Inteligencia Artificial 17(53) (2014), 68-78

INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

An Approach to Argumentative Reasoning Servers with Multiple Preference Criteria

Juan Carlos Teze[1,2,3], Sebastian Gottifredi[1,3], Alejandro J. Garcia[1,3] and Guillermo R. Simari[1]

[1]Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur - Alem 1253
(8000) Bahía Blanca, Buenos Aires, Argentina

[2]Agents and Intelligent Systems Area, Fac. of Management Sciences,
Universidad Nacional de Entre Ríos
(3200) Concordia, Entre Ríos, Argentina

[3]Consejo Nacional de Investigaciones Científicas y Técnicas
E-mail: {jct,sg,ajg,grs}@cs.uns.edu.ar

Abstract Argumentation is an attractive reasoning mechanism due to its dialectical and non monotonic nature, and its properties of computational tractability. In dynamic domains where the agents deal with incomplete and contradictory information, to determine the accepted or warranted information, an argument comparison criterion must be used. Argumentation systems that use a single argument comparison criterion have been widely studied in the literature. In some of these approaches, the comparison is fixed and in others the criterion can be replaced in a modular way. In this work we introduce an argumentative server that provides recommendations to its client agents and the ability to decide how multiple argument comparison criteria can be combined. In the proposed formalism, the argumentative reasoning is based on the criteria selected by the client agents. As a result, a set of operators to combine multiple preference criteria is presented.

Resumen Argumentación es un atractivo mecanismo de razonamiento debido a su naturaleza dialéctica y no monótona, y sus propiedades de tratabilidad computacional. En dominios dinámicos donde los agentes tratan con información incompleta y contradictoria, un criterio de comparación de argumento debe ser utilizado para determinar la información aceptada o garantizada. Los sistemas de argumentación que utilizan un único criterio de comparación de argumento han sido ampliamente estudiados en la literatura. En alguno de estos enfoques, la comparación es fija y en otros el criterio puede ser reemplazado de forma modular. En este trabajo introducimos un servidor argumentativo que provee recomendaciones a sus agentes clientes y la habilidad para que estos puedan decidir como múltiples criterios de comparación pueden ser combinados. En el formalismo propuesto, el razonamiento argumentativo esta basado en los criterios seleccionados por los agentes clientes. Como resultado presentamos un conjunto de operadores para combinar múltiples criterios de preferencia.

Keywords: Reasoning Server, Argumentation System, Preference Criteria, Criteria Combination.

Palabras Clave: Servidor de Razonamiento, Sistema de Argumentación, Criterios de Preferencia, Combinación de Criterios.

ISSN: 1988-3064(on-line)

1 Introduction

An essential characteristic of Multi-Agent Systems (MAS) is the modeling of the interaction among agents. Agents in a MAS interact to perform tasks, that can be collectively carried out by a set of agents or can be individually done by one agent. Generally, deliberative agents reason using two types of knowledge: public knowledge that is shared with other agents, and private knowledge that in part come from their own perception of the environment; in [9] a client-server model was proposed allowing the representation of both private and shared knowledge.

A defeasible argumentation system provides ways to confront contradictory statements to determine whether some particular information can be accepted or, using a technical term, warranted [10, 1, 6, 7, 17]. The result of the argumentation process leads to an answer involving many stages; the comparison of conflicting arguments to decide which one prevails is a particularly important one. For this reason, the definition of a formal criterion for comparing arguments becomes a central problem in defeasible argumentation.

Argumentation systems using a single argument comparison criterion have been widely studied in the literature [18, 19, 5, 10]. The argument comparison criterion represents a fundamental part of an argumentation system because the inferences an agent can obtain from its knowledge will depend on the criterion that is used. In the literature of argumentative systems, several approaches use a fixed comparison criterion embedded in the system and in others the criterion can be replaced in a modular way. In [2, 12, 8], the authors also focused their works in multiple criteria, however, in a different manner to the way is proposed in this paper. The main contribution of this paper is to provide a framework where several comparison criteria can be selected and combined for deciding which argument prevails. Next, we show an example that will serve two purposes: to motivate the main ideas of our proposal, and as a running example to be used in the rest of the paper.

Example 1 *Lets consider a hotel that has the following general information about its clients:*

- *if a client travels alone and she does not want a jacuzzi included with the room, the hotel does not recommend a junior suite,*
- *if a client does not want a jacuzzi but she wants a safe and a fridge included with the room, the hotel recommends a junior suite,*
- *if a client travels alone and she wants a safe and a fridge included with the room, the hotel recommends a junior suite,*
- *if a client travels alone, the hotel recommends a junior suite,*
- *if a client plans a short stay, the hotel recommends a single room,*
- *if a client is tired, the hotel recommends room with jacuzzi included,*
- *finally, if the hotel recommends a single room, then it does not recommend a junior suite.*

Now, two clients (Joan and Michael) request a room to the hotel. Suppose that for both clients provide the same information:

- *each travels alone.*
- *each plans for a short stay trip.*
- *each wants a room with safe and fridge.*

They also share the same criteria for selecting a room, i.e., comfort and price; however, Joan and Michael combine them differently. For Joan the room must satisfy both criteria of the comfort and price. But, Michael is more tolerant and he expects that the room will meet at least one criterion, either comfort or price. In this situation each client can receive contradictory recommendations. This shows how the client's criteria to select a room can be used to establish which recommendation prevails. Since, each client combine the same criteria on different ways, the results are different.

In [3], a framework to reason from multiple points of view on an inconsistent knowledge base was proposed. In that formalism each preference is associated with a context, and these contexts are totally ordered; consequently, the relation among each preference is fixed by this ordering. In contrast to [3], our approach does not depend on a fixed pre-ordering between preferences. Here we generalize the way several preferences can relate to each other through a set of operators. In our approach, a client agent may decide through an operator, for instance, the order in which the criteria will be used.

Recommender systems [16, 15, 11] have become an important research area in AI over the last decade. We focus on a particular form of implementing recommender systems called Recommender Servers that extends the integration of argumentation and recommender systems to a MAS setting. Recommender Servers are based on an implementation of DeLP [10] called DeLP-Server [9]. In this paper we will introduce a defeasible logic programming recommender server that gives to the clients the ability to decide how multiple argument comparison criteria can be combined. A set of criteria-combination operators is proposed to provide that capability.

The rest of the paper is structured as follows. Next, in Section 2 we will present the necessary background introducing basic definitions and some works that will be used in the rest of the paper; in Section 3 we show, in an example in DeLP, how structure of dialectical tree is associated the preference criterion used; then, in Section 4 we will introduce a preference based recommender server. To illustrate the formalism, in Section 5 we introduce an example in DeLP. In Section 6 we discuss related work and the possible directions for our future work. Finally, in Section 7 offer our conclusions.

2 Background

In this section a brief introduction of Defeasible Logic Programming (DeLP) and DeLP-servers is included. In DeLP, strict knowledge is represented using facts and strict rules and defeasible rules are used for representing the weak or tentative information. Facts are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$". An overlined literal will denote the complement of that literal with respect to strong negation, *i.e.,*$\overline{L}$ is $\sim L$, and $\overline{\sim L}$ is L. Defeasible Rules are denoted $L_0 \prec L_1, \ldots, L_n$ and represent defeasible knowledge, *i.e.,* tentative information, where the head L_0 is a literal and the is a set of literals. In this paper we will consider a restricted form of program that do not have strict rules. Thus, a restricted DeLP-program $\mathcal{P}$ will be denoted $\mathcal{P}{=}(\Pi, \Delta)$ distinguishing the set of facts Π, and the set of defeasible rules Δ. We refer to the interest reader to [10] and [9] for more details.

Example 2 *Continuing with Example 1, let $\mathcal{P}_h$ be a DeLP-program that models the described information about hotel clients;*

$$\mathcal{P}_h = \left\{ \begin{array}{l} \sim junior\_suite \prec travel\_alone, \sim jacuzzi \\ junior\_suite \prec safe, fridge, \sim jacuzzi \\ junior\_suite \prec travel\_alone, safe, fridge \\ junior\_suite \prec travel\_alone \\ \sim junior\_suite \prec single\_room \\ jacuzzi \prec tired \\ single\_room \prec short\_stay \end{array} \right\}$$

Two literals are contradictory if they are complementary. In DeLP it is assumed that the set of facts is non-contradictory. That is, there can not be two complementary facts in a valid DeLP-program. Since contradictory literals can be derived from a DeLP-program, the derivation does not provide a strong enough notion to characterize the inference final of the system. For that reason when reasoning with contradictory and dynamic information, DeLP builds arguments from this program. An argument $\mathcal{A}$ for a literal L, denoted $\langle \mathcal{A}, L \rangle$ ($\mathcal{A}$ for short) is a minimal, non contradictory set of defeasible rules such that together with the program strict knowledge allows the derivation of L that will also be called the "conclusion" supported by $\mathcal{A}$. For a given DeLP-program $\mathcal{P}$ the set of all possible arguments will be denoted as *Args*. An argument $\mathcal{A}$ is said to be a subargument of $\mathcal{A}_1$ if $\mathcal{A} \subseteq \mathcal{A}_1$.

To deal with contradictory information, in DeLP, arguments are built. When considering a literal L, supported by argument $\mathcal{A}_2$, arguments that contradict $\mathcal{A}_2$ could exist. An argument $\mathcal{A}_1$ contradicts $\mathcal{A}_2$ iff $\mathcal{A}_1$ is a counter-argument for $\mathcal{A}_2$. We say that the argument $\langle \mathcal{A}_1, L_1 \rangle$ *counter-argues*, *rebutts*, or *attacks* $\langle \mathcal{A}_2, L_2 \rangle$ at literal L, if and only if there exists a sub-argument $\langle \mathcal{A}, L \rangle$ of $\langle \mathcal{A}_2, L_2 \rangle$ such that L and L_1 are contradictory.

Given an argument $\mathcal{A}_1$ that is a counter-argument for $\mathcal{A}_2$, in order to decide which one prevails, these two arguments must be compared using some criterion. In [10], if the argument $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$ w.r.t. the comparison criterion, then $\mathcal{A}_1$ prevails and it will be called a *proper defeater* for $\mathcal{A}_2$. If $\mathcal{A}_2$ is preferred to $\mathcal{A}_1$, then $\mathcal{A}_1$ will not be considered as a defeater for $\mathcal{A}_2$. If neither argument is preferred over the other, a blocking situation occurs, and we will say that $\mathcal{A}_1$ is a *blocking defeater* for $\mathcal{A}_2$. In a blocking defeater situation between $\mathcal{A}_1$ and $\mathcal{A}_2$, both arguments are defeated. In fact, an argument $\mathcal{A}_1$ is a defeater for $\mathcal{A}_2$ iff either $\mathcal{A}_1$ is a proper or blocking defeater for $\mathcal{A}_2$.

In DeLP a query Q is *warranted* from a program $\mathcal{P}$ if there exists a non-defeated argument $\mathcal{A}$ supporting Q. To establish whether an argument $\mathcal{A}$ is a non-defeated argument, defeaters for $\mathcal{A}$ are considered. In turn, each defeater could be defeated, generating a sequence of arguments called *argumentation line*. In an argumentation line, arguments in even positions are known as support arguments and arguments in odd positions are interference arguments.

Like in DeLP, we will consider only acceptable argumentation lines. An acceptable argumentation line Λ is an argumentation line holding that:

- Λ is a sequence finite.
- The set of supporting arguments in Λ is non contradictory and the set of interfering arguments in Λ is non contradictory.
- No argument $\mathcal{A}_j$ in Λ is a subargument of an argument $\mathcal{A}_i$ in Λ, $i < j$.
- Every blocking defeater $\mathcal{A}_i$ in Λ is defeated by a proper defeater $\mathcal{A}_{i+1}$ in Λ.

See [10] for a more detailed motivation of *acceptable argumentation lines*.

For each argument may there exist more a defeater; the presence of multiple defeaters for an argument produces an argumentation lines ramification, giving rise a defeaters tree which is called *dialectical tree*.

Definition 1 (Dialectical Tree) *[10]*
Let $\mathcal{A}_0$ be an argument from a program $\mathcal{P}$. A dialectical tree for $\mathcal{A}_0$, denoted $\mathcal{T}_{\mathcal{A}_0}$, is defined as follows:

1. *The root of the tree is labeled with $\mathcal{A}_0$.*
2. *Let N be a node of the tree labeled $\langle \mathcal{A}_n, h_n \rangle$, and*
   $\Lambda = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$ the sequence of labels of the path from the root to N. Let $\mathcal{B}_1$, $\mathcal{B}_2$, …, $\mathcal{B}_k$ be all the defeaters for $\mathcal{A}_n$.
   For each defeater $\mathcal{B}_i$ $(1 \leq i \leq k)$, such that, the argumentation line $\Lambda' = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n]$ is acceptable, then the node N has a child N_i labeled $\mathcal{B}_i$.
   If there is no defeater for $\mathcal{A}_n$ or there is no $\mathcal{B}_i$ such that Λ' is acceptable, then N is a leaf.

Each path from the root to a leaf corresponds to one different acceptable argumentation line. In a dialectical tree, every node (except the root) is a defeater of its parent, and leaves are non-defeater argument. Thus, in a dialectical tree every node can be marked as defeated and undefeated. Marking of a dialectical tree is a process which will be done by making every node from the leaf to root. Leaf nodes in a dialectical tree will be marked as "*U*", an inner node will be marked as "*D*" iff it has at least a child marked as "*U*", and an inner node will be marked as "*U*" iff each of its children is marked as "*D*".

The dialectical tree is built to establish whether a queried literal is warrant or not from a program DeLP. If the argument $\mathcal{A}$ at the root of a given dialectical tree is marked as "*U*", then $\mathcal{A}$ provides a warrant for the queried literal.

The process of argumentation finishes when DeLP returns an answer. The answer for a query Q from a DeLP-program $\mathcal{P}$ is either: YES, if Q is warranted from P; NO, if the complement of Q is warranted from $\mathcal{P}$; UNDECIDED, if neither Q nor its complement are warranted from $\mathcal{P}$; or UNKNOWN, if Q is not in the language of the program $\mathcal{P}$.

Recently, approaches that use DeLP for knowledge representation in recommender systems were proposed. In [9] an implementation of DeLP, called DeLP-server, has been presented; this system provides an argumentative reasoning service for multi-agent systems. A DeLP-server is a stand-alone application that stores a DeLP-program that is used to answer client queries. To answer queries, the DeLP-server will use the public knowledge stored and represented as Defeasible Logic Program complementing it with individual knowledge that clients might send, thus creating a particular scenario for the query.

In [9], three operators for DeLP-programs (DeLP-operators for short) were introduced to consider different ways in which the specific information of the clients is taken into account at the moment of computing answers; these proposed operators would temporally modify the public knowledge stored in the program. The first operator, dentod "$*$" is used to prioritize strict information stored in sever when there are conflicts with contextual information. The second one, denoted "$+$", which is used to prioritize contextual information when it is in conflict with strict knowledge in the server. For example, let $H = \{\sim a, b, c\}$ be contextual information and $\mathcal{P} = \{a, b, \sim j\}$ be a DeLP-program, them $\mathcal{P} * H = \{a, b, c, \sim j\}$ and $\mathcal{P} + H = \{\sim a, b, c, \sim j\}$. The third operator, denoted "$-$", that uses the context to ignore pieces of knowledge of the stored program when answering the query ,*i.e.,* $\mathcal{P} - H = \{a, \sim j\}$. In this paper we will use the operator "$+$" that in case of a conflict gives priority to information of the clients.

A query is a ground literal Q, and the set of all possible queries will be denoted $\mathbb{Q}$. In [9], several contextual queries were defined, these types of queries allow the inclusion of private pieces of information related to the agents's particular context such that together with operators for DeLP-programs will be taken into consideration when computing the answers. The information that modifies the public knowledge stored in the DeLP-server is called context, denoted $\mathcal{C}o$.

Definition 2 (Contextual query) *Given a DeLP-program $\mathcal{P}$, a contextual query for $\mathcal{P}$ is a pair [$\mathcal{C}o$, Q] where $\mathcal{C}o$ is a non-contradictory set of ground literals, and Q is a query.*

Example 3 *Suppose that a client may want know whether the hotel suggests a rom "junior suite". Given the contextual query:*

$$[\mathcal{P}_c, junior\_suite]$$

Consider the program introduced in Example 2. Let $\mathcal{P}_c$ be the private pieces of information related to the client's particular context. The DeLP-program $\mathcal{P}_m$ results from applying the operator "+";

$$\mathcal{C}o_{Joan} = \mathcal{C}o_{Michael} = \mathcal{P}_c = \left\{ \begin{array}{l} travel\_alone \\ short\_stay \\ fridge \\ safe \end{array} \right\} \mathcal{P}_m = \left\{ \begin{array}{l} \sim junior\_suite \prec travel\_alone, \sim jacuzzi \\ junior\_suite \prec safe, fridge, \sim jacuzzi \\ junior\_suite \prec travel\_alone, safe, fridge \\ junior\_suite \prec travel\_alone \\ \sim junior\_suite \prec single\_room \\ jacuzzi \prec tired \\ single\_room \prec short\_stay \\ travel\_alone \\ short\_stay \\ fridge \\ safe \end{array} \right\}$$

where Joan and Michael particular context is the same.

In [9], the temporal scope of the contextual information sent in a query is limited and it will disappear with the finalization of the process performed by the DeLP-server to answer that query.

3 Preference criteria

In DeLP the argument comparison criterion is modular, thus the most appropriate criterion for the domain that is being represented can be selected. In fact, the argument comparison criterion is a fundamental piece for the process argumentative, having important influence in building the dialectical tree; consequently, the answer to a query may vary according to criterion used. We will denote a preference criterion with the letter C and the set of preference criteria with $\mathsf{S} = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_n\}$.

A preference criteria can be represented by a preference relation or a function. For our convenience, we say that, given a set de arguments *Args*, a preference criterion is a function $\mathsf{C} : Args \times Args \longrightarrow \{\bot, \top\}$, obtaining $\top$ when the first argument is preferred over the second, and $\bot$ otherwise. In the following example show two distinct marked dialectical trees for *junior_suite* built using different criteria.

Example 4 *To briefly illustrate how two criteria can generate different marked trees, suppose that we want to answer a query for "junior_suite". Extending Example 3, from program $\mathcal{P}_m$ the following arguments in favor of recommending and not a "junior suite" can be built:*

$\mathcal{A}_1 = \{junior\_suite \prec travel\_alone\}$

$\mathcal{A}_2 = \{{\sim}junior\_suite \prec travel\_alone, {\sim}jacuzzi\}$

$\mathcal{A}_3 = \{{\sim}junior\_suite \prec single\_room\}$

$\mathcal{A}_4 = \{junior\_suite \prec safe, fridge, {\sim}jacuzzi\}$

$\mathcal{A}_5 = \{junior\_suite \prec travel\_alone, safe, fridge\}$

Continuing with our running example, we assume the criterion $\mathsf{C}_{comfort}$ that favors "comfort" and the criterion C_{price} that favors "price". On the one hand, the function $\mathsf{C}_{comfort}$ will return $\top$ iff the first argument has as information that the room has fridge and safe, and $\perp$ otherwise. On the other hand, C_{price} will return $\top$ iff the first argument has as information that the room has not jacuzzi, and $\perp$ otherwise.

In Fig. 1 each arguments is depicted in triangle-shaped, and dashed lines denote blocking defeat relations and solid lines denote proper defeat relations. Figure 1-(a) gives a graphical representation of a dialectical tree considering $\mathsf{C}_{comfort}$ as argument comparison cirterion, where the argument at the root node is marked as "U"; thus, the conclusion "junior_suite" is warranted, then the answer for the query is YES. However, note that in Figure 1-(b), C_{price} is the used criterion and the answer for the query junior_suite is UNDECIDED*, i.e., neither the conclusion "junior_suite" nor its complement are warranted.*

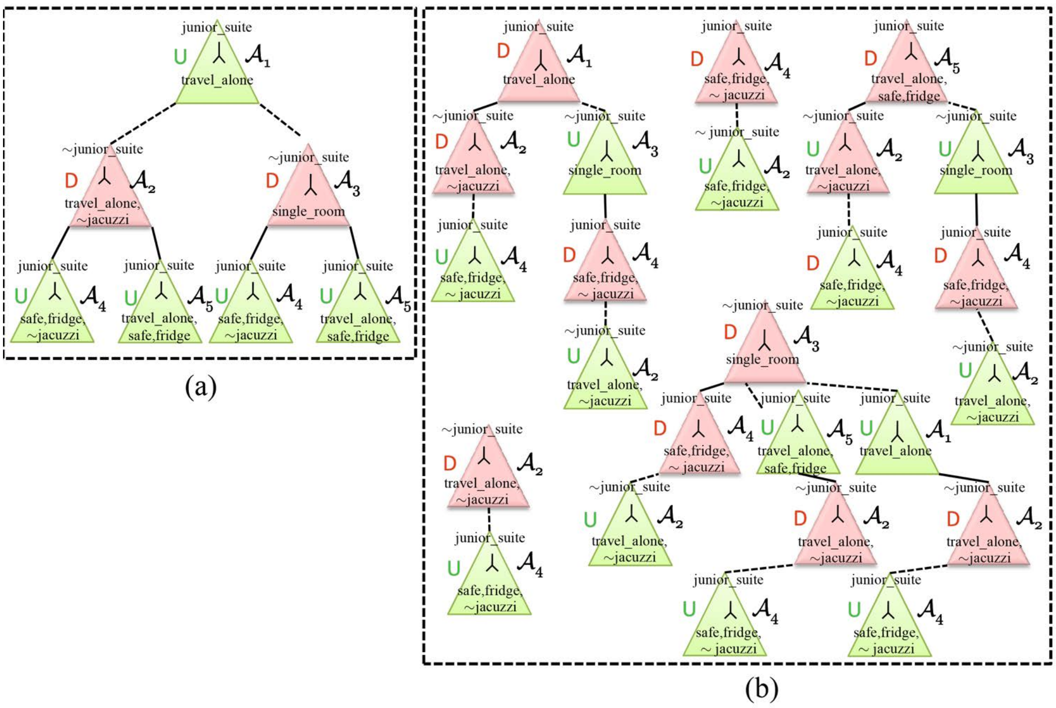

Figure 1: Marked dialectical tree using $\mathsf{C}_{comfort}$ (a) and C_{price} (b).

4 Argumentative reasoning with multiple preferences

As we have stated, our focus of research here is formalizing a server model with multiple comparison criteria. In first place, we will provide a conceptual guide to address this issue by means of what we call a preference-based reasoning server, or $\mathcal{PRS}$-server for short. The formal definition of $\mathcal{PRS}$-server will be introduced after the definition of its components. Figure 2 shows the graphical representation of the proposed server.

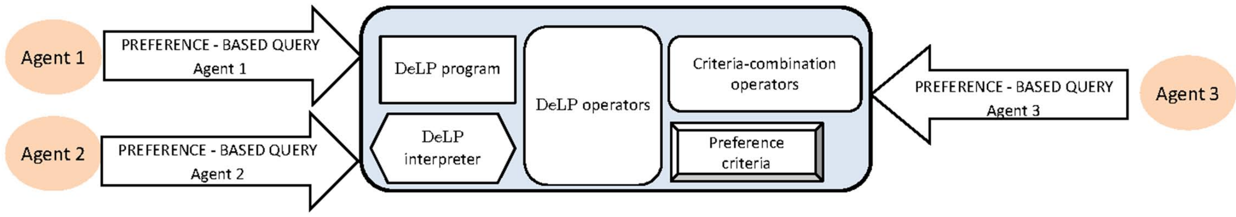

Figure 2: Preference-based reasoning server.

The figure depicts three client agents sending a contextual query, and the main components of a preference-based reasoning server. The components of proposed server will be defined and explained below.

As we have said, from a DeLP-program several arguments can be built. Given two arguments $\mathcal{A}_1$ and $\mathcal{A}_2$ in conflict it is necessary to use a preference criterion to decide which argument prevails and if $\mathcal{A}_1$ is the prevailing argument, $\mathcal{A}_1$ is said to be a defeater of $\mathcal{A}_2$. A $\mathcal{PRS}$-server will be integrated by a set of preference criteria $\mathsf{S} = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_n\}$.

Usually, existing reasoning services based on defeasible argumentation make use of a unique preference criterion that is an integral part of the inference mechanism. A distinctive feature of our proposed server, is the capacity of combining multiple preference criteria. To achieve this, the server will have available specific operators to combine different criteria. Thus, we represent a criteria-combination operator as θ^n where n represents its arity. Next, some examples of possible operators will be introduced.

Example 5 *Given two conflicting arguments* $\mathcal{A}_1, \mathcal{A}_2 \in$ *Args and the preference criteria* $\mathsf{C}_1, \mathsf{C}_2 \in \mathsf{S}$. *Consider the following three binary operators applied to the criteria* $\mathsf{C}_1, \mathsf{C}_2$*:*

- *the operator* $\boxtimes$ *is such that the expression* $\mathsf{C}_1 \boxtimes \mathsf{C}_2$ *says the argument* $\mathcal{A}_1$ *prevails iff* $\mathcal{A}_1$ *is preferred to* $\mathcal{A}_2$ *for both preference criteria.*
- *the operator* $\boxminus$ *is such that the expression* $\mathsf{C}_1 \boxminus \mathsf{C}_2$ *establishes that the argument* $\mathcal{A}_1$ *prevails iff* $\mathcal{A}_1$ *is preferred to* $\mathcal{A}_2$ *for at least one criterion.*
- *the operator* $\boxdot$ *is such that the expression* $\mathsf{C}_1 \boxdot \mathsf{C}_2$ *says the argument* $\mathcal{A}_1$ *prevails iff* $\mathcal{A}_1$ *is preferred to* $\mathcal{A}_2$ *with respect to* C_1 *and if not, then it checks if* $\mathcal{A}_1$ *is preferred to* $\mathcal{A}_2$ *with respect to* C_2. *That is, it returns the same result to* $\mathsf{C}_1 \boxminus \mathsf{C}_2$ *but the order of evaluation is fixed and must be done by the server in such order: first* C_1 *and then* C_2.

These operators will be formally defined below.

We have introduced three possible operators, nevertheless, the available operators will depend on the particular reasoning server. It is also possible to define new operators that could have a particular behavior related to a specific application; for instance, some operators could requiere sets of preference criteria, meanwhile others can be used to enable or disable criteria. Moreover, depending on their properties, these operators may be combined leading to more complex expressions.

A contextual query has the particularity of including the client's own information and that information will be used by the server to compute an answer. However, if the server uses the criteria chosen by a client agent, then it will be necessary to adapt the structure of the context query to include them. The change consists in expanding the contextual query with an expression indicating to the server how to solve the query using the criteria selected by the client agents.

A server will answer a query as long as the preference criteria and the criteria-combination operators indicated by the client are part of a criteria-combination expression, or *cc-exp* for short. We use $\mathbb{E}$ to denote the set of all possible criteria-combination expressions.

Definition 3 (Criteria-combination expression) *Let* S *a set of preference criteria and* Θ *a set of criteria-combination operators. An expression* E *is a cc-exp iff:*

- *$E \in \mathsf{S}$ or*
- *$E = \theta^n(E_1, E_2, \ldots, E_n)$ where E_i is a cc-exp and $\theta^n \in \Theta$ with arity-n.$(1 < i < n)$*

In an expression could arise two situations, either the expression is a preference criterion, or the expression is an operator applied to a set of *cc-exp*. Clearly we can combine any number of preference criteria by gradually applying a sequence of the same or different composition operators. However, the semantics of an n-ary combination, $n > 2$, depend on whether the corresponding operators are associative. In this work, we assume that the associative property is satisfied by every operators introduced above.

Example 6 *Consider operators of Example 5. Given two conflicting arguments $\mathcal{A}_1, \mathcal{A}_2 \in Args$ and a set of preference criteria $\mathsf{S}_1 = \{\mathsf{C}_1, \mathsf{C}_2, \mathsf{C}_3, \mathsf{C}_4\}$. Two possible cases of criteria-combination expressions will be presented bellow.*

$E_1 = ((\mathsf{C}_1 \boxminus \mathsf{C}_2) \boxtimes \mathsf{C}_3)$.

$E_2 = ((\mathsf{C}_1 \boxtimes \mathsf{C}_2) \boxdot (\mathsf{C}_3 \boxtimes \mathsf{C}_4))$

In E_1, we will say that the argument $\mathcal{A}_1$ prevails over $\mathcal{A}_2$ iff it is preferred by the criterion C_3 and at least one of the rest of the criteria. In E_1, the argument $\mathcal{A}_1$ prevails over $\mathcal{A}_2$ iff it is preferred by both criteria, C_1 and C_2, or else by the criteria C_3 and C_4.

Example 7 *Consider Example 4 and the operators introduced in Example 5. The criteria to select a room for Joan and Michael, respectively, will be represented by means of the following criteria-combination expressions:*

$E_{Joan} = (\mathsf{C}_{comfort} \boxtimes \mathsf{C}_{price})$

$E_{Michael} = (\mathsf{C}_{comfort} \boxminus \mathsf{C}_{price})$

The example above shows how simple expressions such as E_{Joan} and more complex expressions such as $E_{Michael}$ could be built. As we will show in the following section, this example is of special interest since it will serve to show the different results two queries using these expressions will produce.

Thus, the client agents can indicate how their queries have to be solved by the server. For that reason, the *cc-exp* denoting how the client wants to use the server preference criteria, will be included in the queries. This new type of contextual query will be called *preference-based query.*

Definition 4 (Preference-based query) *A preference-based query* PQ *is a tuple $[\mathcal{C}o, E, Q]$ where $\mathcal{C}o$ is a particular context for* PQ*, E is a cc-exp, and Q is a query.*

It is important to mention that PQ is an extension of the contextual query introduced in [9]. We refer the interested reader to [9] for details on those queries.

Example 8 *Going back to Example 3 and considering Example 7. Given the query "junior_suite", two preference based queries can be built:*

$[\mathcal{C}o_{Joan}, E_{Joan}, junior\_suite]$

$[\mathcal{C}o_{Michael}, E_{Michael}, junior\_suite]$

The criteria-combination expressions are solved by the inference mechanism. In particular, the DeLP-interpreter of a $\mathcal{PRS}$-server will be responsible of the processing and answering of client queries. As defined next, a DeLP-interpreter will be represented, in general, as a function such that given a program, expression and a query, returns the corresponding answer.

Definition 5 (DeLP-interpreter) *Let $\mathbb{P}$ be the set of valid DeLP-programs, $\mathbb{E}$ be the set of possible cc-exps and $\mathbb{Q}$ be the set of possible queries. A DeLP-interpreter is a function $\mathcal{I} : \mathbb{P} \times \mathbb{E} \times \mathbb{Q} \longrightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of possible answers for $\mathcal{PRS}$-server, i.e., $\mathbb{R} = \{$*NO, YES, UNDECIDED, UNKNOWN*$\}$.*

Given two conflicting arguments $\mathcal{A}_1, \mathcal{A}_2 \in Args$ and a *cc-exp* E. To solve a *cc-exp* the interpreter will use a function $eval(E, \mathcal{A}_2, \mathcal{A}_1)$ such that its range is $\{\perp, \top\}$ which correspond to the answers for a *cc-exp* given.

The application pattern of the preference criteria is established when preferences combination operators are defined. For instance, consider the set of criteria-combination operators $\Theta_j = \{\boxtimes, \boxminus, \boxdot\}$ presented in Example 5 and two *cc-exps* E_i and E_j. The evaluation of each operator belonging to the set Θ_j may be defined as:

i) $eval(E, \mathcal{A}_2, \mathcal{A}_1) = \mathsf{C}(\mathcal{A}_2, \mathcal{A}_1)$ if $E = \mathsf{C}$, or

i) $eval(E_i \boxtimes E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ and $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

ii) $eval(E_i \boxminus E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ or $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

iii) $eval(E_i \boxdot E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$ if $eval(E_i, \mathcal{A}_2, \mathcal{A}_1) = \top$ or else $eval(E_j, \mathcal{A}_2, \mathcal{A}_1) = \top$, or

iv) $\bot$ in other case.

As already mentioned, the argument comparison criterion is a important element in the definition of a defeat relation. In our approach, the function "*eval*" defines the preference relation which is used by the DeLP-interpreter to determine the defeat relation between conflicting arguments.

Now, we formally present the concept of preference-based reasoning server.

Definition 6 (Preference-based reasoning server) *A Preference-based reasoning server is a 5-tuple* $\langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathsf{S}, \Theta \rangle$*, where* $\mathcal{I}$ *is a DeLP-interpreter,* $\mathcal{O}$ *is a set of DeLP-operators,* $\mathcal{P}$ *is a DeLP-program,* S *is a set of preference criteria, and* Θ *is a set of criteria-combination operators.*

In this work, queries are answered using public knowledge stored in the server, plus individual knowledge sent with the query, and an expression where several criteria are selected and combined. The answer will be obtained by means of an argumentative inference mechanism.

Definition 7 (Answer for a query) *Let* $\langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathsf{S}, \Theta \rangle$ *be a* $\mathcal{PRS}$*-server,* $\mathrm{PQ} = [\mathcal{C}o, E, Q]$ *be a preference-based query for* $\mathcal{PRS}$*-server, and* $\mathcal{P}'$ *be a modified program for the context* $\mathcal{C}o$*. An answer for* PQ *from* $\mathcal{PRS}$*-server, denoted Ans(*$\mathcal{PRS}$*-server,E,Q), corresponds to the result of the function* $\mathcal{I}(\mathcal{P}', E, Q)$*.*

5 Application example

In this section we will present a DeLP example showing how the answer to a query varies according to the way in which the criteria used by the server are combined. Let $\mathcal{P}_h$ be the DeLP-program and $\mathcal{P}_c$ be the context presented in Example 2 and 3 respectively, consider the preference-based queries introduced in Example 8;

1. $[\mathcal{C}o_{Joan}, E_{Joan}, junior\_suite]$.
2. $[\mathcal{C}o_{Michael}, E_{Michael}, junior\_suite]$.

such that

$E_{Joan} = (\mathsf{C}_{comfort} \boxtimes \mathsf{C}_{price})$.

$E_{Michael} = (\mathsf{C}_{comfort} \boxminus \mathsf{C}_{price})$.

In both queries mentioned above, the same DeLP $\mathcal{P}_m$ presented in Example 3 is obtained. As showed in Example 4, from the program $\mathcal{P}_m$ several conflicting arguments can be built.

$\mathcal{A}_1 = \{junior\_suite \prec travel\_alone\}$

$\mathcal{A}_2 = \{\sim junior\_suite \prec travel\_alone, \sim jacuzzi\}$

$\mathcal{A}_3 = \{\sim junior\_suite \prec single\_room\}$

$\mathcal{A}_4 = \{junior\_suite \prec safe, fridge, \sim jacuzzi\}$

$\mathcal{A}_5 = \{junior\_suite \prec travel\_alone, safe, fridge\}$

To answer a query, it is necessary to determine whether there exists a non-defeated argument supporting it. In order to decide whether the argument at the root of a given dialectical tree is marked as "U", it is necessary to perform a bottom-up-analysis of the tree. Now, consider the first preference-based query presented above:

$$[\mathcal{C}o_{Joan}, E_{Joan}, junior\_suite]$$

The dialectical tree shown in Fig. 3-a is obtained; since there are not counterarguments that could defeat $\mathcal{A}_1$ then $\mathcal{A}_1$ remains undefeated. The conclusion *junior_suite* is warranted, then the answer for the preference-based query is YES. Moreover, note that the answer for the second preference-based query will not be the same w.r.t. the first one:

$$[\mathcal{C}o_{Michael}, E_{Michael}, junior\_suite]$$

Some defeat relations between arguments will vary regarding ones of the first case, and if the whole argumentative process is considered, then the answer for the query *junior_suite* is UNDECIDED, *i.e.,* neither the conclusion *junior_suite* nor its complement are warranted (see Fig. 3-b). In contrast to Section 3, in this section we present a DeLP example showing how the structure of a marked dialectical tree; consequently, the answer to a query varies according to the way in which the criteria are combined.

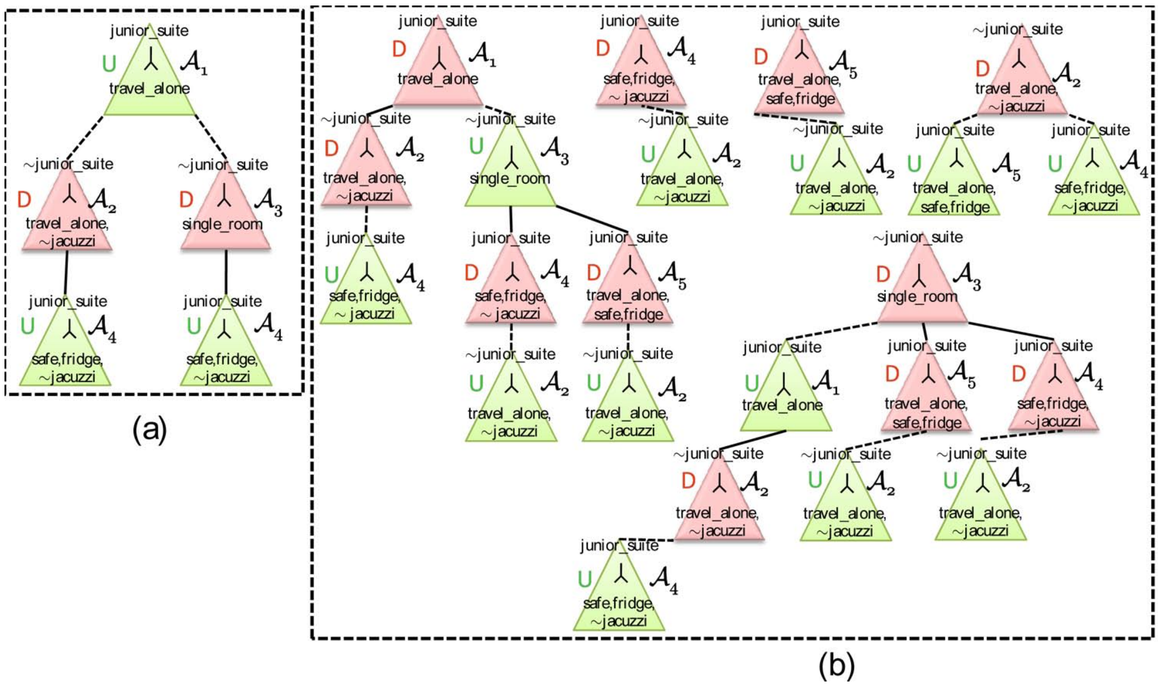

Figure 3: Marked trees after applying the expressions E_{Joan} (a) and $E_{Michael}$ (b)

6 Related and future work

Our approach was in part inspired by [9], where several servers can be created, and knowledge can be shared through them. Nevertheless, both approaches have several differences. In contrast with us, they use a preference criteria embedded in the interpreter, *i.e.,* to answer a query the server is configured to use the same specific criterion. Finally, we provide clients with the possibility of selecting what criteria a server should use to compute the answer for a specific query.

In [3] an approach to handle multiple preference was proposed. To determine the acceptable arguments, the set of preferences is linearly ordered using another preference relation. Their main contribution is to take into account contextual preferences which means that several pre-orderings on the knowledge base may be taken into account together, *i.e.,* preferences which depend upon a particular context. In contrast with us, they provide a framework where the preferences are ordered, in our framework this situation is a particular case, *i.e.,* depending on the criteria-combination operators defined in the server.

Several approaches about combination of preference criteria can be found in [13, 4, 14, 12], although in different directions from ours. In [12], an extension of DeLP has been proposed. The authors introduced an argumentative framework where, for expressing preference on arguments, more than one criterion is considered. The defeat relation between arguments combine both temporal criteria of t-DeLP and the belief strength criteria from P-DeLP. In contrast, our approach generalizes the amount of argument comparison criteria used to establish the defeat relation, and the way which they are combined. Another work that focuses in the combination of preference relations is the one by Jan Chomicki [8]. The main contributions of this article is a logical framework for formulating preferences as preference formulas and distinguishing different type of composition of preference relations.

As future work we are developing an implementation of a DeLP-server that can dynamically handle multiple preference criteria. We are also interested in studying the properties of the criteria-combination operators to define operators for concrete reasoning servers. Another extension will be to integrate our proposed framework with others argumentative systems similar to DeLP.

7 Conclusions

We have presented a model that allows an argumentative reasoning server to handle multiple preference criteria. For this, we formally defined the notion of criteria-combination expression, which allows us indicating criteria to be used by the server, and the way in which these ones are combined. We have introduced three different criteria-combination operators, showing how these operators are evaluated within such expressions. In our approach, DeLP was proposed for knowledge representation and therefore the DeLP-interpreter is in charge of solving queries. To solve each conflict between arguments, the DeLP-interpreter uses a function *eval* which is used to assess the criteria-combination expressions contained in the queries. This means that queries are answered considering the combination of several criteria. Finally, in Section 5 an example in DeLP is presented where an agent performs two queries with the same context but with different preference criteria combinations getting different results.

Acknowledgements

This work is partially supported by CONICET, Universidad Nacional de Entre Ríos (PID-UNER 7041), Universidad Nacional del Sur, SGCyT.

References

[1] Teresa Alsinet, Carlos Iván Chesñevar, Lluis Godo, and Guillermo Ricardo Simari. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets and Systems*, 159(10):1208–1228, 2008.

[2] Leila Amgoud, Jean-François Bonnefon, and Henri Prade. An argumentation-based approach to multiple criteria decision. In *ECSQARU*, pages 269–280, 2005.

[3] Leila Amgoud, Simon Parsons, and Laurent Perrussel. An argumentation framework based on contextual preferences. In *FAPR'2000*, pages 59–67, 2000.

[4] Hajnal Andréka, Mark Ryan, and Pierre-Yves Schobbens. Operators and laws for combining preference relations. *J. Log. Comput.*, 12(1):13–53, 2002.

[5] Grigoris Antoniou, Michael J. Maher, and David Billington. Defeasible logic versus logic programming without negation as failure. *J. Log. Program.*, 42(1):47–57, 2000.

[6] Marcela Capobianco, Carlos Iván Chesñevar, and Guillermo Ricardo Simari. Argumentation and the dynamics of warranted beliefs in changing environments. *Autonomous Agents and Multi-Agent Systems*, 11(2):127–151, 2005.

[7] Marcela Capobianco and Guillermo Ricardo Simari. A proposal for making argumentation computationally capable of handling large repositories of uncertain data. In *SUM*, pages 95–110, 2009.

[8] Jan Chomicki. Preference formulas in relational queries. *ACM Trans. Database Syst.*, 28(4):427–466, 2003.

[9] Alejandro J. García, Nicolás D. Rotstein, Mariano Tucat, and Guillermo R. Simari. An argumentative reasoning service for deliberative agents. In *KSEM*, pages 128–139, 2007.

[10] Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming (TPLP)*, 4:95–138, 2004.

[11] Alejandro J. García and Guillermo R. Simari. Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers. *Argument & Computation , DOI: 10.1080/19462166.2013.869767*, pages 63–88, 2014.

[12] Lluis Godo, Enrico Marchioni, and Pere Pardo. Extending a temporal defeasible argumentation framework with possibilistic weights. In *JELIA*, pages 242–254, 2012.

[13] Souhila Kaci. *Working with Preferences: Less Is More.* Cognitive Technologies. Springer, 2011.

[14] Werner Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322, 2002.

[15] Joseph A. Konstan. Introduction to recommender systems: Algorithms and evaluation. *ACM Trans. Inf. Syst.*, 22(1):1–4, 2004.

[16] Michael J. Maher, Andrew Rock, Grigoris Antoniou, David Billington, and Tristan Miller. Efficient defeasible reasoning systems. *International Journal on Artificial Intelligence Tools*, 10(4):483–501, 2001.

[17] Iyad Rahwan and Guillermo R. Simari. *Argumentation in Artificial Intelligence.* Springer Publishing Company, Incorporated, 1st edition, 2009.

[18] Guillermo R. Simari and Ronald P. Loui. A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence*, 53(2-3):125–157, 1992.

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