An Environment for Possibilistic Logic Programming Based on Dung Semantics

Sergio Alejandro Gómez[1,2]

[1]Laboratorio de I+D en Ingeniería de Software y Sistemas de Información (LISSI)
Departamento de Ciencias e Ingeniería de la Computación,
Universidad Nacional del Sur
San Andrés 800, Bahía Blanca, Argentina
Email: sag@cs.uns.edu.ar
[2]Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, La Plata, Argentina

Abstract. In this paper, we present a tool for possibilistic logic programming. [1] This tool is a desktop-based, stand-alone application that assists a user in creating, editing and querying a possibly inconsistent possibilistic program. The tool computes all the arguments emerging from the program, the grounded extension based on Dung-style semantics, and it is capable of showing arguments and grounded extensions graphically. The language for programs is enriched with pragmas for allowing the user to configure labels for necessity degrees, deciding if using transposes of strict rules, performing consistency checks within arguments, and appeal to the use of accrual of rules for building arguments. We describe its usage, architectural elements and we also provide experimental evaluation of its performance.

Keywords: Possibilistic logic programming · argumentation semantics · knowledge representation · knowledge-based systems · Artificial Intelligence

1 Introduction

Computational Argumentation has been a relevant research area within Artificial Intelligence during at least the last twenty years [3, 7, 15]. Argumentation allows for deciding between contradictory claims known as arguments. In abstract argumentation, the interior of arguments are inaccessible while in structured argumentation, the structure of arguments is accessible and usually made by the derivation of a conclusion from facts or presumptions using defeasible rules. Two arguments in disagreement attack each other but the stronger argument is said to defeat the weaker one. If the attack is made into the final conclusion

[1] A demo of the tool can be watched online at `https://youtu.be/joajw2hfkZg`. For reproducing the results reported here, an executable JAR file along with the examples shown here are included in the Github repository `sergio-alejandro-gomez/argumentative-engine`.

of the argument, then the attack is said to be direct but if the argument is attacked in one of its assumptions, the attack is considered indirect. In abstract argumentation, argument strength is defined by a binary relation. Instead, in structured argumentation, the criteria for deciding argument strength require analyzing the interior of arguments and include generalized specificity, weakest or last link, among others. Once the attack relationship is defined and the attacks between pair of arguments are computed, an argumentation framework resembling a directed graph is induced. From this directed graph, the truths (i.e. the accepted arguments) of the system have to be discovered. The set of accepted arguments is called an extension. There are several kinds of extensions considering that opposing conclusions can exist (e.g. allowing for credulous or brave reasoning) or only consistency is required (typically in skeptical contexts that are sometimes required). Another desirable trait of reasoning in argumentation is the possibility of modeling accrual of arguments where given different arguments supporting the same conclusion, their accrual allows to accumulate their strength [14].

Implementation of argumentation systems is an important research topic (see [1, 4, 6, 13] and references therein). Applications of these implementations range from desktop and web-based systems for both structured and abstract argumentation systems, tools for teaching argumentation in education institutions and decision-making in knowledge-based systems. Several principles of implementation have arisen during the last years that include: (i) a digraph of arguments for abstract argumentation where vertices are arguments and edges model an opposition relation between arguments that arises from the interaction of structured arguments; (ii) implementations have to support a certain level of generality allowing for the computation of several argumentation semantics, and (iii) implementations have to provide a sufficient level of efficiency to deal with the high intrinsic complexity of the problems at hand [6].

In this paper, we report on the advances we have made on the implementation of a Java-based system that uses structured arguments to represent claims based on the language of possibilistic logic programming [2]. Our language allows to represent knowledge as rules with their respective weight (a.k.a. necessity degrees) noted as real numbers between 0 and 1, pragmas for defining labels (symbolic constants for representing rule weights), and for selecting if transposes of strict rules, and accrual of arguments should be used when computing the grounded extension of the argumentation graph induced from the program, thus customizing the user experience. In its current implementation status, our system includes computing arguments based on weakest-link, ability to use transposes of rules, accrual of arguments and grounded semantics. It also allows for the visual presentation of arguments and argumentation graphs, and an IDE for programming. We perform an experimental evaluation showing the behavior of our tool in the presence of increasing demands. We think that the results presented in this work can be useful to developers of knowledge-based systems and to teachers of symbolic artificial intelligence, particularly of argumentation systems.

The rest of this paper is structured as follows. In Sect. 2, we briefly explain reasoning in argumentation frameworks with structured arguments based on possibilistic defeasible logic programming. In Sect. 3, we describe the status of our prototype implementation for the reasoning framework presented discussing its current functionalities, the possible customizations allowed to the user and an empirical evaluation of its behavior with both arguments and linear argumentation lines of increasing size. In Sect. 4, we review related work. Finally, in Sect. 5, we conclude and discuss perspectives of future work.

2 Dung-Style Structured Possibilistic Argumentation with Accrual

Here, we briefly describe the logical formalism underlying the implementation of our system using a running example that relates the theory with its practical application.

2.1 Dung-Style Abstract Argumentation

Abstract argumentation frameworks do not presuppose any internal structure of arguments, thus considering only the interactions of arguments by means of an attack relation between arguments [18]. 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 representing the attack between arguments. In this work, we will consider only finitary argumentation systems (i.e. argument systems with a finite number of arguments). Abstract argumentation frameworks can be concisely represented by directed graphs, where arguments are represented as nodes and edges model the attack relation.

Example 1. Consider the argumentation framework $\mathcal{AF}_1 = (Arg, \rightarrow)$ where $Arg = \{\mathcal{A}_6, \mathcal{A}_{16}, \mathcal{A}_{19}, \mathcal{A}_{20}\}$ and $\rightarrow = \{(\mathcal{A}_{16}, \mathcal{A}_{19}), (\mathcal{A}_{16}, \mathcal{A}_{20}), (\mathcal{A}_{21}, \mathcal{A}_{16}), (\mathcal{A}_{21}, \mathcal{A}_6)\}$. The framework is shown graphically in Fig. 1.

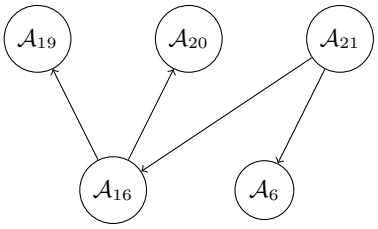

Fig. 1. Abstract argumentation framework presented in Ex. 1

Semantics are usually given to abstract argumentation frameworks by means of extensions that are subsets that give some coherent view on the underlying

argumentation framework. In this work, we will only reason under grounded semantics. Let $\mathcal{AF} = (Arg, \rightarrow)$ be an argumentation framework. An extension $E \subseteq Arg$ is *conflict-free* iff there are no $\mathcal{A}, \mathcal{B} \in E$ 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 an $\mathcal{A}' \in E$ 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 underlying the completeness property is that all acceptable arguments should be accepted. The grounded extension is the minimal set of acceptable arguments and is 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 are no unattacked arguments, the remaining arguments are also removed.

Example 2. Consider again the argumentation framework $\mathcal{AF}_1$ presented in Ex. 1. The grounded extension E of $\mathcal{AF}_1$ includes the arguments $\mathcal{A}_{19}, \mathcal{A}_{20}, \mathcal{A}_{21}$. Therefore these arguments are labeled as IN while the arguments $\mathcal{A}_6$ and $\mathcal{A}_{16}$ are labeled as OUT.

Our application abide to this approach but considering structured arguments based on the P-DeLP language, as introduced next.

2.2 Possibilistic Defeasible Logic Programming (P-DeLP)

The P-DeLP [2] language $\mathcal{L}$ is defined from a set of ground fuzzy atoms (fuzzy propositional variables) $\{p, q, \ldots\}$ together with the connectives $\{\sim, \wedge, \leftarrow\}$. The symbol $\sim$ stands for *negation*. 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, written $\vdash$, is defined based on the generalized modus ponens rule, that from $(L_0 \leftarrow L_1 \wedge \ldots \wedge L_k, \gamma)$ and $(L_1, \beta_1), \ldots, (L_k, \beta_k)$ allows to infer $(L_0, \min(\gamma, \beta_1, \ldots, \beta_k))$, which is a particular instance of the possibilistic resolution rule, and which provides the *non-fuzzy* fragment of P-DeLP with a complete calculus for determining the maximum degree of possibilistic entailment for weighted literals.

In P-DeLP *certain* and *uncertain* clauses can be distinguished. A clause (φ, α) is referred as certain if $\alpha = 1$ 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 in $\mathcal{L}$. A P-DeLP 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. Given a program $\mathcal{P} = (\Pi, \Delta)$, a set $\mathcal{A} \subseteq \Delta$ of uncertain clauses is an *argument* for a goal Q with necessity degree $\alpha > 0$, denoted $\langle A, Q, \alpha \rangle$, iff: (i) $\Pi \cup \mathcal{A} \vdash (Q, \alpha)$; (ii) $\Pi \cup \mathcal{A}$ is non-contradictory, and (iii) there is no $\mathcal{A}_1 \subset \mathcal{A}$ such that $\Pi \cup \mathcal{A}_1 \vdash (Q, \beta)$, $\beta > 0$. Let $\langle \mathcal{A}, Q, \alpha \rangle$ and $\langle \mathcal{S}, R, \beta \rangle$ be two arguments, $\langle \mathcal{S}, R, \beta \rangle$ is a *subargument* of $\langle \mathcal{A}, Q, \alpha \rangle$ iff $\mathcal{S} \subseteq \mathcal{A}$.

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 will be defined on the basis of 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 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 3. Consider the possibilistic program $\mathcal{P}_3$ in Fig. 2 that describes the criteria for deciding whether performing or not a surgical procedure depending on several factors such as severity, availability of a doctor, affordability, etc. Exactly 21 arguments can be built from this knowledge base. We only show the ones that coincide with those presented in Ex. 1 (for clarity, clauses are separated by semicolons):

- $\langle \mathcal{A}_6, \sim canAfford(one\_million), 0.9 \rangle$ where $\mathcal{A}_6 = \{ (\sim canAfford(one\_million), 0.9) \}$
- $\langle \mathcal{A}_{16}, \sim performSurgery(heartDisease), 0.9 \rangle$ where

$$\mathcal{A}_{16} = \left\{ \begin{array}{l} (\sim performSurgery(heartDisease) \leftarrow acuteDisease(heartDisease), \\ performs(john, heartDisease), doctor(john), fee(john, one\_million), \\ \sim canAfford(one\_million), 1.0); \\ (acuteDisease(heartDisease) \leftarrow disease(heartDisease), \\ urgent(heartDisease), 1.0); \\ (disease(heartDisease), 1.0); (urgent(heartDisease), 1.0); \\ (performs(john, heartDisease), 1.0); (doctor(john), 1.0); \\ (fee(john, one\_million), 1.0); (\sim canAfford(one\_million), 0.9) \end{array} \right\}$$

- $\langle \mathcal{A}_{19}, performSurgery(heartDisease), 0.7 \rangle$ where

$$\mathcal{A}_{19} = \left\{ \begin{array}{l} (performSurgery(heartDisease) \leftarrow acuteDisease(heartDisease), 0.7); \\ (acuteDisease(heartDisease) \leftarrow disease(heartDisease), \\ urgent(heartDisease), 1.0); \\ (disease(heartDisease), 1.0); (urgent(heartDisease), 1.0) \end{array} \right\}$$

– $\langle \mathcal{A}_{20}, performSurgery(heartDisease), 0.8 \rangle$ where

$$\mathcal{A}_{20} = \left\{ \begin{array}{l} (performSurgery(heartDisease) \leftarrow acuteDisease(heartDisease), \\ performs(john, heartDisease), expertDoctor(john), 0.8); \\ (acuteDisease(heartDisease) \leftarrow disease(heartDisease), \\ urgent(heartDisease), 1.0); (disease(heartDisease), 1.0); \\ (urgent(heartDisease), 1.0); (performs(john, heartDisease), 1.0); \\ (expertDoctor(john) \leftarrow doctor(john), expert(john), 1.0); \\ (doctor(john), 1.0); (expert(john), 0.8) \end{array} \right\}$$

– $\langle \mathcal{A}_{21}, canAfford(one\_million), 1.0 \rangle$ where

$$\mathcal{A}_{21} = \left\{ \begin{array}{l} (canAfford(one\_million) \leftarrow loanFromBank(one\_million, city\_bank), 1.0); \\ (loanFromBank(one\_million, city\_bank) \leftarrow bank(city\_bank), \\ loan(one\_million, city\_bank), 1.0); \\ (bank(city\_bank), 1.0); (loan(one\_million, city\_bank), 1.0) \end{array} \right\}$$

The attacks among these arguments are exactly those presented in Fig. 1. Notice that the traditional approach to PDeLP, facts are not considered part of arguments as they are not meant to be attacked. We include them in arguments as a notation abuse. Also 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 for modeling attacks into premises (i.e. indirect attacks).

Rules:

$$\begin{array}{rl} (\sim performSurgery(D) & \leftarrow disease(D), \sim urgent(D), 0.6) \\ (performSurgery(D) & \leftarrow acuteDisease(D), 0.7) \\ (acuteDisease(D) & \leftarrow disease(D), urgent(D), 1.0) \\ (\sim performSurgery(D) & \leftarrow acuteDisease(D), performs(M, D), doctor(M), \\ & \quad fee(M, S), \sim canAfford(S), 1.0) \\ (performSurgery(D) & \leftarrow acuteDisease(D), performs(M, D), expertDoctor(M), 0.8) \\ (expertDoctor(M) & \leftarrow doctor(M), expert(M), 1.0) \\ (canAfford(L) & \leftarrow loanFromBank(L, B), 1.0) \\ (loanFromBank(L, B) & \leftarrow bank(B), loan(L, B), 1.0) \end{array}$$

Facts and presumptions:

$$\begin{array}{rl} (disease(heartDisease), 1.0) & (urgent(heartDisease), 1.0) \\ (doctor(john), 1.0) & (expert(john), 0.8) \\ (fee(john, one\_million), 1.0) & (\sim canAfford(one\_million), 0.9) \\ (performs(john, heartDisease), 1.0) & (disease(mole), 1.0) \\ (doctor(peter), 1.0) & (\sim urgent(mole), 0.8) \\ (performs(peter, mole), 1.0) & (bank(city\_bank), 1.0) \\ (loan(one\_million, city\_bank), 1.0) & \end{array}$$

Fig. 2. Possibilistic defeasible logic program for deciding a surgery

2.3 Accrual of Arguments

Now we deal with the problem of accruing arguments. Our approach relies in previous work of Gómez Lucero et al. [11] adapting their approach. Gómez Lucero

et al. 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 [11]:

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, whenever φ is a fact Q then $f^{MP}_{\Phi}(\varphi) = \beta$ but if $\varphi = Q \leftarrow P_1, \ldots, P_n$ then $f^{MP}_{\Phi}(\varphi) = \min(f^{+}_{\Phi}(P_1), \ldots, f^{+}_{\Phi}(P_n))$. ACC stands for 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 $\beta > \alpha'$ ($\bar{\cdot}$ stands for the complement operator where $\overline{P}$ is $\sim P$ and $\overline{\sim P}$ is P).

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 [11], thus leading to a different behavior.

Example 4. Consider again the possibilistic program $\mathcal{P}_3$. From this program, a total of 20 accrued structures can be built. We show the more relevant ones according to Ex. 3. The following complete accrued structures can be computed from $\mathcal{P}_3$: $\mathcal{AS}_2 = [\mathcal{A}_6, \sim canAfford(one\_million), 0.9]$, $\mathcal{AS}_5 = [\mathcal{A}_{16}, \sim performSurgery(heartDisease), 0.9]$, $\mathcal{AS}_9 = [\mathcal{A}_{21}, canAfford(one\_million), 1.0]$ and $\mathcal{AS}_{15} = [\mathcal{A}_{19} \cup \mathcal{A}_{20}, performSurgery(heartDisease), 0.94]$ where arguments $\mathcal{A}_6$, $\mathcal{A}_{16}$, $\mathcal{A}_{19}$, $\mathcal{A}_{20}$ and $\mathcal{A}_{21}$ were already introduced in Ex. 3. We show $\mathcal{AS}_{15}$ in Fig. 3 in abbreviated form for space reasons. Notice that the two leftmost branches of $\mathcal{AS}_5$ stands for a sub-structure for $\mathcal{A}_{19}$ and two rightmost stands for $\mathcal{A}_{20}$. The relevant part of the argumentation framework with accrual is presented in Fig. 4. In this case, $\mathcal{AS}_{15}$ attacks $\mathcal{AS}_5$ and $\mathcal{AS}_9$ attacks $\mathcal{AS}_2$. Accordingly, $\mathcal{AS}_9$ and $\mathcal{AS}_{15}$ are included in the grounded extension of $\mathcal{ASF}(\mathcal{P}_3)$ but $\mathcal{AS}_2$ and $\mathcal{AS}_5$ are excluded.

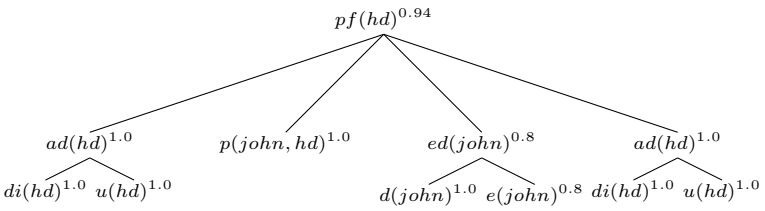

Fig. 3. Accrued structure $\mathcal{AS}_{15}$ for supporting the decision of performing the surgery for heart disease. Nomenclature: *d* stands for *doctor*, *u* stands for *urgent*, *di* stands for *disease*, *hd* stands for *heartDisease*, *e* stands for *expert*, *ed* stands for *expertDoctor*, *ad* stands for *acuteDisease*, *ps* stands for *performSurgery*, and *p* stands for *performs*.

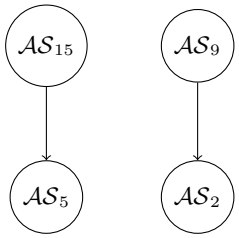

Fig. 4. Abstract argumentation framework $\mathcal{ASF}(\mathcal{P}_3)$ of Ex. 4

3 A Prototype Implementation for an Argumentative Reasoner Using Dung-Style Argumentation

Here we explain the Java-based implementation we developed to enact the reasoning framework presented above.

3.1 Computing Grounded Extensions for Possibilistic Logic Programs

As shown above, computing the accepted arguments emerging from a program is a complex process. We review the algorithmic details of the approach used in our implementation here.
Step 1: Propositionalization of the possibilistic program. Given a program, all the transposes of strict rules are composed (when indicated by the user) and then all the possible propositional enumerations of the rules are generated.
Step 2: Computation of arguments. The set of arguments that can be derived from the propositional program is built inductively from facts and presumptions considering the rules of the program as derivation rules until a fix-point is reached. Checking for internal consistency in argument construction is optional due to its exponential complexity in the size of the argument.
Step 3: Computing attacks between arguments. For creating the argumentation framework, it requires to create a directed graph whose vertices are arguments and its directed edges represent attacks between pairs of arguments

(see Sect. 2.1). Discovering the edges requires iterating over every pair of arguments. Whenever the conclusion of an argument contradicts the conclusion of another argument or some subargument and its weight is greater or equal than the weight of the other argument, then an attack has been found and the subargument is the point of attack.
Step 4: Computing the grounded extension. The algorithm based on the topological sort explained in Sect. 2.1 is run on the directed graph. This requires finding all the roots of the graph (i.e. vertices with null incidence degree), printing these vertexes, deleting them and its successors edges, and then repeating the process until all of the vertices have been processed.

In Fig. 5, we show a visual presentation of an argumentation framework for Ex. 1 and 3 as produced by the application. Arguments, attacks and points of attack are shown.

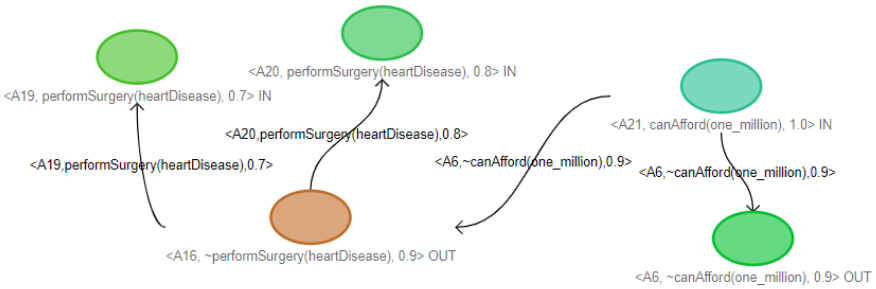

Fig. 5. Visual presentation of an argumentation framework for Ex. 3

3.2 Elements for Personalizing the User Experience

Our approach to provide a text representation for possibilistic programs follows the path marked by DeLP and ASPIC. Facts of the form $(p(a), \alpha)$ are represented as "`p(a) <- true` α" and rules of the form $(p(X) \leftarrow q_1(X), \ldots, q_n(X), \alpha)$ are coded as "`p(X) <- q1(X), ..., qn(X)` α". Besides, the strong negation of `p(X)` is represented with `~p(X)`. In Fig. 6, we present the PDeLP-like script for the program of Ex. 3.

This iteration of our system, besides introducing an Integrated Development Environment in the form of an editor, both computation and visualization of arguments, accrued structures and grounded extensions, and querying the status of arguments and accrued structures, also introduces the capacity of customizing the source code by using pragmas for specifying how the code should be interpreted and to define symbolic constants for the necessity degrees in rules (see Fig. 7).

The pragmas included and the extended syntax in this version of the application are:

```
~performSurgery(D) <- disease(D), ~urgent(D) 0.6.
performSurgery(D) <- acuteDisease(D) 0.7.
acuteDisease(D) <- disease(D), urgent(D) 1.0.
~performSurgery(D) <- acuteDisease(D), performs(M,D), doctor(M),
       fee(M,S), ~canAfford(S) 1.0.
performSurgery(D) <- acuteDisease(D), performs(M,D), expertDoctor(M) 0.8.
expertDoctor(M) <- doctor(M), expert(M) 1.0.
disease(heartDisease) <- true 1.0.          urgent(heartDisease) <- true 1.0.
doctor(john) <- true 1.0.          expert(john) <- true 0.8.
fee(john, one_million) <- true 1.0.
~canAfford(one_million) <- true 0.9.
performs(john, heartDisease) <- true 1.0.
disease(mole) <- true 1.0.          doctor(peter) <- true 1.0.
~urgent(mole) <- true 0.8.          performs(peter, mole) <- true 1.0.
canAfford(L) <- loanFromBank(L, B) 1.0.
loanFromBank(L, B) <- bank(B), loan(L, B) 1.0.
bank(city_bank) <- true 1.0.          loan(one_million, city_bank) <- true 1.0.
```

Fig. 6. Script for the possibilistic defeasible logic program for deciding a surgery

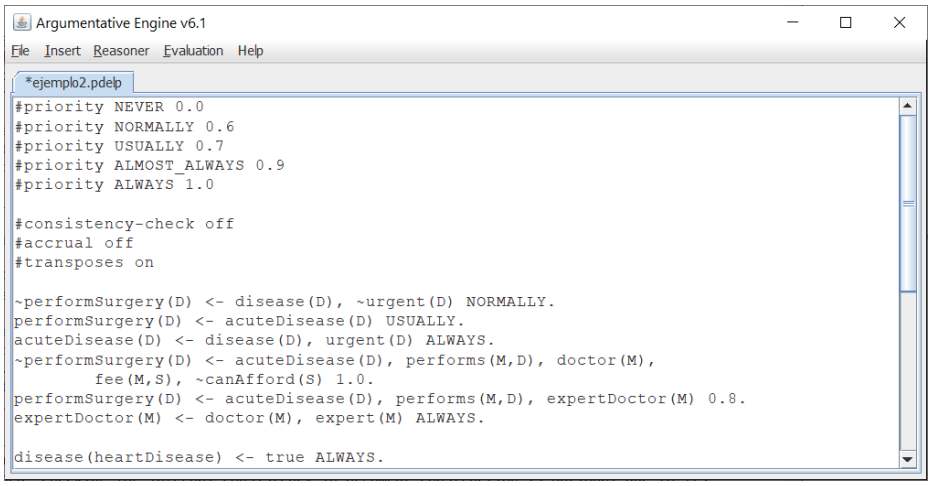

Fig. 7. User interface of desktop IDE of the engine for reasoning with possibly inconsistent possibilistic logic programs

```
#priority NEVER 0.0
#priority SELDOM 0.1
#priority RARELY 0.2
#priority SOMETIMES 0.5
#priority NORMALLY 0.6
#priority USUALLY 0.7
#priority ALMOST_ALWAYS 0.9
#priority ALWAYS 1.0

#consistency-check off
#accrual off
#transposes off

~performSurger(D) <- disease(D), ~urgent(D) NORMALLY.
performSurgery(D) <- acuteDisease(D) USUALLY.
...
```

See how labels for necessity degrees are defined. Rules with degree equal to NEVER are not included in computations. Compiler directives for consistency check, use of accrual of arguments and computation of transposes of strict rules are shown along with its application in rules, facts and presumptions. Its usage can be seen in the screen capture provided in Fig. 7.

3.3 Empirical Evaluation

We now discuss some of the tests we have performed in order to test how our application handles increasing demands in knowledge base size. The performance of our system is affected mainly by (i) the fact that knowledge bases with ground clauses are materialized in a bottom-up fashion (potentially building unusable clauses due to the brute-force approach involved), and (ii) the system is implemented in the JAVA programming language. Our tests were conducted on an ASUS notebook having an Intel Core i7, 3.5GHz CPU, 8GB RAM, 1TB HDD, and Windows 10. They involved the creation of programs (i) containing a single argument of increasing size and (ii) containing a single argumentation line composed of arguments formed by exactly one rule and one fact and that rule attacks the previous argument in the line.

Building linear arguments. In the test case we devised, an argument of size n can be derived from the program (which is built automatically by the test suite): $(p_1(X) \leftarrow p_2(X), 1.0)$, $(p_2(X) \leftarrow p_3(X), 1.0)$, $\ldots$, $(p_{n-1}(X) \leftarrow p_n(X), 1.0)$, $(p_n(a), 1.0)$. We can see that this argument has $n-1$ subarguments that will have relevance when building the argument base and each one will be a vertex of the argumentation framework, thus increasing the execution time of the algorithm for computing the grounded extension of the argumentation framework. In Table 1, we show the results of the tests that we performed. We can see that the performance of the algorithm for computing the extension of the argumentation framework is reasonable but deriving the propositional program from a (simple) program takes considerable time.

Building linear argumentation lines. In the test case we devised for argumentation lines of size n, the test suite uses a pattern for building programs like

Table 1. Running times for processing linear arguments

Number of rules	Source file size [Kilobytes]	Time for loading program [seconds]	Time elapsed computing extension [seconds]
10	0.22	0.066	0.004
20	0.43	0.120	0.007
30	0.64	0.227	0.008
100	2.15	1.237	0.020
1,000	23.20	107.354	5.751

the following. Let $\epsilon = 0.0001$ then, the test program will be composed as: $(p_1(X) \leftarrow\sim p_2(X), 0.1)$, $(\sim p2(a), 0.1)$, $(p_2(X) \leftarrow\sim p_3(X), 0.1+\epsilon)$, $(\sim p_3(a), 0.1+\epsilon)$, $(p_3(X) \leftarrow\sim p_4(X), 0.1+2\epsilon)$, $(\sim p_4(a), 0.1+2\epsilon), \ldots, (p_{n-1}(X) \leftarrow\sim p_n(X), 0.1+n\epsilon)$, $\sim p6(a), 0.1+n\epsilon)$, $(p_n(a), 0.1+(n+1)\epsilon)$. Thus, each new argument defeats the previous one in the argumentation line. In Table 2, we show the results that we obtained when we tested our program. The results show that the program can handle argumentation lines of size 10,000 but the generated graph cannot be understood when it has a size of 1,000 nodes and that the MS Edge browser is not able to display a graph of 10,000 nodes.

Table 2. Running times for processing linear argumentation lines

size	time loading program [seconds]	time elapsed computing extension [seconds]	Could draw the argumentation framework?	size of the web page for AF [Kilobytes]	size of source code [Kilobytes]
10	0.12	0.007	yes	3.308	0.58
20	0.172	0.011	yes	6.400	1.17
30	0.241	0.017	yes	9.495	1.77
100	0.778	0.058	yes	31.650	5.98
1,000	9.802	1.043	yes (with lag, totally unintelligible)	327.000	62.06
10,000	728.843	99.497	no (Edge did not respond in 10')	3,378.000	641.00

4 Related Work

Classical reviews in argumentation systems are [3, 6, 7, 15]; we refer the reader for details of different theoretical approaches in the field. Gómez [9] reviews a comparison of an earlier version of this engine with the following related works [5, 16, 17] that also implement argumentation systems.

More recent implementations of argumentation systems include the following works. DeLP [8] uses a language similar to the one presented here but without the neccesity degrees. DeLP does not need necessity degrees for argument comparison because it uses generalized specificity. DeLP does not use transposes of strict rules. Presumptions in DeLP have to be handled by a special formalization known as PreDeLP [12].

DAQAP [10] is a Web platform for Defeasible Argumentation Query Answering, which offers a visual interface that facilitates the analysis of the argumentative process defined in the Defeasible Logic Programming (DeLP) formalism. The tool presents graphs that show the interaction of the arguments generated from a DeLP program; this is done in two different ways: the first focuses on the structures obtained from the DeLP program, while the second presents the defeat relationships from the point of view of abstract argumentation frameworks, with the possibility of calculating the extensions using Dung's semantics. Using all this data, the platform provides support for answering queries regarding the states of literals of the input program. Our application has a similar functionality except that is desktop based and the language used is that of possibilistic DeLP.

The *TweetyProject* (`http://tweetyproject.org/`) provides a comprehensive collection of Java libraries for logical aspects of artificial intelligence and knowledge representation. Defeasible Logic Programming, Our system has the potential ability of being presented as a Java library once the classes that implement the model of the application are selected.

5 Conclusions and Perspective

We have presented a report of the current state of our application that allows editing possibilistic logic programs. The system computes the arguments, the accrued structures and the grounded extension of the argumentation framework induced from the program. The presented desktop application includes a language that allows the user to customize the description of the knowledge base. Performance tests show that the application is able to successfully handle lines of argument on the order of 10,000 arguments, but its display in the user interface is not satisfactory. As a future perspective, the construction of an API from the source code is proposed to be able to use the functionality from other applications, the export of results to Argument Interchange Format and the use of visualization techniques of large volumes of data to show the results computed by the application. It is also of interest to include the computation of other argument acceptance semantics and provide a command line tool.

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

References

1. Derevyanko A., Morosin O., and Vagin V. Implementation of the argumentation system based on defeasible reasoning theory. In Abraham A., Kovalev S., Tarassov V., and Snášel V., editors, *Proceedings of the First International Scientific Conference "Intelligent Information Technologies for Industry" (IITI'16). Advances in Intelligent Systems and Computing*, volume 450. Springer, Cham, 2016.

2. Teresa Alsinet, Carlos Iván Chesñevar, and Lluis Godo. A level-based approach to computing warranted arguments in possibilistic defeasible logic programming. In Philippe Besnard, Sylvie Doutre, and Anthony Hunter, editors, *COMMA*, volume 172 of *Frontiers in Artificial Intelligence and Applications*, pages 1–12. IOS Press, 2008.
3. Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artificial Intelligence*, 171(10-15):619–641, 2007.
4. Stefano Bistarelli, Fabio Rossi, and Francesco Santini. ConArgLib: An Argumentation Library with Support to Search Strategies and Parallel Search. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–28, 2020.
5. Daniel Bryant, Paul John Krause, and Gerard Vreeswijk. Argue tuProlog: A Lightweight Argumentation Engine for Agent Applications. In *Computational Models of Argument: Proceedings of COMMA 2006*. 2006.
6. Gunther Charwat, Wolfgang Dvořák, Sarah Gaggl, Johannes Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation – a survey. *Artificial Intelligence*, 220:28–63, 2015.
7. Carlos Iván Chesñevar, Ana Maguitman, and Ronald Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, December 2000.
8. Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: Delpservers, contextual queries, and explanations for answers. *Argument & Computation*, 5(1):63–88, feb 2014.
9. Sergio Alejandro Gómez. Reasoning with inconsistent possibilistic ontologies by applying argument accrual. *Journal of Computer Science and Technology*, 17(2):117–126, 10 2017.
10. Mario A. Leiva, Gerardo I. Simari, Sebastian Gottifredi, Alejandro J. Garcia, and Guillermo Simari. DAQAP: Defeasible argumentation query answering platform. In *Proceedings of the 13th International Confernce on Flexible Query Answering Systems (FQAS 2019), LNCS*, volume 11529, pages 126—138. Springer-Verlag, 2019.
11. Mauro Javier Gómez Lucero, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. Modelling argument accrual with possibilistic uncertainty in a logic programming setting. *Inf. Sci.*, 228:1–25, 2013.
12. Maria Vanina Martinez, Alejandro Javier Garcia, and Guillermo Ricardo Simari. On the use of presumptions in structured defeasible reasoning. *Frontiers in Artificial Intelligence and Applications*, 245(1):185–196, January 2012.
13. Mikolaj Podlaszewski, Martin Caminada, and Gabriella Pigozzi. An implementation of basic argumentation components. volume 2, pages 1307–1308, 01 2011.
14. Henry Prakken. Modelling accrual of arguments in ASPIC+. In *ICAIL '19: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Law*, pages 103—-112, june 2019.
15. Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence*. Springer, 2009.
16. Mark Snaith and Chris Reed. TOAST: online ASPIC+ implementation. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*. IOS Press, 2012.
17. Nouredine Tamani and Madalina Croitoru. Fuzzy argumentation system for decision support. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, volume 442, pages 77–86. Communications in Computer and Information Science, 2014.
18. Matthias Thimm. Strategic Argumentation in Multi-Agent Systems. *Künstliche Intelligenz*, 28(3):159–168, August 2014.