Proposition 6. *Each complete extension E of a defeasible logic program $\mathcal{P}$ with respect to a set of conflict resolutions R has the property of positive defeat.*

Proposition 7. *Each complete extension E of a defeasible logic program $\mathcal{P}$ with respect to a set of conflict resolutions R has the property of negative defeat.*

5 Related Work

There are two well-known argumentation-based formalisms with defeasible inference rules – defeasible logic programs [8] and ASPIC$^+$ [14,11]. It is known that the semantics in [8] does not satisfy rationality postulates formalized in [4], especially the closure property. Although ASPIC$^+$ satisfies all postulates in [4],

it uses transposition or contraposition which violate directionality of inference rules [2] and thus violating LP intuitions. It also does not satisfy positive or negative defeat property introduced in this paper.

Francesca Toni's paper [19] describes a mapping of defeasible reasoning into assumption-based argumentation framework. The work takes into account the properties [4] that we also consider (closedness and consistency), and it is formally proven that the constructed assumption-based argumentation framework's semantics is closed and consistent. However no explicit connection with existing LP semantics is discussed in [19].

The paper [20] does not deal with DeLP, but on how to encode defeasible semantics inside logic programs. The main objective is on explicating a preference ordering on defeasible rules inside a logic program, so that defeats (between defeasible logic rules) are properly encoded in LP. This is achieved with a special predicate *defeated* with special semantics.

Caminada et al. [6] investigated how abstract argumentation semantics and semantics for normal logic programs are related. Authors found out that abstract argumentation is about minimizing/maximizing argument labellings, whereas logic programming is about minimizing/maximizing conclusion labellings. Further, they proved that abstract argumentation semantics cannot capture the least-undefined stable semantics for normal logic programs.

6 Conclusion

In this paper we investigated the question of how to define semantics for defeasible logic programs, which satisfies both the existing rationality postulates from the area of structured argumentation and is also aligned with LP semantics and intuitions. To achieve these objectives, we diverged from the usual argumentation process methodology. Most importantly, conflicts are not resolved by attacking some of the conflicting arguments, but by attacking some of the weak parts of an argument called vulnerabilities. Main contributions are as follows:

- We presented an argumentation semantics of defeasible logic programs, based on conflict resolution strategies within assumption-based frameworks, whose semantics satisfies desired properties like consistency and closedness under the set of strict rules.
- We constructed a transformational semantics, which takes a defeasible logic program and a conflict resolution strategy as an input, and generates a corresponding generalized logic program. As a consequence, a computation of argumentation semantics of DeLP in the frame of GLP is enabled.
- Equivalence of a transformational and an argumentation semantics is provided.

Acknowledgements

This work resulted from the Slovak–Greek bilateral project "Multi-context Reasoning in Heterogeneous environments", registered on the Slovak side under no.

SK-GR-0070-11 with the APVV agency and co-financed by the Greek General Secretariat of Science and Technology and the European Union. It was further supported from the Slovak national VEGA project no. 1/1333/12. Martin Baláž and Martin Homola are also supported from APVV project no. APVV-0513-10.

References

1. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.C.: Dynamic logic programming. In: APPIA-GULP-PRODE 1998. pp. 393–408 (1998)
2. Baláž, M., Frtús, J., Homola, M.: Conflict resolution in structured argumentation. In: LPAR-19 (Short Papers). EPiC, vol. 26, pp. 23–34. EasyChair (2014)
3. Bondarenko, A., Dung, P.M., Kowalski, R.A., Toni, F.: An abstract, argumentation theoretic approach to default reasoning. Artif. Intell. 93(1-2), 63–101 (1997)
4. Caminada, M., Amgoud, L.: On the evaluation of argumentation formalisms. Artif. Intell. 171(5-6), 286–310 (2007)
5. Caminada, M., Carnielli, W.A., Dunne, P.E.: Semi-stable semantics. J. Log. Comput. 22(5), 1207–1254 (2011)
6. Caminada, M., Sá, S., Alcântara, J.: On the equivalence between logic programming semantics and argumentation semantics. In: ECSQARU 2014 (2013)
7. Dung, P.M.: On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. Artif. Intell. 77, 321–357 (1995)
8. García, A.J., Simari, G.R.: Defeasible logic programming: an argumentative approach. TPLP 4(2), 95–138 (2004)
9. Gelder, A.V., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. J. ACM 38(3), 619–649 (1991)
10. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: ICLP/SLP. pp. 1070–1080. MIT Press (1988)
11. Modgil, S., Prakken, H.: Revisiting preferences and argumentation. In: IJCAI 2001. pp. 1021–1026. AAAI Press (2011)
12. Nute, D.: Defeasible reasoning and decision support systems. Decision Support Systems 4(1), 97–110 (1988)
13. Pollock, J.L.: Defeasible reasoning. Cognitive Science 11(4), 481–518 (1987)
14. Prakken, H.: An abstract framework for argumentation with structured arguments. Argument & Computation 1(2), 93–124 (2010)
15. Prakken, H., Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of Applied Nonclassical Logics 7(1), 25–75 (1997)
16. Prakken, H., Vreeswijk, G.: Logics for defeasible argumentation. In: Gabbay, D., Guenthner, F. (eds.) Handbook of Philosophical Logic, pp. 219–318. Springer (2002)
17. Przymusinski, T.C.: The well-founded semantics coincides with the three-valued stable semantics. Fundam. Inform. 13(4), 445–463 (1990)
18. Saccà, D.: The expressive powers of stable models for bound and unbound DATALOG queries. J. Comput. Syst. Sci. 54(3), 441–464 (Jun 1997)
19. Toni, F.: Assumption-based argumentation for closed and consistent defeasible reasoning. In: JSAI 2007. LNCS, vol. 4914, pp. 390–402. Springer (2008)
20. Wan, H., Grosof, B., Kifer, M., Fodor, P., Liang, S.: Logic programming with defaults and argumentation theories. In: ICLP 2009. LNCS, vol. 5649, pp. 432–448. Springer (2009)