On the Application of Argument Accrual to Reasoning with Inconsistent Possibilistic Ontologies

Sergio Alejandro Gómez

Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering
Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
Email: sag@cs.uns.edu.ar

Abstract. We present an approach for performing instance checking in a suitable subset of possibilistic description logic programming ontologies by using argument accrual. Ontologies are interpreted in possibilistic logic programming under Dung's grounded semantics. We present a reasoning framework with a case study and a Java-based implementation for enacting the proposed approach.

Keywords: Argument accrual, ontology reasoning, inconsistency handling, Description Logics.

1 Introduction

Reasoning with inconsistent ontologies plays a fundamental role in Semantic Web applications. An ontology defines axiomatically a set of concepts that, given assertions about the membership of individuals to some concepts, allows to determine the membership of individuals to concepts (a task known as instance checking). Ontologies can suffer from incoherence and inconsistency; either correcting these anomalies or dealing with them with non-monotonic reasoning techniques are the two main accepted solutions [1]. Argumentation [2] is an approach to defeasible reasoning that can be applied to handling inconsistency in ontologies. In argumentation, given an inconsistent knowledge base, arguments compete to decide which are the accepted consequences. Argument accrual formalizes the notion that having more arguments for a certain conclusion makes it more credible [3].

In this paper we explore the application of argument accrual to reasoning with inconsistent ontologies. To the best of our knowledge, argument accrual were firstly studied by [4] and its application to the problem of ontology reasoning has only been suggested by Groza [5] in the context of fuzzy description logics. We use possibilistic description logic programming as the language for ontology representation (i.e. an ontology is ultimately interpreted as a possibilistic logic program). Arguments are then computed and accrued to compute the membership of instances to concepts using structured argumentation under Dung's

grounded semantics. Our approach is qualitative providing a case study to show how our approach works, and a downloadable Java-based implementation for enacting our results.

The rest of the paper is structured as follows. In Sect. 2 we recall the fundamentals of possibilistic description logic ontologies. In Sect. 3 we review how argumentation under Dung's semantics with arguments expressed in possibilistic logic programming is achieved and how possibilistic description logic ontologies can be interpreted as possibilistic logic programs. In Sect. 4 we introduce how to reason with inconsistent possibilistic ontologies using argument accrual. We conclude in Sect. 5.

2 Possibilistic Description Logic Ontologies

Classical Description Logics. Description Logics (DL) are a well-known family of knowledge representation formalisms [6]. In this work, we will consider a very tight subset $\mathcal{L}_{DL}$ of DLs, to which we will restrict our discussion, based on the notions of *concepts* (unary predicates, classes). Concept descriptions are built from concept names $C, D, \ldots$ using the constructors conjunction ($C \sqcap D$), disjunction ($C \sqcup D$) and negation ($\neg C$). The empty concept is denoted by $\bot$. A DL ontology $\Sigma = (T, A)$ consists of two finite and mutually disjoint sets: a *Tbox* T which introduces the *terminology* and an *Abox* A (assertional box) which contains facts about particular objects in the application domain. The Tbox contains inclusion axioms $C \sqsubseteq D$, where C and D are (possibly complex) concept descriptions, meaning that every individual of C is also a D. Objects in the Abox are referred to by a finite number of *individual names* and these names may be used in assertional statements $a : C$ (meaning the individual a is a member of concept C).

Description Logic Programming. In this work we are only interested in the reasoning task known as *instance checking* that refers to determining if an individual a is a member of a concept C. Assigning semantics to a DL ontology can be done based on that DL is isomorphic with first-order logic restricted to two variables. Then, $C \sqsubseteq D$ can be interpreted as the formula $(\forall x)(c(x) \rightarrow d(x))$ and $a : C$ as $c(a)$. Description Logic Programming (DLP) approaches [7] take advantage of this to interpret those expressions as the Prolog rules "d(X) :- c(X)." and "c(a).", resp. Therefore instance checking of a is a member of C reduces to proving the goal "$\leftarrow$ c(a)".

Possibilistic Description Logics. We now recall the fundamentals of possibilistic description logic ontologies [8, 9]. A *possibilistic DL ontology* is a set of possibilistic axioms of the form $(\varphi, W(\varphi))$ where φ is an axiom expressed in $\mathcal{L}_{DL}$ and $W(\varphi) \in [0, 1]$ is the degree of certainty (or priority) of φ. Namely, a possibilistic DL ontology Σ is such that $\Sigma = \{(\varphi_i, W(\varphi_i)) : i = 1, \ldots, n\}$. Only somewhat certain information is explicitly represented in a possibilistic ontology. That is, axioms with a null weight ($W(\varphi) = 0$) are not explicitly represented

in the knowledge base. The weighted axiom $(\varphi, W(\varphi))$ means that the certainty degree of φ is at least equal to $W(\varphi)$. A possibilistic DL ontology Σ will also be represented by a pair $\Sigma = (T, A)$ where elements in both T and A may be uncertain. Note that if we consider all $W(\varphi_i) = 1$, then we find a classical DL ontology $\Sigma^* = \{\varphi_i : (\varphi_i, W(\varphi_i)) \in \Sigma\}$. We say that Σ is consistent if the classical ontology obtained from Σ by ignoring the weights associated with axioms is consistent, and inconsistent otherwise. Notice that the weights $W(\cdot)$ for axioms must be provided by the knowledge engineer that designs the knowledge base, thus providing the relative importance of axioms/rules and assertions/facts.

Example 1. (Originally presented in [10].) Let $\Sigma_1 = (T, A)$ be the ontology modeling a variation of the famous Tweety example from the non-monotonic literature. It expresses that a bird usually flies, all penguins are birds, penguins do not usually fly, birds with broken wings normally do not fly either, and pilots can almost always fly. It is known that Tweety is a penguin with almost certainly a broken wing and most likely a pilot. Formally:

$$T = \left\{ \begin{array}{lll} (\mathsf{Bird} \sqsubseteq \mathsf{Flies}, 0.6), & (\mathsf{Penguin} \sqsubseteq \mathsf{Bird}, 1.0), & (\mathsf{Pilot} \sqsubseteq \mathsf{Flies}, 0.9) \\ (\mathsf{Penguin} \sqsubseteq \neg\mathsf{Flies}, 0.8), & (\mathsf{Bird} \sqcap \mathsf{BrokenWing} \sqsubseteq \neg\mathsf{Flies}, 0.7), & \end{array} \right\}$$

$$A = \left\{ \begin{array}{ll} (\mathsf{TWEETY} : \mathsf{BrokenWing}, 0.8), & (\mathsf{TWEETY} : \mathsf{Penguin}, 1.0), \\ (\mathsf{TWEETY} : \mathsf{Pilot}, 0.9) & \end{array} \right\}$$

In Σ_1^*, because Tweety a penguin (and therefore a bird), he is both a member of Flies and $\neg\mathsf{Flies}$, meaning that he is a member of $\bot$. Traditional reasoners are not able to infer anything from such an inconsistent ontology, thus invalidating even reasoning with consistent subsets of the offending ontology.

3 Ontology Reasoning in Possibilistic Argumentation *à la* Dung

We now recall how to reason with possibly inconsistent ontologies by using Dung-style argumentation, an approach we originally explored in [10].

3.1 Notions of Possibilistic Defeasible Logic Programming

The P-DeLP [11] language $\mathcal{L}$ is defined from a set of ground fuzzy atoms (fuzzy propositional variables) $p, q, \ldots$ along with the connectives $\sim$ (strong negation), $\wedge$ (written as a comma in Prolog clauses) and $\leftarrow$. A literal $L \in \mathcal{L}$ is a ground (fuzzy) atom $\sim q$, where q is a ground (fuzzy) propositional variable. A *rule* in $\mathcal{L}$ is a formula of the form $Q \leftarrow L_1 \wedge \ldots \wedge L_n$, where $Q, L_1, \ldots, L_n$ are literals in $\mathcal{L}$. When $n = 0$, the formula $Q \leftarrow$ is called a *fact*. The term *goal* will refer to any literal $Q \in \mathcal{L}$. Facts, rules and goals are the well-formed formulas in $\mathcal{L}$. A *certainty-weighted clause*, or simply weighted clause, is a pair (φ, α), where φ is a formula in $\mathcal{L}$ and $\alpha \in [0, 1]$ expresses a lower bound for the certainty of φ in terms of a necessity measure.

The proof method for P-DeLP formulas, noted $\vdash$, is based on the generalized modus ponens rule, that from $(P \leftarrow Q_1, \ldots, Q_k, \gamma)$ and $(Q_1, \beta_1), \ldots, (Q_k, \beta_k)$ allows to infer $(P, \min(\gamma, \beta_1, \ldots, \beta_k))$. A clause (φ, α) is referred to as certain when $\alpha = 1$ and uncertain otherwise. A set of clauses Γ is deemed as *contradictory*, denoted $\Gamma \vdash \bot$, when $\Gamma \vdash (Q, \alpha)$ and $\Gamma \vdash (\sim Q, \beta)$, with $\alpha > 0$ and $\beta > 0$, for some atom Q in $\mathcal{L}$. A program is a set of weighted rules and facts in $\mathcal{L}$ in which certain and uncertain information is distinguished. As an additional requirement, certain knowledge is required to be non-contradictory. A *P-DeLP program* $\mathcal{P}$ (or just *program* $\mathcal{P}$) is a pair (Π, Δ), where Π is a non-contradictory finite set of certain clauses, and Δ is a finite set of uncertain clauses. We build arguments $\mathcal{A}$ for Q with weight γ, noted as $\langle \mathcal{A}, Q, \gamma \rangle$, inductively: If (Q, γ) is a fact, then $\langle \{(Q, \gamma)\}, Q, \gamma \rangle$ is an argument; from $(P \leftarrow Q_1, \ldots, Q_n, \gamma)$ and n arguments $\langle \mathcal{A}_1, Q_1, \beta_1 \rangle$, $\ldots$, $\langle \mathcal{A}_n, Q_n, \beta_n \rangle$, when $Q_1, \ldots, Q_n, P$ is consistent, then we get $\langle \bigcup_{i=1}^{n} \mathcal{A}_i \cup \{(P \leftarrow Q_1, \ldots, Q_n, \gamma)\}, P, \min(\beta_1, \ldots, \beta_n, \gamma) \rangle$.

Conflict among arguments is formalized by the notions of counterargument and defeat. Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments in $\mathcal{P}$. We say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ *counterargues* $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff there exists a subargument (called *disagreement subargument*) $\langle \mathcal{S}, Q, \beta \rangle$ of $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ such that $\Pi \cup \{(Q_1, \alpha_1), (Q, \beta)\}$ is contradictory. The literal (Q, β) is called *disagreement literal*. Defeat among arguments involves the consideration of *preference criteria* defined on the set of arguments. The criterion applied here is based on necessity measures associated with arguments. Let $\mathcal{P}$ be a program, and let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments in $\mathcal{P}$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a *defeater* for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ iff $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counterargues argument $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ with disagreement subargument $\langle \mathcal{A}, Q, \alpha \rangle$, with $\alpha_1 \geq \alpha$. If $\alpha_1 > \alpha$ then $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is called a *proper defeater*, otherwise ($\alpha_1 = \alpha$) it is called a *blocking defeater*. Notice that we digress from the original P-DeLP formalism of Chesñevar et al. [11] in that (i) we include facts in the support of arguments and (ii) facts are allowed to have a weight different than one (so allowing them to be considered as presumptions).

Example 2. Consider again the ontology Σ_1 presented in Ex. 1. This ontology is interpreted as the equivalent possibilistic program $\mathcal{P}_1$ where:

$$\mathcal{P}_1 = \left\{ \begin{array}{ll} (\mathit{brokenWing}(\mathit{tweety}), 0.8), & (\mathit{penguin}(\mathit{tweety}), 1.0), \\ (\mathit{pilot}(\mathit{tweety}), 0.9), & \\ (\mathit{flies}(X) \leftarrow \mathit{bird}(X), 0.6), & \\ (\mathit{bird}(X) \leftarrow \mathit{penguin}(X), 1.0), & \\ (\sim \mathit{flies}(X) \leftarrow \mathit{penguin}(X), 0.8), & \\ (\sim \mathit{flies}(X) \leftarrow \mathit{bird}(X), \mathit{brokenWing}(X), 0.7), & \\ (\mathit{flies}(X) \leftarrow \mathit{pilot}(X), 0.9) & \end{array} \right\}.$$

Exactly 8 arguments can be built from this program (notice that they coincide with the ones presented in Ex. 3):

- $\langle \mathcal{A}_1, \mathit{brokenWing}(\mathit{tweety}), 0.8 \rangle$ where $\mathcal{A}_1 = \{ (\mathit{brokenWing}(\mathit{tweety}), 0.8) \}$
- $\langle \mathcal{A}_2, \mathit{penguin}(\mathit{tweety}), 1.0 \rangle$ where $\mathcal{A}_2 = \{ (\mathit{penguin}(\mathit{tweety}), 1.0) \}$
- $\langle \mathcal{A}_3, \mathit{pilot}(\mathit{tweety}), 0.9 \rangle$ where $\mathcal{A}_3 = \{ (\mathit{pilot}(\mathit{tweety}), 0.9) \}$

- $\langle \mathcal{A}_4, \mathit{bird}(\mathit{tweety}), 1.0\rangle$ where $\mathcal{A}_4 = \{ (\mathit{bird}(\mathit{tweety}) \leftarrow \mathit{penguin}(\mathit{tweety}), 1.0) \} \cup \mathcal{A}_2$
- $\langle \mathcal{A}_5, \sim\mathit{flies}(\mathit{tweety}), 0.8\rangle$ where

$$\mathcal{A}_5 = \{ (\sim\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{penguin}(\mathit{tweety}), 0.8) \} \cup \mathcal{A}_2$$

- $\langle \mathcal{A}_6, \sim\mathit{flies}(\mathit{tweety}), 0.7\rangle$ where

$$\mathcal{A}_6 = \{ (\sim\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{bird}(\mathit{tweety}), \mathit{brokenWing}(\mathit{tweety}), 0.7) \} \cup \mathcal{A}_1 \cup \mathcal{A}_4$$

- $\langle \mathcal{A}_7, \mathit{flies}(\mathit{tweety}), 0.9\rangle$ where $\mathcal{A}_7 = \{ (\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{pilot}(\mathit{tweety}), 0.9) \} \cup \mathcal{A}_3$
- $\langle \mathcal{A}_8, \mathit{flies}(\mathit{tweety}), 0.6\rangle$ where $\mathcal{A}_8 = \{ (\mathit{flies}(\mathit{tweety}) \leftarrow \mathit{bird}(\mathit{tweety}), 0.6) \} \cup \mathcal{A}_4$

The attacks among these arguments are exactly those presented in Fig. 1. Notice that here the attacks are made into final conclusions (thus they are direct attacks). Nonetheless the reasoning framework presented here and the application we built also allow modeling attacks into premises (i.e. indirect attacks).

3.2 Dung-Style Abstract Argumentation

Abstract argumentation frameworks [12] do not presuppose any internal structure of arguments, thus considering only the interactions of arguments by means of an attack relation between arguments. An *abstract argumentation framework* $\mathcal{AF}$ is a pair $(Arg, \rightarrow)$ where Arg is a set of arguments and $\rightarrow$ is a relation of Arg into Arg. For two arguments $\mathcal{A}$ and $\mathcal{B}$ in Arg, the relation $\mathcal{A} \rightarrow \mathcal{B}$ means that the argument $\mathcal{A}$ attacks the argument $\mathcal{B}$. Abstract argumentation frameworks can be concisely represented by directed graphs, where arguments are represented as nodes and edges model the attack relation.

Example 3. Consider the argumentation framework $\mathcal{AF}_3 = (Arg, \rightarrow)$ where $Arg = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_5, \mathcal{A}_6, \mathcal{A}_7, \mathcal{A}_8\}$ and $\rightarrow = \{(\mathcal{A}_5, \mathcal{A}_8), (\mathcal{A}_6, \mathcal{A}_8), (\mathcal{A}_7, \mathcal{A}_5), (\mathcal{A}_7, \mathcal{A}_6)\}$. The framework is shown graphically in Fig. 1 and, although it is not necessary from a mathematical viewpoint, we can assign meaning to the above arguments to provide some intuition (notice that these arguments coincide with the ones in Ex. 2):

- $\mathcal{A}_1$: Tweety has a broken wing
- $\mathcal{A}_2$: Tweety is a penguin
- $\mathcal{A}_3$: Tweety is a pilot
- $\mathcal{A}_4$: Tweety is a bird
- $\mathcal{A}_5$: Tweety does not fly because he is a penguin and penguins do not usually fly
- $\mathcal{A}_6$: Tweety does not fly because he has a broken wing
- $\mathcal{A}_7$: Tweety flies because he is also a pilot
- $\mathcal{A}_8$: Tweety flies because he is a bird and birds normally fly

Semantics are usually given to abstract argumentation frameworks by means of extensions. An extension E of an argumentation framework $\mathcal{AF} = (Arg, \rightarrow)$ is subset of Arg that gives some coherent view on the argumentation underlying $\mathcal{AF}$. In this work, we will reason under grounded semantics despite that other semantics have been proposed. Let $\mathcal{AF} = (Arg, \rightarrow)$ be an argumentation framework. An extension $E \subseteq Arg$ is *conflict-free* iff there are no $\mathcal{A}, \mathcal{B} \in E$

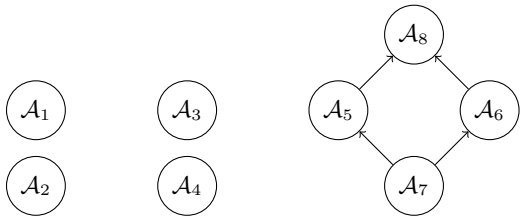

Fig. 1. Abstract argumentation framework presented in Ex. 3

with $\mathcal{A} \rightarrow \mathcal{B}$. An argument $\mathcal{A} \in Arg$ is *acceptable* with respect to an extension $E \subseteq Arg$ iff for every $\mathcal{B} \in Arg$ with $\mathcal{B} \rightarrow \mathcal{A}$ there is a $\mathcal{A}' \in E$ with $\mathcal{A}' \rightarrow \mathcal{B}$. An extension $E \subseteq Arg$ is *admissible* iff it is conflict free and all $\mathcal{A} \in E$ are acceptable with respect to E. An extension $E \subseteq Arg$ is *complete* iff it is admissible and there is no $\mathcal{A} \in Arg \backslash E$ that is acceptable with respect to E. An extension $E \subseteq Arg$ is *grounded* iff it is complete and E is minimal with respect to set inclusion.

The intuition behind admissibility is that an argument can only be accepted if there are no attackers that are accepted and if an argument is not accepted then there has to be an acceptable argument attacking it. The idea behind the completeness property is that all acceptable arguments should be accepted. The grounded extension is the minimal set of acceptable arguments and uniquely determined. It can be computed as follows: first, all arguments that have no attackers are added to the empty extension E and those arguments and all arguments that are attacked by one of these arguments are removed from the framework; then the process is repeated; if one obtains a framework where there is no unattacked arguments, the remaining arguments are also removed.

Example 4. Consider again the argumentation framework $\mathcal{AF}_3$ presented in Ex. 3. The grounded extension E of $\mathcal{AF}_3$ is given by $E = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_7, \mathcal{A}_8\}$.

3.3 Expressing Possibilistic DL Ontologies as Possibilistic Logic Programs

Grosof et al. [7] provide a way of expressing a subset of Description Logic ontologies in logic programming, namely the description logic programming subset of DL that can be expressed as Horn knowledge bases. The idea consist of expressing both DL assertional statements and terminological axioms as equivalent Horn clauses. We will explain only the part of the algorithm relevant to this work.

Given an ontology $\Sigma = (T, A)$, for every terminological axiom or assertional statement $(\varphi, W(\phi))$ we will generate a possibilistic axiom $(\mathcal{T}(\varphi), W(\varphi))$, where $\mathcal{T}(\cdot)$ is the transformation function from the language of description logics to the language of Horn clauses. The specification of the $\mathcal{T}$ is as follows: Assertional statements in A of the form $a : C$ are expressed as facts $c(a)$. We obtain an equivalent ontology composed only of inclusion axioms of the form $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq D$ which are expressed as Horn clauses of the form $d(X) \leftarrow c_1(X), \ldots, c_n(X)$.

Given a possibilistic DL ontology Σ, Σ is expressed as an equivalent P-DeLP program $\mathcal{P}$. With this program, a grounded extension E will be computed. If $(c(a), \alpha)$ belongs to E then we will say that the individual A is a member of the concept C with certainty degree α.

Example 5. Recall from Ex. 4 that $E = \{\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3, \mathcal{A}_4, \mathcal{A}_7, \mathcal{A}_8\}$. Therefore we can affirm that, from $\mathcal{A}_1$, TWEETY is a member of BrokenWing with certainty degree 0.8; from $\mathcal{A}_2$, TWEETY is a member of Penguin with certainty degree 1.0; from $\mathcal{A}_3$, TWEETY is a member of Pilot with certainty degree 0.9; from $\mathcal{A}_4$, TWEETY is a member of Bird with certainty degree 1.0; from $\mathcal{A}_7$, TWEETY is a member of Flies with certainty degree 0.9, and from $\mathcal{A}_8$, TWEETY is a member of Flies with certainty degree 0.6.

From $\mathcal{A}_7$ and $\mathcal{A}_8$, as Tweety is both a member of Flies with both degrees 0.6 and 0.9, we take a credulous approach considering that TWEETY is a member with degree 0.9.

4 Ontology Reasoning with Argument Accrual

Now we deal with the problem of accruing arguments for justifying the membership of individuals to concepts, thus redefining the task of instance checking by means of argument accrual. Our approach relies in previous work of Gómez Lucero et al. by presenting a variation of their approach that we apply to ontology reasoning. Gómez Lucero et al. [3] presented an approach to model accrual of arguments in a possibilistic setting where, given different arguments supporting the same conclusion, they are able to accumulate their strength in terms of possibilistic values. For this, they define the notion of *accrued structure* whose necessity degree is computed in terms of two mutually recursive functions: $f^{+}_{\Phi}(\cdot)$ (the accruing function) and $f^{MP}_{\Phi}(\cdot)$ (that propagates necessity degrees). The latter is parameterized w.r.t. a user-defined function ACC that supports *non-depreciation* (i.e. accruing arguments results in a necessity degree no lower than any single argument involved in the accrual) and *maximality* (i.e. accrual means total certainty only if there is an argument with necessity degree 1). We recall the notion of *argument accrual* as interpreted by [3]:

Definition 1. *Let $\mathcal{P}$ be a P-DeLP program and let Ω be a set of arguments in $\mathcal{P}$ supporting the same conclusion H, i.e. $\Omega = \{\langle \mathcal{A}_1, H, \alpha_1\rangle, \ldots, \langle \mathcal{A}_n, H, \alpha_n\rangle\}$. The accrued structure $\mathcal{AS}_H$ for H is a 3-uple $[\Phi, H, \alpha]$, where $\Phi = \mathcal{A}_1 \cup \ldots \cup \mathcal{A}_n$ and α is obtained as follows. Let Q be a literal in Φ and let $(\varphi_1, \beta_1), \ldots, (\varphi_n, \beta_n)$ be all the weighted clauses in Φ with head Q, then*

$$f^{+}_{\Phi}(Q) = ACC(f^{MP}_{\Phi}(\varphi_1), \ldots, f^{MP}_{\Phi}(\varphi_n)).$$

Let (φ, β) be a weighted clause in Φ, then

$$f^{MP}_{\Phi}(\varphi) = \begin{cases} \beta & \text{if } \varphi \text{ is a fact } Q \\ \min(f^{+}_{\Phi}(P_1), \ldots, f^{+}_{\Phi}(P_n)) & \text{if } \varphi = Q \leftarrow P_1, \ldots, P_n. \end{cases}$$

And ACC is the one-complement accrual: $ACC(\alpha_1, \ldots, \alpha_n) = 1 - \prod_{i=1}^{n}(1 - \alpha_i)$.

Given $[\Phi, H, \alpha]$ and $[\Theta, K, \gamma]$, $[\Theta, K, \gamma]$ is an accrued substructure if $\Theta \subseteq \Phi$. Also $[\Theta, K, \gamma]$ is a complete accrued substructure of $[\Phi, H, \alpha]$ iff for any other accrued substructure $[\Theta', K, \gamma']$ of $[\Phi, H, \alpha]$ it holds that $\Theta' \subset \Theta$. We say $[\Psi, K, \beta]$ attacks $[\Phi, H, \alpha]$ at literal H' iff there is a complete accrued substructure $[\Phi', H', \alpha']$ of $[\Phi, H, \alpha]$ such that $K = \overline{H'}$ and $\beta > \alpha'$. [1]

Definition 2. *Let $\mathcal{P}$ be a possibilistic logic program. Let $Accruals(\mathcal{P})$ be the set of complete accrued structures of $\mathcal{P}$. Let $\mathcal{ASF}(\mathcal{P}) = (Accruals(\mathcal{P}), attacks)$ be the argumentation framework induced by the accruals of $\mathcal{P}$ where $attacks \subseteq Accruals(\mathcal{P}) \times Accruals(\mathcal{P})$. The extension of $\mathcal{P}$ is defined as the grounded extension of $\mathcal{ASF}(\mathcal{P})$ where attacks stands for the attack relation between complete accrued structures.*

Notice that the notions of both attack and valid conclusions of the system presented here differ from those of [3], thus leading to a different behavior.

Example 6. Consider again the ontology Σ_1 and its interpretation as the possibilistic program $\mathcal{P}_1$. The following complete accrued structures can be computed from $\mathcal{P}_1$: $\mathcal{AS}_1 = [\mathcal{A}_2, penguin(tweety), 1.0]$, $\mathcal{AS}_2 = [\mathcal{A}_7 \cup \mathcal{A}_8, flies(tweety), 0.96]$, $\mathcal{AS}_3 = [\mathcal{A}_3, pilot(tweety), 0.9]$, $\mathcal{AS}_4 = [\mathcal{A}_1, brokenWing(tweety), 0.8]$, $\mathcal{AS}_5 = [\mathcal{A}_5 \cup \mathcal{A}_6, \sim flies(tweety), 0.94]$ and $\mathcal{AS}_6 = [\mathcal{A}_4, bird(tweety), 1.0]$. We show $\mathcal{AS}_2$ and $\mathcal{AS}_5$ in Fig. 2. In this case $\mathcal{AS}_2$ attacks $\mathcal{AS}_5$ at the literal $flies(tweety)$ (see Fig. 3). Therefore the grounded extension of $\mathcal{ASF}(\mathcal{P}_1)$ is $\{\mathcal{AS}_1, \mathcal{AS}_2, \mathcal{AS}_3, \mathcal{AS}_4, \mathcal{AS}_6\}$.

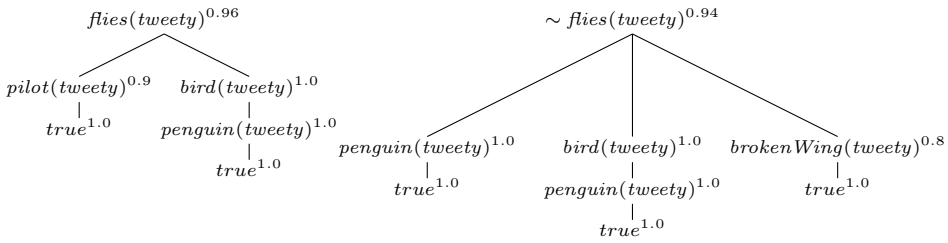

Fig. 2. Accrued structures $\mathcal{AS}_2$ and $\mathcal{AS}_5$ from $\mathcal{P}_1$ (Notice that the leftmost branch of $\mathcal{AS}_5$ stands for a sub-structure for $\sim flies(tweety)$ based on the argument $\mathcal{A}_5$ while the two rightmost branches for another based on the argument $\mathcal{A}_6$.)

Definition 3. *Given a possibilistic ontology Σ, a concept C and an individual a, we say that a is member of C with certainty degree α iff there is a complete accrued structure for $[\Phi, c(a), \alpha]$ in the grounded extension of $\mathcal{ASF}(\mathcal{T}(\Sigma))$.*

[1] $\overline{\cdot}$ is referred to as the complement operator where $\overline{P}$ is $\sim P$ and $\overline{\sim P}$ is P.

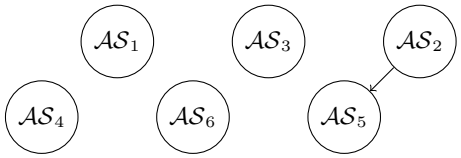

Fig. 3. Abstract argumentation framework $\mathcal{ASF}(\mathcal{P}_1)$ of Ex. 6

Example 7. Consider again the ontology Σ_1, $\mathcal{T}(\Sigma_1)$ is the program $\mathcal{P}_1$. The complete accrued structures from this program are those presented in Ex. 6. As $\mathcal{AS}_2 = [\mathcal{A}_7 \cup \mathcal{A}_8, \mathit{flies}(\mathit{tweety}), 0.96]$ belongs to the grounded extension of accrued structures of $\mathcal{P}_1$, then we conclude TWEETY is a member of Flies with certainty degree 0.96.

Property 1. When each accrued structure is formed by exactly one argument, the grounded extension of the argumentation framework induced by the program coincides with the argumentation framework induced from the accrued program.

Proof: Let Σ be a possibilistic ontology and $\mathcal{P}$ be $\mathcal{T}(\Sigma)$. Suppose that there is only one argument $\mathcal{A}$ in $\mathcal{P}$ with certainty degree α supporting H. Then the accrued structure $\mathcal{AS}$ for H is formed only by $\mathcal{A}$. Because of how $f^{+}_{\Phi}(H)$ is defined, the certainty degree of $\mathcal{AS}$ is also α. Then the graph of the argumentation framework formed by arguments is identical to the argumentation framework formed by accrued structures. Therefore their grounded extensions coincide.

Implementation details. In order to enact the approach proposed in this work, we developed a Java-based implementation that extends the one already presented in [10]. This application can be downloaded from the author's institutional site at `http://cs.uns.edu.ar/~sag/engine-v2/`. The implementation also provides the functionality of loading several examples, editing them, obtaining the equivalent P-DeLP program, computing grounded extensions and also provides a graphical representation of the argumentation framework, individual arguments and accrued structures based on the D3.JS and Dracula Javascript libraries. With this, we were able to replicate the accrued structures presented by [3]. Based on the experience gained by testing examples, we argue that this approach presents a more complex way to compute the vertexes of the argumentation framework but the resulting graph is smaller, thus leading to a much simpler reasoning framework.

5 Conclusions and Future Work

We have presented an approach for performing instance checking in inconsistent possibilistic description logic ontologies based on argument accrual, leading to more concise representations. Our approach involves translating ontologies into the language of possibilistic logic programming and grouping arguments for the

same conclusion via argument accrual. We have developed a Java-based implementation that allows the user to input an ontology and select reasoning either with or without argument accrual under a grounded semantics based on Dung-style argumentation. In our implementation, we only implemented direct attack between pairs of accrued structures; implementing the full approach of [3] in the context of Dung-based structured argumentation requires further research.

Acknowledgments: This research is funded by Secretaría General de Ciencia y Técnica, Universidad Nacional del Sur, Argentina.

References

1. Zhang, X., Xiao, G., Lin, Z., den Bussche, J.V.: Inconsistency-tolerant reasoning with OWL-DL. International Journal of Approximate Reasoning **55** (2014) 557–584
2. Bench-Capon, T.J.M., Dunne, P.E.: Argumentation in artificial intelligence. Artificial Intelligence **171**(10-15) (2007) 619–641
3. Lucero, M.G., Chesñevar, C.I., Simari, G.R.: Modelling Argument Accrual in Possibilistic Defeasible Logic Programming. In ad G. Chemello, C.S., ed.: ECSQARU 2009, LNAI 5590. (2009) 131–143
4. Verheij, B.: Rules, Reasons, Arguments: Formal studies of argumentation and defeat. PhD thesis, University of Maastricht (1996)
5. Letia, I., Groza, A.: Modelling Imprecise Arguments in Description Logics. Advances in Electrical and Computer Engineering **9**(3) (2009) 94–99
6. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook – Theory, Implementation and Applications. Cambridge University Press (2003)
7. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description Logic Programs: Combining Logic Programs with Description Logics. WWW2003, May 20-24, Budapest, Hungary (2003)
8. Benferhat, S., Bouraoui, Z., Lagrue, S., Rossit, J.: Merging Inconmensurable Possibilistic DL-Lite Assertional Bases. In Papini, O., Benferhat, S., Garcia, L., Mugnier, M.L., eds.: Proceedings of the IJCAI Workshop 13 Ontologies and Logic Programming for Query Answering. (2015) 90–95
9. Gómez, S.A., Chesñevar, C.I., Simari, G.R.: Using Possibilistic Defeasible Logic Programming for Reasoning with Inconsistent Ontologies. In Giusti, A.D., Diaz, J., eds.: Computer Science & Technology Series. XVII Argentine Congress of Computer Science Selected Papers. (2012) 19–29
10. Gómez, S.A.: Towards a practical implementation of a reasoner for inconsistent possibilistic description logic programming ontologies. In: Proceedings of the 2do. Simposio Argentino de Ontologías y sus Aplicaciones (SAOA 2016). SADIO (2016) (accepted)
11. Alsinet, T., Chesñevar, C.I., Godo, L.: A level-based approach to computing warranted arguments in possibilistic defeasible logic programming. In Besnard, P., Doutre, S., Hunter, A., eds.: COMMA. Volume 172 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 1–12
12. Dung, P.M.: On the aceptability of arguments and its fundamental role in nonmonotonic reasoning and logic programming. In: Proceedings of the 13th International Joint Conference in Artificial Intelligence (IJCAI). (1993) 852–857