Expert Systems with Applications 42 (2015) 8243–8258

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Improving argumentation-based recommender systems through context-adaptable selection criteria

Juan C.L. Teze[a,b,c,*], Sebastian Gottifredi[a,c], Alejandro J. García[a,c], Guillermo R. Simari[a]

[a] Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur – Alem 1253, (8000) Bahía Blanca, Buenos Aires, Argentina
[b] Agents and Intelligent Systems Area, Fac. of Management Sciences, Universidad Nacional de Entre Ríos – Monseñor Tavella 1424, (3200) Concordia, Entre Ríos, Argentina
[c] Consejo Nacional de Investigaciones Científicas y Técnicas, Av. Rivadavia 1917, (C1033AAJ) Ciudad Autónoma de Buenos Aires, Argentina

ARTICLE INFO

Article history:
Available online 9 July 2015

Keywords:
Reasoning server
Argumentation system
Multiple preference criteria
Criterion selection

ABSTRACT

Recommender systems based on argumentation represent an important proposal where the recommendation is supported by qualitative information. In these systems, the role of the comparison criterion used to decide between competing arguments is paramount and the possibility of using the most appropriate for a given domain becomes a central issue; therefore, an argumentative recommender system that offers an interchangeable argument comparison criterion provides a significant ability that can be exploited by the user. However, in most of current recommender systems, the argument comparison criterion is either fixed, or codified within the arguments. In this work we propose a formalization of context-adaptable selection criteria that enhances the argumentative reasoning mechanism. Thus, we do not propose of a new type of recommender system; instead we present a mechanism that expand the capabilities of existing argumentation-based recommender systems. More precisely, our proposal is to provide a way of specifying how to select and use the most appropriate argument comparison criterion effecting the selection on the user's preferences, giving the possibility of programming, by the use of conditional expressions, which argument preference criterion has to be used in each particular situation.

1. Introduction

Recommender systems represent an important addition to platforms where the user is confronted with many choices. To come up with recommendations, recommender systems usually rely on user preferences, popularity indexes, or on similarity measures defined between users or contents, which are computed on the basis of methods coming from the social sciences, the information retrieval or the machine learning communities (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013). Although the effectiveness of these kind of recommenders is remarkable, in Chesñevar, Maguitman, and González (2009, chap. 20) some limitations regarding their underlying quantitative model are pointed out. On the one hand, these quantitative models are not equipped with mechanism to revise previous conclusions; thus, given that the dynamic nature of user preferences usually leads to conflictive and incomplete domain information, these recommenders deal poorly with changes in preferences. On the other hand, these quantitative approaches do not have a clean underlying model that is easily understandable by the user; this obscuring the explanations the system provides to support its recommendations. As mentioned in Tintarev and Masthoff (2007), qualitative explanations are relevant to user support systems, because users prefer recommendations when they can understand the reasons behind its selection.

Several current studies claim that the qualitative reasoning mechanism provided by defeasible argumentation is useful to tackle the above mentioned limitations (see Amgoud & Prade (2009), Chesñevar et al. (2009, chap. 20), & Monteserin & Amandi (2011)). The reason behind this observation is that argumentation can qualitatively deal with conflicting domain information, and its dialectical reasoning nature can provide a clear explanation of the recommendation without further cost as shown in Garća, Chesñevar, Rotstein, and Simari (2013). Such advantages have motivated the development of several argumentation based recommender and decision support systems: Bedi and Vashisth (2011), Bedi and Vashisth (2015), Briguez, Capobianco, and Maguitman (2013), Briguez et al. (2014), Chesñevar et al. (2009, chap. 20), and Amgoud and Prade (2009).

* Corresponding author at: Artificial Intelligence Research and Development Laboratory, Department of Computer Science and Engineering, Universidad Nacional del Sur – Alem 1253, (8000) Bahía Blanca, Buenos Aires, Argentina. Tel.: +54 3455288349.

E-mail addresses: jct@cs.uns.edu.ar (J.C.L. Teze), sg@cs.uns.edu.ar (S. Gottifredi), ajg@cs.uns.edu.ar (A.J. García), grs@cs.uns.edu.ar (G.R. Simari).

http://dx.doi.org/10.1016/j.eswa.2015.06.048
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

Intuitively, defeasible argumentation provides ways of confronting contradictory statements to determine whether some particular information can be accepted as warranted (Rahwan & Simari, 2009). To obtain an answer, an argumentation reasoning process goes through a series of steps. A very important one is the comparison of conflicting arguments to decide which one prevails; this requires the introduction of a preference relation to settle the question. Argumentation frameworks using a single argument comparison criterion have been extensively studied (Amgoud, Cayrol, & Berre, 1996; Antoniou, Maher, & Billington, 2000; García & Simari, 2004; Godo, Marchioni, & Pardo, 2012; Simari & Loui, 1992; Vreeswijk, 1997).

Despite its importance, mechanisms to dynamically change the argument comparison criterion have not been extensively studied. For instance, there are works that propose to use two comparison criteria (Deagustini et al., 2013; Godo et al., 2012), but the proposed combination is fixed in the formalism. There exist other systems that allow to work with dynamic information, for instance (Tucat, Garcia, & Simari, 2009) can answer to queries based on facts received as contextual information. Nevertheless, in such systems is not possible to adjust dynamically the argument comparison criterion in order to satisfy user's preference for different contexts.

In the literature of argumentative recommender systems (Briguez et al., 2014) the importance of having the possibility of changing the comparison criteria within their systems has been recognized. However, in most of current recommender systems, the comparison criterion is either fixed (*i.e.,* cannot be changed) or codified within the arguments (*i.e.,* the arguments are built to satisfy a particular criteria) (Bedi & Vashisth, 2011, 2015; Briguez et al., 2013; Chesñevar et al., 2009, chap. 20). In others approaches, as in Deagustini, Dalibón, Gottifredi, Falappa, and Simari (2012) and Briguez et al. (2014), when several criteria are available, the system does not provide a formal way to select or change the comparison criteria that should be used. In other words, none of these approaches presents a mechanism to dynamically change the comparison criteria, and none of them propose a formalism to address the presence of multiple comparison criteria within their system.

In terms of the importance and significance of our approach, here we propose a novel approach to systematically select among several argumentation comparison criteria that can benefit most argumentative systems. In particular, even when our proposal does not focus on introducing a new type of recommender system, it presents a mechanism that contributes to the strengthening of existing argumentive-based recommender systems. In this sense, the contribution of our proposal is to provide a way of specifying a context-adaptable selection mechanism that allows to select and use the most appropriate argument comparison criterion based on user's preferences. Therefore, our approach gives the possibility of programming, by the use of conditional expressions, which argument preference criterion has to be used in each particular situation.

Next, we present an example that will serve two purposes: to motivate the main ideas of our proposal, and to serve as a running example.

Example 1. Consider an onboard computer programmed to offer recommendations to the user of the vehicle in which is installed, like suggesting a hotel nearby. To offer advice, the system uses two types of knowledge: the user's particular preferences which are obtained in advance, and particular information about the context, *e.g.* nearby hotels, which is obtained dynamically during traveling. Besides that knowledge, the system also requires certain argument comparison criteria for deciding what to recommend. For instance, suppose the system can use two argument comparison criteria: one of them based on driver's safety and the other based on driver's comfort. Then, before starting a journey, the driver informs in which situations to use each comparison criterion by specifying conditions; *e.g.*, in a dangerous zone the safety criterion has to be used, whereas in a safe zone the comfort criterion has to be selected.

To formalize our approach, we will start from Defeasible Logic Programming Servers, or *DeLP*-servers for short (García & Simari, 2014; García, Rotstein, Tucat, & Simari, 2007); this implemented framework has been used in the integration of argumentation and recommendation systems to Multi-Agent System settings (Briguez et al., 2014; Deagustini et al., 2013). These servers were developed to provide an argumentative reasoning service which is able to handle queries from several client agents. To provide a recommendation for client queries, a *DeLP*-server uses public knowledge, stored in the server, and individual knowledge of the client agents which will act as private context for such queries. The recommendation will be obtained using an argumentative reasoning mechanism (García & Simari, 2004) that considers an exhaustive analysis of arguments for and against the recommendation.

In a *DeLP*-server the argument preference criterion used for deciding which arguments prevail is fixed. Thus, users do not have the possibility of adjusting the behavior of the system towards its preferences by selecting the argument comparison criterion dynamically. A recommendation will be more compelling if the server adapts the argument comparison criterion of the inference engine to the one that suits the user's needs. Another interesting point to consider is that *DeLP*-servers act like black boxes, without providing the user of any insight of the argument preference criterion under which a recommendation will be based. For instance, considering the application domain of Example 1, the user can not express in a query for a recommendation that the server should use a different argument comparison criteria when the hotel is in a dangerous zone.

Given these considerations, it is desirable to provide the servers with the additional capability of complementing each query with way of selecting the argument preference criterion the server will use in computing the corresponding recommendation. In this paper, we will propose a defeasible logic programming recommender server with these features.

The rest of the paper is structured as follows. In Section 2, an overview of Defeasible Logic Programming (*DeLP*) and *DeLP*-servers is included. In Section 3 we will introduce the formalization of the components of our proposed framework. Then, in Section 4 some properties of how our framework handles the selection of the comparison criteria are shown. In Section 5, we will formalize our proposed mechanism for answering queries for recommendations. Finally, in Section 6, we discuss some related work, and in Section 7, we offer our conclusions and possible directions for our future work.

2. Preliminaries

In this section we will briefly introduce the background related to Defeasible Logic Programming (*DeLP* for short) and *DeLP*-servers (García & Simari, 2014). *DeLP*-servers were developed to provide an argumentative reasoning service based on *DeLP* which is able to handle queries from several client agents in a multi-agent settings (García et al., 2007). In Tucat et al. (2009), *DeLP*-serves were also used for implementing a recommender system. A brief description of *DeLP*-servers will be included below, then in the next section, this kind of servers will be extended to dynamically select the argument comparison criteria.

To answer client's queries, a *DeLP*-server uses public knowledge stored in the server and represented by a *DeLP*-program. The individual knowledge of the client agents (also represented as a *DeLP* program) acts as private context for queries and it is integrated with the public knowledge in order to compute and issue a recommendation. Answers are computed by a *DeLP* interpreter that performs an exhaustive analysis to consider all the arguments for and against a recommendation. Therefore, a *DeLP*-server can be represented as a 3-tuple $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}\rangle$ where $\mathbf{I}$ is a *DeLP*-interpreter, $\mathcal{P}$ is a *DeLP*-program which represents the public knowledge, and $\mathbb{O}$ is a set of operators for handling the integration of client contexts with $\mathcal{P}$.

We include below an introduction to the language of *DeLP* that will be used for representing public and private knowledge in our approach (a complete description of *DeLP* can be found in García & Simari (2004, 2014)). We will also include a brief description of how a *DeLP* interpreter computes an answer. Nevertheless, the explanation of operators for integrating client contexts is out of the scope of this paper. A complete explanation of this kind of operators can be found in García et al. (2007) and García and Simari (2014).

DeLP is a formalization that combines results of Logic Programming and Defeasible Argumentation. This formalism allows to represent information declaratively using rules, and employing a defeasible argumentation inference mechanism for warranting the entailed conclusions. Defeasible rules are used to represent a relation between pieces of knowledge that could be defeated after all things are considered. A *DeLP*-program is a set of facts, strict rules, and defeasible rules, defined as follows. Facts are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$". *Strict Rules* represent non-defeasible information and are denoted $L_0 \leftarrow L_1, \ldots, L_n$, where L_0 is a ground literal and $\{L_i\}_{0<i\leqslant n}$ is a set of ground literals. *Defeasible Rules* represent tentative information that may be used if nothing could be posed against it and are denoted $L_0 \prec L_1, \ldots, L_n$, where L_0 is a ground literal and $\{L_i\}_{0<i\leqslant n}$ is a set of ground literals. A defeasible rule *Head* $\prec$ *Body* expresses that *reasons to believe in the antecedent Body give reasons to believe in the consequent Head*. Following García and Simari (2004), *DeLP* rules can also be represented as schematic rules with variables. As usual in Logic Programming, schematic variables are denoted with an initial uppercase letter. For convenience a *DeLP*-program $\mathcal{P}$ can also be denoted as (Π, Δ) distinguishing the subset Π of facts and strict rules, and the subset Δ of defeasible rules. Example 2 below shows a *DeLP*-program that represents information of the application domain introduced in Example 1 and will be used as a running example in the rest of the paper.

Example 2. Let $\mathcal{P}_l = (\Pi_l, \Delta_l)$ be a *DeLP*-program where:

$$\Pi_l = \left\{ \begin{array}{l} mStops \\ mHDriving \\ atNight \\ tJam \leftarrow tSlow \end{array} \right\} \quad \Delta_l = \left\{ \begin{array}{l} suggest(H) \prec good(H), nbHotel(H) \\ suggest(H) \prec sStop, nbHotel(H) \\ \sim suggest(H) \prec dangerZ(H) \\ dangerZ(H) \prec theftZ(H) \\ \sim dangerZ(H) \prec pOfficersZ(H) \\ good(H) \prec stars(H,S), S \geqslant 3 \\ \sim good(H) \prec stars(H,S), S < 3 \\ \sim sStop \prec mStops \\ sStop \prec mHDriving \\ sStop \prec atNight \end{array} \right\}$$

Observe that the set Π_l has three facts and one strict rule. These facts represent information about the current situation that can be automatically obtained during travel: the driver has been driving for many hours (mHDriving), during the night (atNight), and the driver has stopped several times (mStops). The strict rule represents that "a traffic jam exists when the traffic is very slow."

The set Δ_l contains several defeasible rules. The first two defeasible rules represent two tentative reasons to suggest a hotel *H*: "if *H* is a good, nearby hotel" or "if *H* is a nearby hotel and there is a reason to suggest to stop". The last three rules represents reasons for and against suggesting a stop: being many hours driving or driving at night are both reasons for suggesting a stop, whereas having done several stops before is a reason against suggesting a stop. Observe that the sixth and seventh defeasible rules are used for establishing whether H is a good hotel. The third rule represents that "if a hotel *H* is in a dangerous zone then there exists a tentative reason for not suggesting *H*". Finally, the fourth and fifth rules can be read as follows: "the hotel is in a dangerous zone if it is in a zone with many reported thefts", and "the hotel is not in a dangerous zone if it is a zone with police officers."

With the use of strict and defeasible rules it is possible to derive other literals from a *DeLP*-program $\mathcal{P}$. A *defeasible derivation* of a literal L from $\mathcal{P}$ is a finite sequence of ground literals $L_1, L_2, \ldots, L_n = L$, where each literal L_i is in the sequence because: (a) L_i is a fact in $\mathcal{P}$, or (b) there exists a rule R_i in $\mathcal{P}$ (strict or defeasible) with head L_i and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element L_j of the sequence appearing before L_i ($j < i$). We will say that L has a *strict derivation* from $\mathcal{P}$ if either L is a fact or all the rules used for obtaining the sequence $L_1, L_2, \ldots, L_n$ are strict rules. For instance, from the program $\mathcal{P}_l$ of Example 2 there are defeasible derivations for *sStop* and $\sim$*sStop*, and a strict derivation for *atNight*.

Strong negation is allowed in the head of program rules, and hence, may be used to represent contradictory knowledge; two literals are contradictory if they are complementary. Given a literal L, the complement with respect to the strong negation will be noted $\overline{L}$, i.e., $\overline{L}$ is $\sim L$, and $\overline{\sim L}$ is L. Thus, a set is contradictory iff a pair of complementary literals can be obtained from this set. In *DeLP*, $\Pi \cup \Delta$ can be contradictory (*e.g., sStop* and $\sim$ *sStop* are derived from $\mathcal{P}_l$). However, it will be assumed that the set Π is non-contradictory, i.e., in a valid *DeLP*-program there cannot be strict derivations for contradictory literals.

Since contradictory literals can be defeasibly derived, *DeLP* builds arguments supporting tentative conclusions and then, this arguments can be compared to decide which conclusion prevails. Hence, argument comparison criteria play a central role in this kind of formalism.

In *DeLP*, an *argument* for a literal L from a program (Π,Δ) is denoted $\langle \mathcal{A}, L\rangle$, where $\mathcal{A} \subseteq \Delta$ is a minimal and non-contradictory set, such that together with Π allows a defeasible derivation of L. Given a *DeLP*-argument $\langle \mathcal{A}, L\rangle$, L is called the *conclusion* of the argument, and sometimes for simplicity we will say that $\mathcal{A}$ is the argument that *supports* L. For instance, the following two arguments can be constructed from the program $\mathcal{P}_l$ of Example 2: $\langle \mathcal{A}_3, sStop\rangle$ and $\langle \mathcal{A}_4, \sim sStop\rangle$ where $\mathcal{A}_3 = \{\text{sStop} \prec \text{mHDriving}\}$ and $\mathcal{A}_4 = \{\sim \text{sStop} \prec \text{mStops}\}$. Observe that $\mathcal{A}_3$ represents a reason for suggesting a stop (the driver has been many hours driving), whereas $\mathcal{A}_4$ represents a reason against suggesting a stop (the driver has stopped many times during the trip). An argument $\langle \mathcal{A}, L\rangle$ is said to be a subargument of $\langle \mathcal{A}_1, L_1\rangle$, if $\mathcal{A} \subseteq \mathcal{A}_1$.

Two literals L and L_1 *disagree* regarding a program (Π, Δ), when the set $\Pi \cup \{L, L_1\}$ is contradictory. We say that the argument $\langle \mathcal{A}_1, L_1\rangle$ *counter-argues* 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 disagree. For instance, the two arguments introduced above: $\langle \mathcal{A}_3, sStop\rangle$ and $\langle \mathcal{A}_4, \sim sStop\rangle$, attack each other because literals *sStop* and $\sim$ *sStop* disagree. Note that in *DeLP* an argument can attack the conclusion or an inner point of other argument: consider $nbHotel(h1)$ is added to $\mathcal{P}_l$ then $\mathcal{A}_4$ will be a counterargument for $\langle \mathcal{A}_{10}, suggest(h1)\rangle$ where $\mathcal{A}_{10} = \{suggest(h1) - \prec sStop, nbHotel(h1), sStop \prec mHDriving\}$. In Section 5 an application example that considers inner attacks will be given.

In *DeLP*, given an argument that attacks another (*e.g.,* $\mathcal{A}_3$ and $\mathcal{A}_4$), in order to decide which one prevails, the arguments are compared using an *argument preference criterion*. We will denote that an argument $\mathcal{A}_1$ is preferred to $\mathcal{A}_2$, as $\mathcal{A}_1 \succ \mathcal{A}_2$. In *DeLP* the notion of *defeat* is considered as an attack that is effective, i.e., an argument $\mathcal{A}_1$ is a defeater for an argument $\mathcal{A}_2$ if $\mathcal{A}_1 \succ \mathcal{A}_2$ (called proper defeater) or both arguments are incomparable (called blocking defeater). As explained above, in *DeLP* (and hence in *DeLP*-servers) arguments are compared using a fixed and predefined preference criterion.

In the next sections we will propose a new approach that will allow to program how to select dynamically which argument preference criterion to use. In Section 3 we will propose an extension of *DeLP*-servers (called Conditional-preference based Reasoning Server) that will allow to store a repository of argument preference criteria. We will also introduce a special type of contextual query that will allow to specify how the server should select a criterion for such repository. In *DeLP* the answer for a query Q can be: YES if the Q is warranted; NO if an argument against Q is warranted, UNDECIDED if neither arguments for nor against Q can be warranted, and UNKNOWN if the query includes literals that are not in the program's language. In Section 5 we will explain in details the notion of warranted adapted to our new formalization and how answers are computed in our proposed formalism, and then in Sections 5.2 and 5.3 we will show an application examples.

3. Conditional-preference based Reasoning Server

In this section, we propose a Conditional-preference based Reasoning Server (CRS-server for short) that will answer queries using an argumentative inference mechanism, and it will allow a context-adaptable selection of the argument preference criterion. In our proposed approach, a CRS-server will answer queries selecting one preference criterion that will be indicated by the client agent using a special type of expression in the submitted query. CRS-servers will be defined as an extension of *DeLP*-servers. As described above, a *DeLP*-server has three components: $\langle \mathbf{I}, \mathbb{O}, \mathcal{P} \rangle$: a *DeLP*-interpreter $\mathbf{I}$, a set of operators $\mathbb{O}$ used for the handling of the received contextual information and the public knowledge stored in the server as a *DeLP*-program $\mathcal{P}$. In our proposal a CRS-server will have four components $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K} \rangle$, where K (formally introduced below) will be used for storing all the argument preference criteria that will be available on that CRS-server. Then, the notion of contextual query will be extended to include a special kind of expression that will be used to automatically select one argument preference criterion from K for answering that specific query.

Several argument preference criteria have been proposed in the literature of structured argumentation, for instance: evidence based (Tang, Hang, Parsons, & Singh, 2012), literals based (Ferretti, Errecalde, García, & Simari, 2008), presumptions based (Martinez, García, & Simari, 2012; Deagustini et al., 2012), rule's priorities (García & Simari, 2004; Prakken & Sartor, 1997), generalized specificity (García & Simari, 2004). Some of these criteria require additional domain information to compare arguments. Consider for instance, the criterion called rule's priorities introduced in García and Simari (2004). This criterion requires a partial order defined over defeasible rules (denoted "$\rhd$"). The intuitive meaning of "$r_1 \rhd r_2$" is that rule r_1 is deemed better than r_2 under some criterion (*e.g.,* driver's safety) considering the application domain. Then, given two arguments $\mathcal{A}$ and $\mathcal{B}$, $\mathcal{A}$ is preferred over $\mathcal{B}$ with respect to $\rhd$, if there exist at least a rule $r_a \in \mathcal{A}$ and a rule $r_b \in \mathcal{B}$ such that $r_a \rhd r_b$, and there is no rule $r \in \mathcal{B}$ such that $r \rhd r'$ for any $r' \in \mathcal{A}$. Hence, if in an application like the one introduced in Example 1, one argument comparison criterion is implemented with rule's priorities (*e.g.,* driver's safety), then a partial order $\rhd_S$ over defeasible rules (with respect to safety conditions) should be provided and associated with this criterion. If the application has another argument comparison criterion (*e.g.,* driver's comfort) then another type of information may be associated.

Therefore, in a CRS-server, we propose to have a repository that will be used to associate each available criterion with the specific information needed by it. We assume a finite set of criterion identifiers $\mathbb{N}$ and then, we can denote each available preference criterion as $\succ_x$, where $x \in \mathbb{N}$.

Definition 1 (*Repository set*). Let $\mathbb{N}$ be a set of criterion identifiers. A repository set is a finite set $\mathrm{K} = \{\mathcal{R}_1, \mathcal{R}_2 \ldots \mathcal{R}_n\}$ where each $\mathcal{R}_j (1 \leqslant j \leqslant n)$ is a pair $\langle Ic_j, S_j \rangle$ such that $Ic_j \in \mathbb{N}$ and S_j stores all the specific information needed by Ic_j. There cannot be two elements $\langle Ic_i, S_i \rangle$ and $\langle Ic_j, S_j \rangle$ in K such that $Ic_i = Ic_j$, and the set $\mathbb{N}_{\mathrm{K}} \subseteq \mathbb{N}$ represents all the criterion identifiers in the repository set K.

Example 3. Consider the application domain introduced in Example 1, where two argument preference criteria are considered: driver's safety and driver's comfort. Then we can define the following repository set $\mathrm{K}_l = \{\langle sec, S_{sec} \rangle, \langle comf, S_{comf} \rangle\}$ where there are two criterion identifiers $\mathbb{N}_{\mathrm{K}_l} = \{sec, comf\}$ that are respectively associated with the following sets:

$$S_{sec} = \left\{ \begin{array}{l} (\sim suggest(H) \mathrel{-\!\!\prec} dangerZ(H)) \rhd_S (suggest(H) \mathrel{-\!\!\prec} good(H), nbHotel(H)) \\ (\sim suggest(H) \mathrel{-\!\!\prec} dangerZ(H)) \rhd_S (suggest(H) \mathrel{-\!\!\prec} sStop, nbHotel(H)) \\ (\sim sStop \mathrel{-\!\!\prec} mStops) \rhd_S (sStop \mathrel{-\!\!\prec} mHDriving) \\ (sStop \mathrel{-\!\!\prec} atNight) \rhd_S (\sim sStop \mathrel{-\!\!\prec} mStops) \\ (dangerZ(H) \mathrel{-\!\!\prec} theftZ(H)) \rhd_S (\sim dangerZ(H) \mathrel{-\!\!\prec} pOfficersZ(H)) \end{array} \right\}$$

$$S_{comf} = \left\{ \begin{array}{l} (suggest(H) \mathrel{-\!\!\prec} good(H), nbHotel(H)) \rhd_C (\sim suggest(H) \mathrel{-\!\!\prec} dangerZ(H)) \\ (suggest(H) \mathrel{-\!\!\prec} sStop, nbHotel(H)) \rhd_C (\sim suggest(H) \mathrel{-\!\!\prec} dangerZ(H)) \\ (\sim sStop \mathrel{-\!\!\prec} mStops) \rhd_C (sStop \mathrel{-\!\!\prec} mHDriving) \end{array} \right\}$$

Then, a CRS-server with the repository K_l introduced in Example 3 will offer to select and use two available argument comparison criteria: $\succ_{sec}$ and $\succ_{comf}$. The argument preference criterion $\succ_{sec}$ will prefer arguments containing information that favors the security of the driver or the vehicle, whereas $\succ_{comf}$ will prefer arguments containing information that promotes the comfort of driver. For the examples of this paper both will be implemented with rule's priorities but each one with different priorities among rules: S_{comf} and S_{sec}.

One of our goals is to allow CRS-servers to select dynamically a criterion depending on certain conditions; to this end, guards would offer a special way of associating conditions to client preferences. A guard could be viewed as a way of guiding the choice of a criterion depending on the user's preference over certain information stored on server. For us a *guard* will be a set of literals that should be satisfied by a given *DeLP*-program.

Definition 2. A set of literals $\mathcal{G}$ that we will call guard, is satisfied by a *DeLP*-program $\mathcal{P}$ iff for each literal $L \in \mathcal{G}$ there exists a strict derivation from $\mathcal{P}$.

Remark 1. The guard $\emptyset$ is satisfied by any *DeLP*-program.

Example 4. Consider the guards $\mathcal{G}_1 = \{e, f, \sim p\}, \mathcal{G}_2 = \{a, \sim p\}$, $\mathcal{G}_3 = \{b\}$ and $\mathcal{G}_4 = \emptyset$; and the *DeLP*-program $\mathcal{P}_4 = (\Pi_4, \Delta_4)$ with the following sets of rules: $\Pi_4 = \{(a \leftarrow b), (f \leftarrow h), (\sim p \leftarrow h, e), d, e, h\}$, and $\Delta_4 = \{(b \mathrel{-\!\!\prec} d), (\sim b \mathrel{-\!\!\prec} d, e)\}$. The guard $\mathcal{G}_1$ is satisfied by $\mathcal{P}_4$ because all the literals in $\mathcal{G}_1$ have strict derivations from $\mathcal{P}_4$. Since $\emptyset$ can be always satisfied by any *DeLP*-program; then, $\mathcal{G}_4$ is satisfied by $\mathcal{P}_4$. However, $\mathcal{G}_2$ and $\mathcal{G}_3$ are not satisfied by $\mathcal{P}_4$.

The rationale for using strict derivations for determining whether a guard is satisfied by a *DeLP*-program, is that there is no need for an auxiliary criterion to decide between conflicting information since Π is non-contradictory; clearly, restricting the guards to use only strict derivations is a design choice. Other alternatives are already being explored and, given the complexities, they will be subject of a future publication.

In our approach, a server will answer a query considering the preference criteria indicated by the client agent in a conditional-preference expression, or *cp-exp* for short. Thus, as we will show later, every client query will include a *cp-exp* and, in that manner, the server will be able to determine the criterion used to answer the corresponding query. A *cp-exp* will be either a criterion identifier or a guard $\mathcal{G}$ followed by two conditional-preference expressions as formalized below.

Definition 3 (*Conditional-preference expression*). Let K be a repository set and *Ic* a criterion identifier in $\mathbb{N}_K$. Given a set of literals $\mathcal{G}$. A *cp-exp* $\mathcal{E}$ is a finite sequence of symbols recursively defined as:

$$\mathcal{E} = \begin{cases} Ic, or \\ [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2] \text{ where } \mathcal{E}_1 \text{ and } \mathcal{E}_2 \text{ are cp-exps.} \end{cases}$$

The set of all possible *cp-exps* will be denoted $\mathbb{E}$. A *cp-exp* $\mathcal{E}$ will be interpreted in the following way: if $\mathcal{E}$ is a criterion identified Ic, then the criterion $\succ_{Ic}$ is applied, whereas if $\mathcal{E}$ is $[\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2]$ and $\mathcal{G}$ is satisfied by a DeLP-program $\mathcal{P}$, then $\mathcal{E}_1$ is evaluated, otherwise $\mathcal{E}_2$ is evaluated. This last evaluation sequence is applied recursively until a criterion identifier is found. This intuitive idea of how a *cp-exp* $\mathcal{E}$ is evaluated will be captured by the function *cond*, which is formalized below.

Definition 4 (*Condition Evaluation Function*). Let $\mathbb{E}$ be the set of all possible $cp-exps$, $\mathbb{P}$ the set of valid *DeLP*-programs, and $\mathbb{N}_K$ a set of criterion identifiers from a repository set K. The function $cond : \mathbb{E} \times \mathbb{P} \longrightarrow \mathbb{N}_K$, is defined as:

$$cond(\mathcal{E},\mathcal{P}) = \begin{cases} Ic \text{ if } \mathcal{E} = Ic, \text{ or} \\ cond(\mathcal{E}_1,\mathcal{P}) \text{ if } \mathcal{E} = [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2] \text{ and } \mathcal{G} \text{ is satisfied by } \mathcal{P}, \text{ or} \\ cond(\mathcal{E}_2,\mathcal{P}) \text{ if } \mathcal{E} = [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2] \text{ and } \mathcal{G} \text{ is not satisfied by } \mathcal{P} \end{cases}$$

Using the *cp-exp* the client will able to program which argument preference criterion the server will select in each particular situation. The way in which we have modeled the function for evaluation these expressions provides a context-adaptable selection mechanism, that will select a preference criterion depending of which guards from the expression are satisfied. As we will show below to establish which guards are satisfied the server will consider both the public knowledge stored in the server and the private knowledge provided by the client.

Example 5. Consider the criterion identifiers $\mathbb{N}_{K_l} = \{sec, comf\}$ introduced in Example 3. The criterion to establish whether the hotel "*h*1" is recommended may be obtained by means of some of the following expressions:

$$\mathcal{E}_1 = [\{pOfficersZ(h1), stars(h1,5)\} : comf, sec]$$

$$\mathcal{E}_2 = [\{theftZ(h1), tJam\} : sec, [\{stars(h1,5)\} : comf, sec]]$$

The expression $\mathcal{E}_1$ can be read as follows: "If there is a strict derivation for *pOfficersZ*(*h*1) and *stars*(*h*1, 5) then use the criterion which favors driver's comfort, otherwise use the criterion which favors driver's security". The expression $\mathcal{E}_2$ can be interpreted in the following way: "If there is a strict derivation for *theftZ*(*h*1) and *tJam*, then use the security criterion, otherwise the expression $[\{stars(h1,5)\} : comf, sec]$ should be evaluated". Consider that the set $\mathcal{P}_c = \{nbHotel(h1), stars(h1,5), theftZ(h1)\}$ is added to the *DeLP*-program of Example 2, and lets $\mathcal{P}'_l = \mathcal{P}_l \cup \mathcal{P}_c$. For $\mathcal{E}_1$, the guard $\{pOfficersZ(h1), stars(h1,5)\}$ is not satisfied by $\mathcal{P}'_l$, since $pOfficersZ(h1)$ is not strictly derived from $\mathcal{P}'_l$; thus $cond(\mathcal{E}_1, \mathcal{P}'_l) = sec$. For $\mathcal{E}_2$, the guard $\{theftZ(h1), tJam\}$ is not satisfied by the program, since *tJam* does not have a strict derivation from $\mathcal{P}'_l$; then we should evaluate the subexpression $[\{stars(h1,5)\} : comf, sec]$. For this subexpression the guard $\{stars(h1,5)\}$ is satisfied by $\mathcal{P}'_l$, since $stars(h1,5)$ has a strict derivation from $\mathcal{P}'_l$ (it is a fact of that program); thus we have that $cond(\mathcal{E}_2, \mathcal{P}'_l) = comf$ holds.

As previously mentioned the CRS-server will answer client queries containing a *cp-exp*. These queries, denoted as *conditional-preference based query*, are an extension of the contextual queries from the *DeLP*-Servers. Thus, besides a *cp-exp*, these queries will be characterized by a context given by the clients private knowledge and a literal with the *DeLP*-query. We formalize this notion as follows:

Definition 5 (*Conditional-preference based query*). Given a *DeLP* program $\mathcal{P} = (\Pi, \Delta)$, a conditional-preference based query to $\mathcal{P}$ is a tuple $[\mathcal{C}o, \mathcal{E}, Q]$, such that where $\mathcal{C}o$ is a set of ground literals that is non-contradictory with Π, called the context, $\mathcal{E}$ is a *cp-exp*, called the selection criterion, and Q is a literal that represents the *DeLP*-query.

Example 6. Consider for instance the *cp-exp* introduced above in Example 5 $\mathcal{E}_1 = [\{pOfficersZ(h1), stars(h1,5)\} : comf, sec]$.

Then, the following conditional-preference based query can be formulated:

$$CQ_1 = [\mathcal{P}_c, \mathcal{E}_1, suggest(h1)],$$

where $\mathcal{P}_c = \{nbHotel(h1), stars(h1,5), theftZ(h1)\}$ is a set of literals representing agent's private contextual information related to the hotel current situation. With CQ_1 the client agent wants to know the answer for the *DeLP*-query "*suggest*(*h*1)", considering that "*h1*" is a nearby hotel, it is a five stars hotel, and the hotel zone has many theft cases. Note that in CQ_1 the agent proposes $\mathcal{E}_1$ to be used for the selection of the argument comparison criterion to be used by the server is being consulted.

A different conditional-preference based query can be formulated with the same context but different *cp-exp*, for instance:

$$CQ_2 = [\mathcal{P}_c, \mathcal{E}_2, suggest(h1)]$$

where $\mathcal{E}_2 = [\{theftZ(h1), tJam\} : sec, [\{stars(h1,5)\} : comf, sec]]$ is the *cp-exp* introduced in Example 5. Observe that CQ_1 and CQ_2 have the same *DeLP*-query and context, but the proposed selection criterion is different, therefore the answer could be different.

The CRS-servers, like the *DeLP*-Servers, will answer an incoming query combining the contextual information with the stored program using an available operator stored in $\mathbb{O}$. In particular, in our proposal we will use the "+" operator defined in García et al. (2007), which in case of a conflict gives priority to the information received in the query. That is, given a *DeLP*-program $\mathcal{P} = (\Pi, \Delta)$ and a non contradictory set of literals $\mathcal{C}o$ received as the context of the query, then: $\mathcal{P} + \mathcal{C}o = (((\Pi \setminus R) \cup \mathcal{C}o), \Delta)$ where $R = \{\overline{L} : L \in \mathcal{C}o\}$. As we mentioned, the detailed study of such operators is out of the scope of this article (for details refer to García et al. (2007) and García & Simari (2014)).

For instance, if we combine the context $\mathcal{P}_c$ of the queries from Example 6 with the program $\mathcal{P}_l = (\Pi_l, \Delta_l)$ of Example 2 using the "+" operator, we will have as a resulting program just $(\Pi_l \cup \mathcal{P}_c, \Delta_l)$ since there is no fact of Π_l that is in contradiction with a literal of $\mathcal{P}_c$. This resulting program is the one that the server will use to evaluate the *cp-exp* in the query. Then, using that program and the selected preference criterion from the expression evaluation, the server will invoke the *DeLP*-interpreter to answer the query. Next, we present the structure of the *DeLP*-interpreters used in the CRS-servers, and after that we formalize the intuitions of how our server answers a query, recalling that a valid *DeLP*-program is such that there cannot be strict derivations for contradictory literals.

Definition 6 (*DeLP-interpreter*). Let $\mathbb{P}$ be the set of valid *DeLP*-programs, $\mathbb{N}_K$ the set of criterion identifiers from a repository set K and $\mathbb{Q}$ the set of possible *DeLP*-queries. A *DeLP*-interpreter is a function $\mathbf{I} : \mathbb{P} \times \mathbb{N}_K \times \mathbb{Q} \longrightarrow \mathbb{R}$, where $\mathbb{R}$ is the set of *DeLP* answers.

Having introduced every component, now we will formally define the CRS-servers.

Definition 7 (*Conditional-preference based Reasoning Server*). A conditional-preference based reasoning server is a 4-tuple $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K} \rangle$, where $\mathbf{I}$ is a *DeLP*-interpreter, $\mathbb{O}$ is a set of *DeLP*-operators, $\mathcal{P}$ is a *DeLP*-program, and K is a repository set.

Next, we will formalize the intuitions on how the server handles an incoming query and how it invokes the interpreter to answer that query.

Definition 8 (*Answer to a query*). Let $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K} \rangle$ *be a CRS*-server, CQ $= [\mathcal{C}o, \mathcal{E}, Q]$ a conditional-preference based query for CRS-server, $\mathcal{P}'$ a program modified with contextual information $\mathcal{C}o$, and $cond(\mathcal{E}, \mathcal{P}') = Ic$ a criterion identifier in K obtained from evaluating the expression $\mathcal{E}$. An answer for CQ from CRS-server, denoted *Ans*(CRS-server, CQ), corresponds to the result of the function $\mathbf{I}(\mathcal{P}', Ic, Q)$.

Fig. 1 depicts a situation with a CRS-server and a conditional-preference based query $[\mathcal{C}o, \mathcal{E}, Q]$ sent by a client agent. Operators from $\mathbb{O}$ handle the integration of the contextual information $\mathcal{C}o$ with the program stored $\mathcal{P}$, generating a new program $\mathcal{P}'$. Then, given the program $\mathcal{P}'$ and the *cp-exp* $\mathcal{E}$, a criterion identifier *Ic* is obtained with the function $cond(\mathcal{E}, \mathcal{P}')$. This criterion identifier *Ic* determines the argument preference criterion, and consequently, consulting the repository K obtaining the pair $\langle Ic, S \rangle$, and from there, the specific information S to be used in the comparison of arguments. Thus, the *DeLP*-interpreter $\mathbf{I}$ takes the program $\mathcal{P}'$, the identifier *Ic*, and the queried literal *Q* as input, and finally returns the computed answer *Ans* for *Q*.

In Section 5 we will explain in detail how answers are computed in our proposed formalism and we will develop an application example. Nevertheless, first, in the following section, we will study some properties of *cp-exp*.

4. Evaluation of conditional-preference expressions

We will perform more detailed analysis of the properties of *cp-exp* and its evaluation semantics considering relevant characteristics that would lead to their optimization. To facilitate this task we will characterize *cp-exp* using a tree representation.

4.1. Tree representation

Conditional-preference expressions can be represented through a full binary tree where every inner node is labeled with a guard and each leaf node is labeled with a criterion identifier. A special case of this type of binary tree corresponds to the representation of the most simple cp-exp ($\mathcal{E}$ = Ic) where the root is also a leaf node that is labeled with a criterion identifier.

Definition 9 (*Tree representation*). Given a set of comparison criterion identifiers $\mathbb{N}$ and a conditional-preference expression $\mathcal{E}$, a tree representation for $\mathcal{E}$, denoted $\mathrm{T}_{\mathcal{E}}$, is a full binary tree defined recursively as follows:

1. If $\mathcal{E} = Ic (Ic \in \mathbb{N})$ then $\mathrm{T}_{\mathcal{E}}$ contains only one node labeled with Ic.
2. If $\mathcal{E} = [\mathcal{G} : \mathcal{E}_i; \mathcal{E}_j]$, then the tree representation $\mathrm{T}_{\mathcal{E}}$ is a full binary tree where:
   –the root is labeled with the guard $\mathcal{G}$,
   –the left child of the root is the tree representation for $\mathcal{E}_i$, and
   –the right child of the root is the tree representation for $\mathcal{E}_j$.

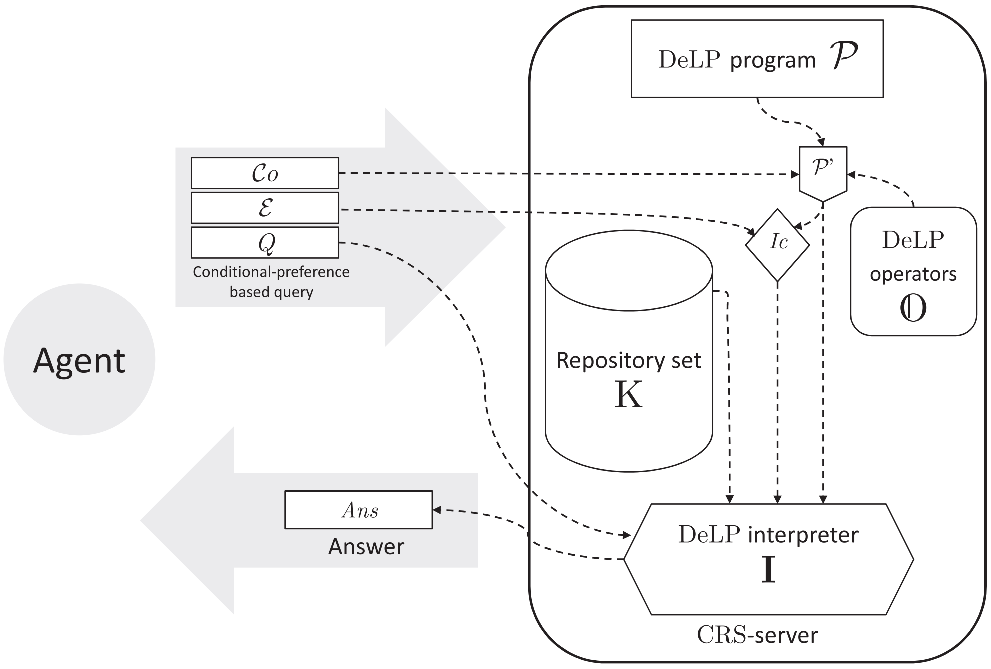

Fig. 1. Main components of a conditional-preference based query and CRS-server.

As will be show below the tree representation of a *cp-exp* will be useful for showing some formal results. The following proposition shows that for any *cp-exp* there exist a unique tree representation. This proposition will be also used for the proof of an important result at the end of this section.

Proposition 1. *There exists an unique tree representation* $\mathsf{T}_{\mathcal{E}}$ *for a cp-exp* $\mathcal{E}$.

Proof. Straightforward from Definition 9. □

Fig. 2 shows the tree representation for the *cp-exps* $\mathcal{E}_1$ and $\mathcal{E}_2$ introduced in Example 5. The next example shows two *cp-exps* with a greater level of nested conditions that will be used in the rest of the section.

Example 7. Consider the following conditional-preference expressions:

$\mathcal{E}_a = Ic_1$

$\mathcal{E}_b = [\{f, e, d\} : [\{\sim p, \sim a\} : [\{m, a\} : Ic_1; Ic_2]; Ic_3]; [\{\sim p, h\} : Ic_1; Ic_3]]$

$\mathcal{E}_c = [\{f, e\} : [\{h, \sim p\} : [\{d\} : Ic_1; Ic_2]; [\{\sim p, \sim e\} : Ic_3; Ic_2]]; [\{\sim p, h\} : [\{e, \sim p, \sim f\} : Ic_3; Ic_2]; [\{d\} : Ic_1; Ic_2]]]$

Definition 10 (*Annotated path/criterion selection structure*). Let $\mathsf{T}_{\mathcal{E}}$ be the tree representation for a *cp-exp* $\mathcal{E}$. Let *Ic* be the label of a leaf *L* in $\mathsf{T}_{\mathcal{E}}$ and $[\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n] (n \geqslant 1)$, the sequence of labels (guards) of inner nodes from the root of $\mathsf{T}_{\mathcal{E}}$ labeled $\mathcal{G}_1$ to the leaf *L*. An annotated path for *L* in $\mathsf{T}_{\mathcal{E}}$ is a tuple denoted $\langle \Gamma, Ic \rangle_{\mathsf{T}_{\mathcal{E}}}$ such that $\Gamma = [\mathcal{G}_1^{x_1}, \mathcal{G}_2^{x_2}, \ldots \mathcal{G}_n^{x_n}] (n \geqslant 1)$, and where:

(a) for each $\mathcal{G}_i$ $(1 \leqslant i \leqslant n-1)$ the annotation x_i is "+" if $\mathcal{G}_{i+1}$ is the left child of $\mathcal{G}_i$ in $\mathsf{T}_{\mathcal{E}}$, or "-" otherwise, and
(b) the annotation x_n is "+" if *Ic* is the left child of $\mathcal{G}_n$ in $\mathsf{T}_{\mathcal{E}}$, or "-" otherwise.

We will also say that, a criterion selection structure for the criterion identifier *Ic* in $\mathsf{T}_{\mathcal{E}}$, extracted from the annotated path $\langle \Gamma, Ic \rangle_{\mathsf{T}_{\mathcal{E}}}$, is a triplet $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ such that: $\Gamma^+ = \{\mathcal{G}_i | \mathcal{G}_i^+ \in \Gamma, 1 \leqslant i \leqslant n\}$, and $\Gamma^- = \{\mathcal{G}_j | \mathcal{G}_j^- \in \Gamma, 1 \leqslant j \leqslant n\}$.

Example 8. Given the tree representation $\mathsf{T}_{\mathcal{E}_b}$ in Fig. 3, in following table we show each annotated path that can be obtained from $\mathsf{T}_{\mathcal{E}_b}$ together with its respective criterion selection structure:

Annotated path	Criterion selection structure
$\langle [\{f, e, q\}^+, \{\sim p, \sim a\}^+, \{m, a\}^+], Ic_1 \rangle_{\mathsf{T}_{\mathcal{E}_b}}$	$\mathsf{C}_1 = (\{\{f, e, q\}, \{\sim p, \sim a\}, \{m, a\}\}, \{\}, Ic_1)_{\mathsf{T}_{\mathcal{E}_b}}$
$\langle [\{f, e, q\}^+, \{\sim p, \sim a\}^+, \{m, a\}^-], Ic_2 \rangle_{\mathsf{T}_{\mathcal{E}_b}}$	$\mathsf{C}_2 = (\{\{f, e, q\}, \{\sim p, \sim a\}\}, \{\{m, a\}\}, Ic_2)_{\mathsf{T}_{\mathcal{E}_b}}$
$\langle [\{f, e, q\}^+, \{\sim p, \sim a\}^-], Ic_3 \rangle_{\mathsf{T}_{\mathcal{E}_b}}$	$\mathsf{C}_3 = (\{\{f, e, q\}\}, \{\{\sim p, \sim a\}\}, Ic_3)_{\mathsf{T}_{\mathcal{E}_b}}$
$\langle [\{f, e, q\}^-, \{\sim p, h\}^+], Ic_1 \rangle_{\mathsf{T}_{\mathcal{E}_b}}$	$\mathsf{C}_4 = (\{\{\sim p, h\}\}, \{\{f, e, q\}\}, Ic_1)_{\mathsf{T}_{\mathcal{E}_b}}$
$\langle [\{f, e, q\}^-, \{\sim p, h\}^-], Ic_3 \rangle_{\mathsf{T}_{\mathcal{E}_b}}$	$\mathsf{C}_5 = (\{\}, \{\{f, e, q\}, \{\sim p, h\}\}, Ic_3)_{\mathsf{T}_{\mathcal{E}_b}}$

Fig. 3 shows the tree representations $\mathsf{T}_{\mathcal{E}_a}$, $\mathsf{T}_{\mathcal{E}_b}$, and $\mathsf{T}_{\mathcal{E}_c}$ associated to $cp - exps$ $\mathcal{E}_a$, $\mathcal{E}_b$, and $\mathcal{E}_c$. Note that $\mathsf{T}_{\mathcal{E}_a}$ is a tree with only one node, and that in the same tree, different leaves can be labeled with the same criterion identifier.

It is interesting to observe that every path from the root to a leaf in a tree $\mathsf{T}_{\mathcal{E}}$ represents a finite sequence of guards supporting the decision that chooses a preference criterion. The following definition characterize this notion and will be used for the analysis and propositions in the rest of the section.

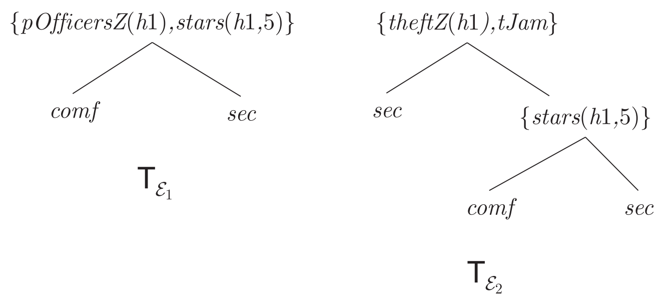
{pOfficersZ(h1),stars(h1,5)}
comf
sec
$\mathsf{T}_{\mathcal{E}_1}$
{theftZ(h1),tJam}
sec
{stars(h1,5)}
comf
sec
$\mathsf{T}_{\mathcal{E}_2}$

Fig. 2. Tree representations for $\mathcal{E}_1$ and $\mathcal{E}_2$ of Example 5.

Given a *DeLP*-program $\mathcal{P}$ and a annotated path $\langle \Gamma, Ic \rangle_{\mathsf{T}_{\mathcal{E}}}$, a guard $\mathcal{G}_i^+$ in the sequence Γ is interpreted as *a guard* $\mathcal{G}_i$ *that must be satisfied by* $\mathcal{P}$ *in order to apply the preference criterion identified with Ic*, whereas with $\mathcal{G}_j^-$, is interpreted as *a guard* $\mathcal{G}_j$ *that must not be satisfied by* $\mathcal{P}$. Thus, if we consider the criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ associated to the given path, the elements of Γ^+ must be satisfied from $\mathcal{P}$ and the elements of Γ^- must not satisfied from $\mathcal{P}$. Then, when a program and a criterion selection structure fulfil these constraints, we will say that the criterion selection structure is conformant regarding to the program.

Definition 11 (*Conformance*). Given a *DeLP*-program $\mathcal{P}$. A criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ is conformant regarding to $\mathcal{P}$ if it holds:

- For all $\mathcal{G} \in \Gamma^+, \mathcal{G}$ is satisfied by $\mathcal{P}$, and
- for all $\mathcal{G} \in \Gamma^-, \mathcal{G}$ is not satisfied by $\mathcal{P}$.

For instance, the structure C_4 of Example 8 is conformant with respect to the program $\mathcal{P}_4$ of Example 4.

Remark 2. A tree $\mathsf{T}_{\mathcal{E}}$ with a single node will contain just a criterion identifier *Ic*; thus, this tree will have a single annotated path $\langle \emptyset, Ic \rangle_{\mathsf{T}_{\mathcal{E}}}$ which will have $(\emptyset, \emptyset, Ic)_{\mathsf{T}_{\mathcal{E}}}$ as its corresponding criterion selection structure and this is conformant with respect to any *DeLP*-program $\mathcal{P}$. For this reason and for the purpose of this section, from now on we will focus the study on trees containing more than one node.

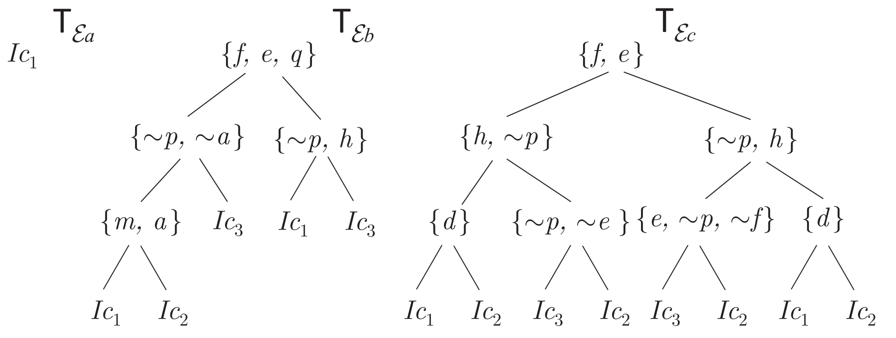

Fig. 3. Tree representations $\mathsf{T}_{\mathcal{E}_a}$, $\mathsf{T}_{\mathcal{E}_b}$, and $\mathsf{T}_{\mathcal{E}_c}$.

The following proposition shows that for any two criterion selection structures in the same tree representation, at least one guard $\mathcal{G}$ is shared by both structures. This result will be used for the proof of the Proposition 3. For instance, the structures C_1 and C_2 of Example 8 share all their guards, whereas C_3 and C_4 share only the guard $\{f, e, q\}$.

Proposition 2. *Let $\mathsf{T}_{\mathcal{E}}$ be a tree representation. Given two different annotated paths $\langle[F_1^{x_1}, F_2^{x_2}, \ldots F_m^{x_m}], Ic_i\rangle_{\mathsf{T}_{\mathcal{E}}}$ and $\langle[H_1^{x_1}, H_2^{x_2}, \ldots H_n^{x_n}], Ic_j\rangle_{\mathsf{T}_{\mathcal{E}}}$ and their associated criterion selection structures $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$. There exists at least a guard $\mathcal{G}$ in both $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$ such that either $\mathcal{G} \in \Gamma_i^+ \cap \Gamma_j^-$ or $\mathcal{G} \in \Gamma_j^+ \cap \Gamma_i^-$.*

Proof. Let $\langle[F_1^{x_1}, F_2^{x_2}, \ldots F_m^{x_m}], Ic_i\rangle_{\mathsf{T}_{\mathcal{E}}}$ and $\langle[H_1^{x_1}, H_2^{x_2}, \ldots H_n^{x_n}], Ic_j\rangle_{\mathsf{T}_{\mathcal{E}}}$ be two different annotated paths. It is clear that they will share a prefix $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$, $k \leqslant m$ *and* $k \leqslant n$, such that $F_i = H_i = \mathcal{G}_i (i \leqslant k)$. Indeed, if there exists only one guard $\mathcal{G}_i$ in the prefix, that is $i = k$ and $k = 1$, then the prefix is the root labeled $\mathcal{G}_i$ of the tree $\mathsf{T}_{\mathcal{E}}$ and $\mathcal{G}_i \in \Gamma_i^+ \cap \Gamma_j^-$ or $\mathcal{G}_i \in \Gamma_j^+ \cap \Gamma_i^-$. In contrast, if there exists more than a guard in the prefix, then the last guard $\mathcal{G}_k$ in $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ is such that $\mathcal{G}_k \in \Gamma_i^+ \cap \Gamma_j^-$ or $\mathcal{G}_k \in \Gamma_j^+ \cap \Gamma_i^-$. □

Remark 3. For any tree representation $\mathsf{T}_{\mathcal{E}}$ there will exist two different criterion selection structures $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$ such that $\Gamma_i^+ = \emptyset$ and $\Gamma_j^- = \emptyset$. Given that $\mathsf{T}_{\mathcal{E}}$ is a full binary tree, then the left-most path of the root of $\mathsf{T}_{\mathcal{E}}$ will be represented by the structure $(\Gamma_i^+, \emptyset, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and the right-most path by $(\emptyset, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$. In Example 8, C_1 and C_5 are the structures in $\mathsf{T}_{\mathcal{E}_b}$ (Fig. 3) where $\Gamma_1^- = \emptyset$ and $\Gamma_5^+ = \emptyset$. Note also that, given the tree $\mathsf{T}_{\mathcal{E}_a}$ of Fig. 3, the only obtained structure is $(\emptyset, \emptyset, Ic_1)_{\mathsf{T}_{\mathcal{E}_a}}$, where both $\Gamma^- = \emptyset$ and $\Gamma^+ = \emptyset$.

The following proposition states that given any tree representation $\mathsf{T}_{\mathcal{E}}$ and a *DeLP*-program $\mathcal{P}$, it is not possible to find two different criterion selection structures in $\mathsf{T}_{\mathcal{E}}$ such that they are conformant w.r.t. $\mathcal{P}$. As we will show further below, this result will be used to establish that the selected preference criterion is obtained from the structure that is conformant w.r.t. $\mathcal{P}$.

Proposition 3. *Given a DeLP-program $\mathcal{P}$ and a cp-exp $\mathcal{E}$ and its tree representation $\mathsf{T}_{\mathcal{E}}$. There exists a unique criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ in $\mathsf{T}_{\mathcal{E}}$ such that $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ is conformant w.r.t. $\mathcal{P}$.*

Proof. Assume $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$ are two different structures in the same tree $\mathsf{T}_{\mathcal{E}}$. By Proposition 2 there exists at least one guard $\mathcal{G}$ in $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$ such that $\mathcal{G} \in \Gamma_i^+ \cap \Gamma_j^-$ or $\mathcal{G} \in \Gamma_j^+ \cap \Gamma_i^-$. Thus, one of the structures would be conformant w.r.t. $\mathcal{P}$ with $\mathcal{G}$ satisfied by $\mathcal{P}$, and the another one with $\mathcal{G}$ not satisfied by $\mathcal{P}$. However, by Definition 2 a guard $\mathcal{G}$ is either satisfied or not satisfied by a given *DeLP*-program $\mathcal{P}$, then $(\Gamma_i^+, \Gamma_i^-, Ic_i)_{\mathsf{T}_{\mathcal{E}}}$ and $(\Gamma_j^+, \Gamma_j^-, Ic_j)_{\mathsf{T}_{\mathcal{E}}}$ can never be conformant w.r.t. the same *DeLP*-program $\mathcal{P}$. □

Consider again the *DeLP*-program $\mathcal{P}_4$ of Example 4 and the trees $\mathsf{T}_{\mathcal{E}_b}$ and $\mathsf{T}_{\mathcal{E}_c}$ in Fig. 3. The structure $(\{\{\sim p, h\}\}, \{\{f, e, q\}\}, Ic_1)_{\mathsf{T}_{\mathcal{E}_b}}$ of $\mathsf{T}_{\mathcal{E}_b}$ is the only one conformant w.r.t. $\mathcal{P}_4$ and the structure $(\{\{f, e\}, \{h, \sim p\}, \{d\}\}, \{\}, Ic_1)_{\mathsf{T}_{\mathcal{E}_c}}$ is the only one conformant w.r.t. $\mathcal{P}_4$.

When analysing whether a structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ is conformant w.r.t. a *DeLP*-program, repeated literals may appear in the guards of the set Γ^+. For instance, when analysing the structure $(\{\{\sim p, h\}, \{e, \sim p, \sim f\}\}, \{\{f, e\}\}, Ic_3)_{\mathsf{T}_{\mathcal{E}_c}}$ of the tree representation $\mathsf{T}_{\mathcal{E}_c}$ in Fig. 3, the literal $\sim p$ is repeated. In this case, this structure will be conformant w.r.t. a *DeLP*-program if $\{\sim p, h\}$ and $\{e, \sim p, \sim f\}$ are satisfied by the program, that is, if there is a strict derivation for every literal in these guards. Thus, in this particular case, we will have to analyse twice if there is a strict derivation for $\sim p$, leading to a redundant computation. In the following definition we will characterize which paths will suffer from this kind of redundancy.

Definition 12 (*Redundancy*). Let $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ be a criterion selection structure. We say that $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ is redundant iff $\bigcap_{\mathcal{G}_i \in \Gamma^+} \mathcal{G}_i \neq \emptyset$.

Note that this redundancy problem could be solved by using the following transformation. Let $\langle\Gamma, Ic\rangle_{\mathsf{T}_{\mathcal{E}}}$ be the annotated path associated to the structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$ where $\Gamma = [\mathcal{G}_1^{x_1}, \ldots, \mathcal{G}_i^{x_i}, \ldots, \mathcal{G}_n^{x_n}] (n \geqslant 0$ and $i \leqslant n)$. We can build a new path $\langle\Gamma', Ic\rangle_{\mathsf{T}_{\mathcal{E}}}$ from $\langle\Gamma, Ic\rangle_{\mathsf{T}_{\mathcal{E}}}$ such that there exists not repeated literals in the guards $\mathcal{G}_i'^+ \in \Gamma'$, that is

$$\mathcal{G}_i'^+ = \mathcal{G}_i'^+ \setminus \bigcap_{\mathcal{G}_j'^+ \in \Gamma', j \leqslant i} \mathcal{G}_j'^+.$$

Then, the criterion selection structure $(\Gamma'^+, \Gamma'^-, Ic)_{\mathsf{T}_{\mathcal{E}}}$, obtained from the path $\langle\Gamma', Ic\rangle_{\mathsf{T}_{\mathcal{E}}}$, is not redundant. For instance, as shown previously, the structure $\langle\{\{\sim p, h\}, \{e, \sim p, \sim f\}\}, \{\{f, e\}\}, Ic_3\rangle_{\mathsf{T}_{\mathcal{E}_c}}$ is redundant. Thus, after the transformation just described, the new structure $\langle\{\{\sim p, h\}, \{e, \sim f\}\}, \{\{f, e\}\}, Ic_3\rangle_{\mathsf{T}_{\mathcal{E}_c}}$ without repeated literals is obtained. We will assume that every structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ in a tree $\mathsf{T}_\mathcal{E}$ is non redundant.

Next, we continue with the analysis of particular situations that can arise in a *cp-exp*. Propositions 4 and 5 will identify characteristics that make a structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ non conformant w.r.t. any given *DeLP*-program.

Proposition 4. *Given a cp-exp $\mathcal{E}$, its tree representation $\mathsf{T}_\mathcal{E}$, and a program $\mathcal{P}$, let $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ be a criterion selection structure. If $\Gamma^+ \cap \Gamma^- \neq \emptyset$, then $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is not conformant regarding to $\mathcal{P}$.*

Proof. Suppose that $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is conformant regarding $\mathcal{P}$ and $\Gamma^+ \cap \Gamma^- \neq \emptyset$. By Definition 11 the guards of the set Γ^+ would be satisfied and the guards of Γ^- would not be satisfied by $\mathcal{P}$. However, if $\Gamma^+ \cap \Gamma^- \neq \emptyset$, there exists a guard $\mathcal{G}$ such that $\mathcal{G} \in \Gamma^+ \cap \Gamma^-$. Thus, $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ would be not conformant regarding $\mathcal{P}$. □

Note that, a guard $\mathcal{G}$ may contain contradictory literals. the interesting case to analyse is when contradictory literals appear in the set Γ^+ of a structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$. The following proposition shows that if there exists contradictory literals in Γ^+, then at least one guard in Γ^+ will not be satisfied by any given *DeLP*-program $\mathcal{P}$. Thus, $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ will not be conformant w.r.t. $\mathcal{P}$.

Proposition 5. *Given a criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ and a DeLP-program $\mathcal{P} = (\Pi, \Delta)$. If the set resulting $\mathcal{S} = \bigcup_{\mathcal{G} \in \Gamma^+} \mathcal{G}$ is contradictory, then $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is not conformant regarding to $\mathcal{P}$.*

Proof. Assume that $\mathcal{S}$ is a contradictory set, therefore, there exist strict derivations for two complementary literals $L, \sim L \in \mathcal{S}$ from $\mathcal{P}$. Then Π is contradictory and $\mathcal{P}$ would not be a valid *DeLP*-program. □

For instance, the annotated path $\langle[\{f, e\}^+, \{h, \sim p\}^-, \{\sim p, \sim e\}^+], Ic_3\rangle$ of $\mathsf{T}_{\mathcal{E}_c}$ with its criterion selection structure $(\{\{f, e\}, \{\sim p, \sim e\}\}, \{\{h, \sim p\}\})_{\mathsf{T}_{\mathcal{E}_c}}$. Note that, there is no *DeLP*-program capable of satisfying both $\{f, e\}$ and $\{\sim p, \sim e\}$, since no valid program can have strict derivations for both e and $\sim e$.

We will consider as sound criterion selection structures those structures that do not suffer from the problems characterized in Propositions 4 and 5. This notion is formalized in the following definition.

Definition 13 (*Sound criterion selection structure/tree representation*). Let $\mathsf{T}_\mathcal{E}$ be a tree representation and $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ a criterion selection structure. The structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is a sound criterion selection structure iff:

(i) Γ^+ and Γ^- are disjoint sets.
(ii) The set Γ^+ is non contradictory.

A sound tree representation $\mathsf{T}_\mathcal{E}$ contains only valid criterion selection structures.

Definition 14. Let $\mathcal{E}$ be a *cp-exp* and $\mathsf{T}_\mathcal{E}$ its associated tree representation. We say that the expression $\mathcal{E}$ is valid iff the tree $\mathsf{T}_\mathcal{E}$ is sound.

Observe that in Fig. 3 not all trees are sound tree representations. For instance, $\mathsf{T}_{\mathcal{E}_c}$ is not sound since, as we previously discussed, in the structure $\langle\{\{f, e\}, \{\sim p, \sim e\}\}, \{\{\sim p, h\}\}, Ic_3\rangle_{\mathsf{T}_{\mathcal{E}_c}}$, the set $\{f, e\} \cup \{\sim p, \sim e\}$ is contradictory. On the other hand note that both $\mathsf{T}_{\mathcal{E}_1}$ and $\mathsf{T}_{\mathcal{E}_2}$ in Fig. 2 are sound tree representation.

Proposition 6. *Given a valid cp-exp $\mathcal{E}$ and its tree representation $\mathsf{T}_\mathcal{E}$. For every criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$, there exists a DeLP-program $\mathcal{P}$ such that $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is conformant w.r.t. $\mathcal{P}$.*

Proof. Given that the tree $\mathsf{T}_\mathcal{E}$ is sound and $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ an arbitrary structure in $\mathsf{T}_\mathcal{E}$, then $\bigcup_{\mathcal{G}_i \in \Gamma^+} \mathcal{G}_i$ is a *DeLP*-program that satisfies the condition in the statement. □

Given the tree representation $\mathsf{T}_\mathcal{E}$ associated to the *cp-exp* $\mathcal{E}$, different sequences of guards can possibly lead to different criterion identifiers. Nevertheless, from a *DeLP*-program $\mathcal{P}$, the evaluation result obtained from $cond(\mathcal{E}, \mathcal{P})$ reflects the fact that there exists only one path from the root of the tree $\mathsf{T}_\mathcal{E}$ to the criterion identifier resulting of such evaluation process. For example, considering the expression $\mathcal{E}_2$ of Example 5 and the program $\mathcal{P}_l$ of Example 2 the answer of the function $cond(\mathcal{E}_2, \mathcal{P}_l)$ is the criterion identifier *sec* due to the fact that $theftZ(h1)$, *tJam*, and $stars(h1, 5)$ have not strict derivations. The following lemma shows that it is always possible to obtain a criterion identifier from a sound tree representation.

lemma 1. *Given a DeLP-program $\mathcal{P}$, a cp-exp $\mathcal{E}$ with its tree representation $\mathsf{T}_\mathcal{E}$, and a criterion identifier Ic. It holds that $cond(\mathcal{E}, \mathcal{P}) = Ic$ iff Ic appears in a criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ that is conformant w.r.t. $\mathcal{P}$.*

Proof. If $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ is conformant w.r.t. $\mathcal{P}$, then by Proposition 3 the structure obtained from the tree representation $\mathsf{T}_\mathcal{E}$ using the program $\mathcal{P}$ is $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$. By Proposition 1, $\mathsf{T}_\mathcal{E}$ is the tree associated the expression $\mathcal{E}$, then *Ic* would be the evaluation result obtained from $cond(\mathcal{E}, \mathcal{P})$.

If $cond(\mathcal{E}, \mathcal{P}) = Ic$, then by Definition 4 either $\mathcal{E} = Ic$ or $\mathcal{E} = [\mathcal{G} : \mathcal{E}_i; \mathcal{E}_j]$. If $\mathcal{E} = Ic$, by Definition 9 there exists a tree representation for $\mathcal{E}$ with only one node labeled with *Ic*. Then, by Remark 2 this tree has a single annotated path $\langle\emptyset, Ic\rangle_{\mathsf{T}_\mathcal{E}}$ with the unique respectively associated criterion selection structure $(\emptyset, \emptyset, Ic)_{\mathsf{T}_\mathcal{E}}$. If $\mathcal{E} = [\mathcal{G}: \mathcal{E}_i; \mathcal{E}_j]$, then by Definition 9, $\mathcal{E}$ has an associated tree representation $\mathsf{T}_\mathcal{E}$ such that by Proposition 3 there exists a criterion selection structure $(\Gamma^+, \Gamma^-, Ic)_{\mathsf{T}_\mathcal{E}}$ conformant w.r.t. $\mathcal{P}$ in $\mathsf{T}_\mathcal{E}$. □

Several conclusions can be drawn from the results that we obtained in this section. In general terms, we would like to emphasize the fact that a tree representation not only helps to understand the evaluation semantics of conditional-preference expressions, but also allowed us to discuss several aspects related to the selection of a criterion. For example, it is easy to see that would not be a coherent decision if the user specifies the guards in such a way that to select a criterion it would be required to infer contradictory literals from the strict part of the program. In fact, an important point to mention is that the flexibility of our approach regarding how *cp-exps* are constructed by the user, allowing the possibility of building expressions that could be incoherent or contradictory. Situations of these type were described in the Propositions 4 and 5; to deal with these possible problems and forcing the construction of expressions that satisfy internal coherence is that *cp-exps* in a conditional-preference based query are required to be valid expressions.

5. Computing answers

We will introduce here the warrant procedure carried out by the interpreter of a CRS-server for computing answers. Then, we will exemplify how the CRS-server handles conditional-preference based queries in our running example. Finally, we will show how our proposal can be applied in a mobile robot environment.

5.1. Warranting queries

The notion of *attack* introduced in Section 2 helps to identify a conflict between two arguments; to decide which argument prevails, a preference criterion must be used. We will now formalize the notion of *defeat* which considers the preference criterion selected by the CRS-server. As we will show in the next section, when a CRS-server receives a conditional-preference based query it will use the *cp-exp* for selecting the appropriate preference criterion that will be used for computing the answer of the received query. This selected preference criterion will help in the comparison of arguments as the following definition shows.

Definition 15 (*Defeat*). Let $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K}\rangle$ be a CRS-server, and Ic a criterion identifier in the repository set K. Let $\langle \mathcal{A}_1, L_1\rangle$ and $\langle \mathcal{A}_2, L_2\rangle$ be two arguments from $\mathcal{P}$. We say that $\langle \mathcal{A}_1, L_1\rangle$ defeats $\langle \mathcal{A}_2, L_2\rangle$ according the preference criterion $\succ_{Ic}$, iff there exists a sub-argument $\langle \mathcal{A}, L\rangle$ of $\langle \mathcal{A}_2, L_2\rangle$ such that $\langle \mathcal{A}_1, L_1\rangle$ counter-argues $\langle \mathcal{A}_2, L_2\rangle$ at L and it holds that:

1. $\mathcal{A}_1 \succ_{Ic} \mathcal{A}$ (proper defeater), or
2. $\mathcal{A}_1 \not\succ_{Ic} \mathcal{A}$ and $\mathcal{A} \not\succ_{Ic} \mathcal{A}_1$ (blocking defeater).

As shown in Section 2, the argument $\langle \mathcal{A}_3, sStop\rangle$ attacks $\langle \mathcal{A}_4, \sim sStop\rangle$ at literal $\sim sStop$. Considering the comfort criterion $\succ_{comf}$ and the priorities among rules associated with this criterion as presented in Example 3. With this priorities, $\mathcal{A}_4$ is preferred to $\mathcal{A}_3$ because $(\sim sStop \prec mStops) \rhd_C (sStop \prec mHDriving)$, then $\mathcal{A}_4$ is a proper defeater for $\mathcal{A}_3$, whereas the argument $\langle \mathcal{A}_5, sStop\rangle$ where $\mathcal{A}_5 = \{sStop \prec atNight\}$ is a blocking defeater for $\mathcal{A}_4$, and vice versa.

In *DeLP* (García & Simari, 2004), a queried literal Q is *warranted* from a program $\mathcal{P}$ if there exists a non-defeated argument $\mathcal{A}$ for Q built from $\mathcal{P}$. To establish whether $\mathcal{A}$ is a undefeated argument, all the defeaters for $\mathcal{A}$ are considered, and the defeaters for these defeaters are looked up recursively in a dialectical process. As each defeater could in turn be defeated, a sequence of arguments called *argumentation line* $\Lambda = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n]$ arises where each argument (except the first one) is a defeater of its predecessor. In Λ, in regard to the initial argument the arguments in even positions play a role as supporting arguments and the ones in odd positions act as interfering arguments. In DeLP, an argumentation line Λ is acceptable if the following conditions hold: (1) Λ is finite, (2) the set of supporting arguments in Λ is non contradictory and the set of interfering arguments in Λ is non contradictory, (3) no argument $\mathcal{A}_j$ in Λ is a subargument of an argument $\mathcal{A}_i$ in Λ, $i < j$, and (4) every blocking defeater $\mathcal{A}_i$ in Λ is defeated by a proper defeater $\mathcal{A}_{i+1}$ in Λ. These four constraints are necessary for avoiding fallacious situations (see García & Simari (2004) for a complete discussion). Since there can be more than one defeater for a given argument, a set of argumentation lines is produced, referred as a "bundle" using the terminology of Chesñevar and Simari (2007). A convenient tree structure can be build to consider all the involved argumentation lines together. Next we will formally define this notion considering the preference criteria identifier provided to the interpreter.

Definition 16 (*Dialectical Tree*). Let $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K}\rangle$ be a CRS-server, and Ic a criterion identifier in the repository set K. Let $\mathcal{A}_0$ be an argument for a query Q from the program $\mathcal{P}$. A dialectical tree for $\mathcal{A}_0$ under the preference criterion $\succ_{Ic}$, denoted $\mathcal{T}^{Ic}_{\mathcal{A}_0}$, is defined as follows:

1. The root of the tree is labeled with $\mathcal{A}_0$.
2. Let N be a node on the tree labeled $\mathcal{A}_n$, and $\Lambda = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n]$ the sequence of labels on the path from the root to N. Let $\mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_k$ be all the defeaters according the criterion $\succ_{Ic}$ for $\mathcal{A}_n$. For each $\mathcal{B}_i$ $(1 \leqslant i \leqslant k)$ such that, the argumentation line $\Lambda' = [\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n, \mathcal{B}_i]$ is acceptable, then the node N has a child N_i labeled $\mathcal{B}_i$.
   If there is no defeater w.r.t. the criterion $\succ_{Ic}$ for $\mathcal{A}_n$ or there is no $\mathcal{B}_i$ such that Λ' is acceptable, then N is a leaf.

In a dialectical tree, each path from the root to a leaf corresponds to a different acceptable argumentation line. Note also that every node (except the root) is a defeater of its parent, and leaves are undefeated arguments.

The *marking of a dialectical tree* (García & Simari, 2004) is a process that assigns every node the mark of defeated ("*D*") or the mark of undefeated ("*U*") as follows. Leaf nodes are marked as "*U*"; and, an inner node is marked as "*D*" if it has at least a child marked as "*U*", while an inner node is marked as "*U*" if all its children are marked as "*D*".

Definition 17 (*Warrant*). Let $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}, \mathrm{K}\rangle$ be a CRS-server, and Ic a criterion identifier from the repository set K. A literal Q is warranted from the program $\mathcal{P}$ under $\succ_{Ic}$, if there exists an argument $\mathcal{A}$ for Q from $\mathcal{P}$ and the root of the dialectical tree $\mathcal{T}^{Ic}_{\mathcal{A}}$ is marked as "*U*".

Given a *DeLP*-program $\mathcal{P}$, a criterion identifier Ic and a literal Q, the function $\mathbf{I}(\mathcal{P}, Ic, Q)$ (Definition 8) will return YES, if the literal Q is warranted from $\mathcal{P}$; 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 signature of $\mathcal{P}$.

5.2. Application example

We will develop here a detailed account of how answers for conditional-preference based queries can be obtained following our running example introduced in Example 2. In particular, we will show how the answer can change depending on the criterion that is selected after evaluating a *cp-exp*.

Consider a CRS-server $\langle \mathbf{I}, \mathbb{O}, \mathcal{P}_l, \mathrm{K}_l\rangle$, where $\mathbf{I}$ is a *DeLP*-interpreter, $\mathbb{O} = \{+\}$ has just the context integration operator that was introduced in Section 3; in case of a conflict this operator gives priority to the information received in the query (García et al., 2007), $\mathcal{P}_l = (\Pi_l, \Delta_l)$ is the program introduced in Example 2, and $\mathrm{K}_l = \{\langle sec, S_{sec}\rangle, \langle comf, S_{comf}\rangle\}$ is the repository set presented in Example 3. We will show next how $\langle I, \mathbb{O}, \mathcal{P}_l, \mathrm{K}_l\rangle$ computes an answer for the conditional-preference based query CQ_1 introduced in Example 6 and showed below:

$$CQ_1 = [\mathcal{P}_c, \mathcal{E}_1, suggest(h1)], \quad \text{where}$$
$$\mathcal{P}_c = \{nbHotel(h1), stars(h1, 5), theftZ(h1)\}, \quad \text{and}$$
$$\mathcal{E}_1 = [\{pOfficersZ(h1), stars(h1, 5)\} : comf, sec]$$

Next, we will explain in detail the different steps that a CRS-server performs to compute the answer for CQ_1. The first step is to integrate the contextual information with the stored program (depicted in Fig. 1 as the program $\mathcal{P}'$). In our example this means to integrate $\mathcal{P}_c$ with $\mathcal{P}_l$ in the following way: $\mathcal{P}_l + \mathcal{P}_c = (((\Pi_l \setminus R) \cup \mathcal{P}_c), \Delta)$, where $R = \{\overline{L} : L \in \mathcal{P}_c\}$. Since $\mathcal{P}_c$ is not in conflict with Π_l, then,

$nbHotel(h1), stars(h1,5)$, and $theftZ(h1)$ are added to the set Π_l, and the program $\mathcal{P}_m = (\Pi_m, \Delta_m)$ is obtained:

$$\Pi_m = \left\{ \begin{array}{l} mStops \\ mHDriving \\ atNight \\ nbHotel(h1) \\ stars(h1,5) \\ theftZ(h1) \\ tJam \leftarrow tSlow \end{array} \right\} \Delta_m = \left\{ \begin{array}{l} suggest(H) \prec good(H), nbHotel(H) \\ suggest(H) \prec sStop, nbHotel(H) \\ \sim suggest(H) \prec dangerZ(H) \\ dangerZ(H) \prec theftZ(H) \\ \sim dangerZ(H) \prec pOfficersZ(H) \\ good(H) \prec stars(H,S), S \geqslant 3 \\ \sim good(H) \prec stars(H,S), S < 3 \\ \sim sStop \prec mStops \\ sStop \prec mHDriving \\ sStop \prec atNight \end{array} \right\}$$

The second step is to select the preference criterion that has to be used for determining if CQ_1 is warranted. For this selection, it is necessary to evaluate the expression $\mathcal{E}_1$; since there is no strict derivation of $pOfficersZ(h1)$ from $\mathcal{P}_m$, as a result of evaluating $cond(\mathcal{E}_1, \mathcal{P}_m)$, the selected preference criterion will be *sec*.

The third step is to build dialectical trees using the criterion *sec* in order to determine if $suggest(h1)$ is warranted. Note that $\langle sec, S_{sec} \rangle$ will be the pair from the repository considered in this dialectical process. Fig. 4(a) gives a graphical representation of the resulting dialectical analysis for CQ_1 considering $\succ_{sec}$ as preference criterion. In that figure each argument is depicted using triangle shape (for the detailed structure of these arguments see Fig. 5), dashed lines denote blocking defeat relations and solid lines denote proper defeat relations. In particular, in Fig. 4(a), it can be seen that the root argument of the most right dialectical tree, which supports the conclusion "$\sim suggest(h1)$", is marked as "*U*"; thus, the conclusion "$suggest(h1)$" is not warranted, and the answer for the query is NO.

Now, lets consider that the same CRS-server receives the following conditional-preference based query:

$CQ_2 = [\mathcal{P}_c, \mathcal{E}_2, suggest(h1)]$, *where*

$\mathcal{E}_2 = [\{theftZ(h1), tJam\} : sec, [\{stars(h1,5)\} : comf, sec]]$

For answering CQ_2, the CRS-server uses the same *DeLP*-program $\mathcal{P}_m = (\Pi_m, \Delta_m)$ introduced above because CQ_1 and CQ_2 have the same context $\mathcal{P}_c$. Then when evaluating the expression $\mathcal{E}_2$, *tJam* has not strict derivation, while $stars(h1,5)$ does, thus, in contrast with $cond(\mathcal{E}_1, \mathcal{P}_m)$, the function $cond(\mathcal{E}_2, \mathcal{P}_m)$ will return the criterion identifier *comf*. A graphic representation of resulting dialectical trees by using the criterion $\succ_{comf}$ for CQ_2 is showed in Fig. 4(b), where the conclusion "$suggest(h1)$" is warranted, thus the resulting answer for CQ_2 is YES. Observe that defeat relations arising from the criteria *sec* and *comf* differ, thus different tree structures are built and showed in Fig. 4(a), and (b), respectively.

As we mentioned above, in García et al. (2007) the proposed server is configured to use a fixed comparison criterion embedded in the system; therefore, the answers to our example queries will be always solved using the same criterion. Our approach, as is shown in the complete example above, extends the behavior of García et al. (2007) showing that the same queried literal with the same context but with different conditional-preference expressions, can use a different criterion, possibly obtaining different answers. This was one of our established goals.

5.3. Application example in a mobile robotic environment

Below, we will show how the proposed model could be applied for making a recommendation in the robotic environment that was described in Ferretti et al. (2008). In particular, the application domain consists of a micro-world environment using robots for cleaning tasks. There are boxes spread over the environment and the robot will obtain a recommendation about which box is more convenient to select next in order to move it to a particular place called *store*. In Fig. 6(a) and (b) we present two different scenarios of this robotic environment that will be used in the examples below.

In Ferretti et al. (2008) it was shown that several criteria for selecting boxes may be pertinent: a robot could select the smallest box, or the nearest to itself, or the box that is nearest to the store. In that work, a literal-based preference criterion for comparing arguments (called "*lit-priority*") was presented; this criterion uses a strict partial order (denoted >) over some distinguished literals

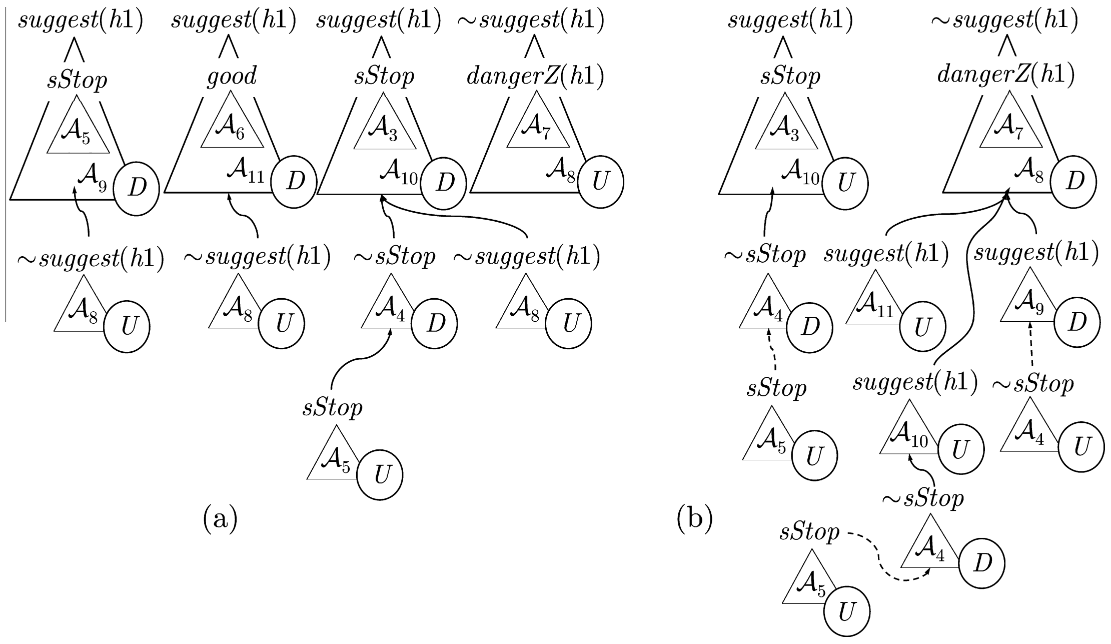

Fig. 4. Dialectical analysis for the queries CQ_1 (a) and CQ_2 (b).

$$\langle \mathcal{A}_{11}, suggest(h1)\rangle, \text{ where } \mathcal{A}_{11} = \begin{array}{l} suggest(h1) \prec good(h1), nbHotel(h1) \\ good(h1) \prec stars(h1,5), 5 \geq 3 \end{array}$$

$$\langle \mathcal{A}_{10}, suggest(h1)\rangle, \text{ where } \mathcal{A}_{10} = \begin{array}{l} suggest(h1) \prec sStop, nbHotel(h1) \\ sStop \prec mHDriving \end{array}$$

$$\langle \mathcal{A}_{9}, suggest(h1)\rangle, \text{ where } \mathcal{A}_{9} = \begin{array}{l} suggest(h1) \prec sStop, nbHotel(h1) \\ sStop \prec atNight \end{array}$$

$$\langle \mathcal{A}_{8}, \sim suggest(h1)\rangle, \text{ where } \mathcal{A}_{8} = \begin{array}{l} \sim suggest(h1) \prec dangerZ(h1) \\ dangerZ(h1) \prec theftZ(h1) \end{array}$$

$$\langle \mathcal{A}_7, dangerZ(h1)\rangle, \text{ where } \mathcal{A}_7 = \{dangerZ(h1) \prec theftZ(h1)\}$$
$$\langle \mathcal{A}_6, good(h1)\rangle, \text{ where } \mathcal{A}_6 = \{good(h1) \prec stars(h1,5), 5 \geq 3\}$$
$$\langle \mathcal{A}_5, sStop\rangle, \text{ where } \mathcal{A}_5 = \{sStop \prec atNight\}$$
$$\langle \mathcal{A}_4, \sim sStop\rangle, \text{ where } \mathcal{A}_4 = \{\sim sStop \prec mStops\}$$
$$\langle \mathcal{A}_3, sStop\rangle, \text{ where } \mathcal{A}_3 = \{sStop \prec mHDriving\}$$

Fig. 5. Arguments considered as part of dialectical analysis showed in Fig. 4(a) and (b).

used in arguments: $L > L'$ means that the literal L is preferred to the literal L'. Using *lit-priority* an argument $\mathcal{A}$ will be preferred to an argument $\mathcal{B}$ with respect to a particular order >, iff there are two literals $L_1 \in^* \mathcal{A}$ and $L_2 \in^* \mathcal{B}$ such that, $L_1 > L_2$, and there are no literals L_3 and L_4 such that $L_3 \in^* \mathcal{A}, L_4 \in^* \mathcal{B}$, and $L_4 > L_3$. Note that $L \in^* \mathcal{A}$ means that there exists a defeasible rule $(q_0 \prec q_1, q_2, \ldots, q_n)$ in $\mathcal{A}$ *and* $L = q_i$ $(1 \leqslant i \leqslant n)$.

If a different priority order > among literals is used, then a different literal-based preference criteria can be defined. In our application example, we will consider three different preference criteria based on literals.

- The criterion denoted $\succ_{sm}$ means that *"the robot will prefer first smaller boxes, then boxes near itself, and in the last case boxes near to the store"*.
- The criterion denoted $\succ_{nr}$ means that *"the robot will prefer first boxes near itself, then boxes near to the store, and in the last case smaller boxes"*.
- The criterion denoted $\succ_{ns}$ means that *"the robot will prefer first boxes near to the store, then boxes near itself, and in the last case smaller boxes"*.

In order to recommend which box the robot should move next, the following *DeLP* program will be used.

$$\Pi_k = \left\{ \begin{array}{l} stuffed\_store \leftarrow stored\_boxes(Num), Num \geqslant 3 \\ complex\_path \leftarrow free\_boxes(Num), Num \geqslant 5 \end{array} \right\}$$

$$\Delta_k = \left\{ \begin{array}{l} recommend(Box) \prec better(Box, Obox) \\ better(Box, Obox) \prec nearer\_robot(Box, Obox) \\ better(Box, Obox) \prec nearer\_store(Box, Obox) \\ better(Box, Obox) \prec smaller(Box, Obox) \\ \sim better(Box, Obox) \prec nearer\_robot(Obox, Box) \\ \sim better(Box, Obox) \prec nearer\_store(Obox, Box) \\ \sim better(Box, Obox) \prec smaller(Obox, Box) \end{array} \right\}$$

In Ferretti et al. (2008), a single comparison criterion was used; instead, here we propose to provide the CRS-server with the program (Π_k, Δ_k), and the three literal-based criterion described above. Then, we propose to use a *cp-exp* to program how to select dynamically the more suitable comparison criterion depending on the boxes that are in the environment in that moment. We will present a *cp-exp* that implements the following intuition: "*if the store is stuffed with boxes then use a criterion that prioritizes small boxes, else if there are several free boxes use a criterion that prioritizes boxes near the store, otherwise use a criterion that prioritizes boxes near to the robot*". This intuition can be captured with the *cp-exp* $\mathcal{E}_4$ included below. Observe that the literals *stuffed_store* and *complex_path* can be derived using strict rules from Π_k and both rely on information that depends on the particular scenario where the robot is involved.

$$\mathcal{E}_4 = [\{stuffed\_store\} : sm, [\{complex\_path\} : ns, nr]]$$

Recall the scenario depicted in Fig. 6(a) where there are three boxes at the Store and two free boxes to select: *box*1 and *box*2. The following conditional-preference based query can be used to ask the CRS-server for a recommendation that considers both the current scenario and the *cp-exp* $\mathcal{E}_4$ defined above:

$$CQ_4 = [\mathcal{P}_c^{(a)}, \mathcal{E}_4, recommend(X)]$$

The contextual information of CQ_4 is $\mathcal{P}_c^{(a)} = \{free\_boxes(2), stored\_boxes(3), nearer\_robot(box2, box1), nearer\_store(box1, box2), smaller(box1, box2)\}$ contains the information perceived by the robot about the environment: there are three boxes at the Store, two free boxes(*box*1 and *box*2), *box*2 is nearer than *box*1, *box*1 is nearer to the Store than *box*2, and *box*1 is smaller than *box*2.

To answer CQ_4 a CRS-server considers the program $\mathcal{P}_{CQ_4} = (\Pi_k \cup \mathcal{P}_c^{(a)}, \Delta_k)$ as the result of adding the elements of $\mathcal{P}_c^{(a)}$ as facts to the program stored in the server. Observe that *stuffed_store* is strictly derived from $\mathcal{P}_{CQ_4}$ using one strict rule from Π_k and the literal *stored_boxes*(3) of $\mathcal{P}_c^{(a)}$. Therefore the evaluation of the *cp-exp* $\mathcal{E}_4$ from $\mathcal{P}_{CQ_4}$ results in the selection of the preference criterion $\succ_{sm}$, which prefers to pick small boxes first.

Below, in Fig. 7 we present the dialectical trees that are build for the query CQ_4 from the program $\mathcal{P}_{CQ_4}$ considering the preference criterion $\succ_{sm}$. Observe that there are three trees, the first and the second trees correspond to arguments for recommending *box*1 (*i.e.*, $X = box1$), and third one to the argument for recommending *box*2 (*i.e.*, $X = box2$). Since there is at least one dialectical tree for *recommend*(*box*1) that have the root node marked as *U* (undefeated), then the answer to CQ_4 will be YES, with $X = box1$.

Therefore, in this scenario, due to the selected preference criterion $\succ_{sm}$, the RS-server warrants the recommendation for selecting *box*1 first. Next, we will show that in a different scenario, like the one described in Fig. 6(b), because the contextual information is different, the same *cp-exp* $\mathcal{E}_4$ will perform differently and it will return a different preference criterion.

Consider now the scenario depicted in Fig. 6(b) where there is one box at the Store and two free boxes to select: *box*1 and *box*2. The conditional-preference based query CQ_5 that we show below

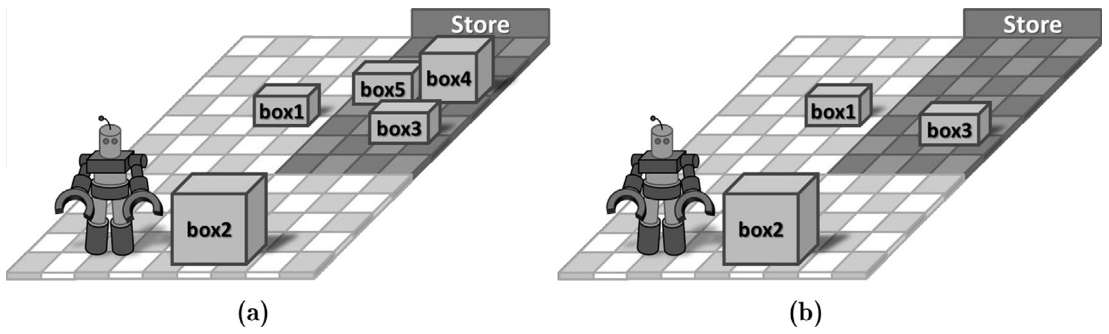

Fig. 6. Two scenarios of the robotic environment.

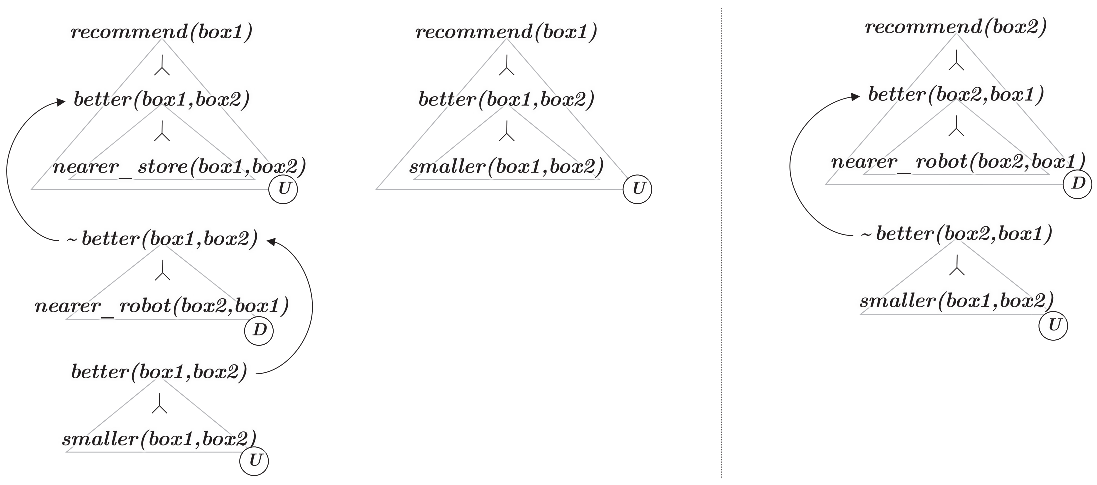

Fig. 7. Dialectical trees built to answer CQ_4.

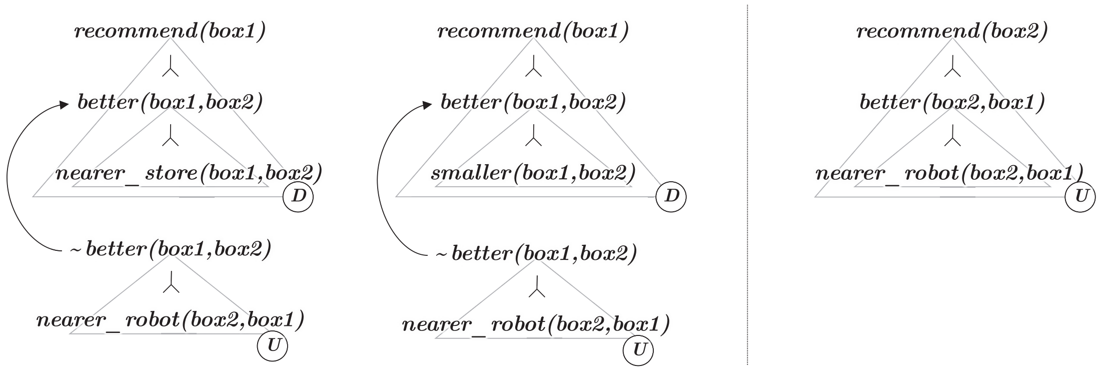

Fig. 8. Dialectical trees built to answer CQ_5.

can be used to ask the CRS-server for a recommendation that considers that scenario and the *cp-exp* $\mathcal{E}_4$ defined above:

$$CQ_5 = [\mathcal{P}_c^{(b)}, \mathcal{E}_4, recommend(X)]$$

Note that the only difference between this query and CQ_4, is the contextual information. Here, the context is $\mathcal{P}_c^{(b)} = \{free\_boxes(2), stored\_boxes(1), nearer\_robot(box2, box1), nearer\_store(box1, box2), smaller(box1, box2)\}$. In this case, when evaluating $\mathcal{E}_4$ from $\mathcal{P}_{CQ_5} = (\Pi_k \cup \mathcal{P}_c^{(b)}, \Delta_k)$ it holds that neither *stuffed_store*, nor *complex_path* have strict derivations, thus, the selected preference criterion is $\succ_{nr}$. The dialectical trees in Fig. 8 are built from $\mathcal{P}_{CQ_5}$ to answer the query.

In this case, since there is at least one dialectical tree for *recommend*(*box*2) that have the root node marked as *U* (undefeated), then the answer to CQ_5 will be YES, with $X = box2$. That is, in this scenario, due to the selected preference criterion $\succ_{nr}$, the CRS-server warrants the recommendation for selecting *box*2 first.

6. Related work

In considering the existing literature, in SubSection 6.1 we discuss the related work in the area of recommendation systems, highlighting the contributions of our approach. Next, in

SubSection 6.2, we analyze the differences of our work with other approaches that focus on multiple argument preference criteria. Then, we discuss how conditional expressions are used in the field of preference representation.

6.1. Works on recommendation systems

As mentioned in Section 1, the integration of argumentation-based reasoning with recommender systems was done with the goal of producing reasoned recommendations. Several works of the literature (Bedi & Vashisth, 2011; Budzynska, Rocci, & Yaskorska, 2014; Heras et al., 2013; Rajpal et al., 2014) have already applied argumentation methods to recommendation systems. Recently, Bedi and Vashisth (2015), have presented an interest-based recommender system (IBRS) for personalization of recommendations; IBRS considers user's preference and employs argumentation to generate suggestions. The use of argumentation allows these systems to reason about the underlying interests behind user's personal preferences, and helps to resolve conflicts (using preferences) between the recommendations with convincing arguments. The originality of this approach lies in that the framework considers the behavior of a group of agents, which work collectively, and these agents have argumentative dialogues to revise the user's model and improve upon current recommendations. Clearly, integrating our proposal to IBRSs will add several interesting characteristics to this type of systems. If the recommender system is a CRS-server, then the user could interact directly with the inference engine. The *DeLP*-program rules could model recommendation aspects which could be modified by the user during the revision phase. Incorporating qualitative approaches for comparing arguments is another useful characteristic for IBRSs since it provides users with a natural and easy way of understanding how the recommendations are obtained. Despite sharing the motivation of generating convincing recommendations to a final user, we do not focus on constructing a formalism exclusively for recommender systems, indeed we generalize our proposal to argumentation-based user support systems improving aspects of these systems such as transparency (in inference process), flexibility (for changing their preference behaviors), and reliability (in their answers).

In Briguez et al. (2013), the authors present an argumentative trust-based news recommender systems. Their systems is centered in a set of postulates that capture the intuitions behind user's trust obtained from interactions of news sources, reports, topics, and viewers. Like us, they use *DeLP* as the qualitative reasoning mechanism to infer the recommendations, and their recommender system uses a *DeLP*-program with defeasible rules, where each rule models a postulate; nevertheless, unlike our approach, in their system the comparison criteria is fixed. They use as comparison criterion generalized specificity (see García & Simari (2004)), which prefers more precise argument (*i.e.,* with greater information content) or a more concise arguments (i.e., with less use of rules). Since they want to establish a particular preference among the postulates, they codify the defeasible rules in such a way that this fixed criterion will capture those preferences. This can lead to complications if the user wants to change the preferences among these postulates, since this change will require a complete revision of every rule *DeLP*-program to adapt them to the fixed criterion. In contrast, in our approach it would not be necessary to modify the *DeLP*-program, because the user can select the most appropriate comparison criterion that captures the desired preference among the rules that model the postulates simply by using *cp-exp*. A similar situation arises in Briguez et al. (2014), where an argumentative movie recommender system is proposed, because they use a fixed comparison criteria to establish the preferences among the defeasible rules that model their postulates. The authors of both works recognize the importance of changing the preferences among their postulates or introducing new postulates (which requires a revision of the postulates preference ordering); therefore, our proposed mechanism can provide a useful tool to improve these argumentative recommender systems.

In Tucat et al. (2009) a particular implementation of recommender systems, based on *DeLP*-servers, called recommender server (*re-server*) was developed; there, the authors focused on the definition of different types of *contextual queries* that a *re*-server can solve. The first type of query, the regular contextual query allows the clients to send a query to the server adding a specific context. The second type, called multiple contextual query, provides the clients with the facility of grouping several regular contextual queries in just one message. The third type of query is a particular case of previous query consisting of a sequence of queries with only one context (see Tucat et al. (2009) for further details). Finally, the authors define the notion of contextual interrogation which is a generalization of the other tree queries where the context modifications effected over the server by the queries may remain for subsequent queries of the same message. In a similar development, our proposal is based on *DeLP*-servers; nevertheless, in contrast with us, they use a preference criterion embedded into the *DeLP*-interpreter, i.e., to answer a query, the server is configured to use always the same criterion. In fact, we provide clients with the possibility of indicating to the server what criterion could use at the moment of computing the answer for a specific query; consequently, the criterion used by server varies dynamically. A interesting characteristic of our model is that the different types of queries proposed in Tucat et al. (2009) could be extended incorporating the conditional expression introduced here.

6.2. Works on argumentation

Handling multiple preference criteria has not been widely studied in the argumentation literature (Kaci, 2011). In Amgoud, Parsons, and Perrussel (2000) an approach to reason from multiple preference relations was proposed. Their main contribution is to take into account several pre-orderings on the same knowledge base. These different pre-orderings are given by the notion of contextual preference, i.e., preferences which depend upon a particular context. Each preference relation between arguments is induced from a pre-ordering expressed in a particular context. To determine the acceptable arguments, the set of preference relations is linearly ordered using another preference relation. Similarly to them, our proposal takes into account several preference relations between arguments by means of the different argument preference criteria considered by a server, notwithstanding, there are several differences between them. First, our approach is focused on structured argumentation, while theirs is based on abstract argumentation. Second, our aim is not the handle inconsistency considering several preference criteria together, but to introduce a tool where users can take part of the inferential process indicating the criterion upon which the reasoning system will base its answers. In Amgoud et al. (2000) there is no specific tool for the user to specify which preference relation to use. In our approach user's preferences play an important role in the choice of a particular argument preference criterion.

In Amgoud, Bonnefon, and Prade (2005), an argument based approach to multi-criteria decision making was presented. In this work arguments supports decisions; the idea is that a decision has some justification if it leads to the satisfaction of some decision policy. A decision policy may be satisfied either in a positive way (if the satisfaction degree is higher than the neutral point of a given scale) or in a negative way (if the satisfaction degree is lower than the neutral point of the given scale). Thus, the force of an argument depends on three components: the certainty level of the argument, the importance degree of the choice policy which is evaluated for the decision supported by that argument, and the (dis) satisfaction degree of that policy. These three components are used as argument preference criteria for their abstract argumentation system. In contrast, our approach is focused on considering several preference criteria, but in a structured argumentation setting. The main difference is that they do not specify how their argument preference criteria are chosen. Given this consideration, the conditional expressions presented here could be useful to model tools that allow users to change these criteria and principles in decision support systems using the approach proposed in Amgoud et al. (2005).

In the decision making literature there are works focused on the association of conditions to user's preferences (Boutilier, Brafman, Hoos, & Poole, 1999; Li, Vo, & Kowalczyk, 2011); however, this association differs from the way in which is proposed in our approach. In Boutilier et al. (1999), an approach was proposed where the preference is subject to conditional dependence. A preference relation is defined as a total preorder (a ranking) over some set of variables such that the preference over the values of one variable depends on the value of others. Their main contribution is a graphical representation of preferences that reflects conditional dependence and independence of preference statements under a *ceteris paribus* (*all other things being equal*) interpretation. Similarly to us, the authors present a model for representing and reasoning with the user's preferences, where conditional preference expressions are allowed; but, contrariwise, they provide a framework where the preferences are considered for decision making where the space of possible actions or decisions available to someone is fixed, with well-understood dynamics. In our framework the situation is different, i.e., the selected application domains are dynamic and agents deal with incomplete and contradictory information; for that reason, our research is focused on argumentative systems that can handle this type of epistemic state. In fact, it is also important to remark that in contrast with them, we use conditional expressions to be able to choose, in a declarative way, the way used by the server for comparing arguments.

A conditional expression is a structure commonly used in the literature of computation science. For instance, Dijkstra in Dijkstra (1975) suggested different forms of selection and loop structures. In order to provide control statements to be supported by programs, Dijkstra introduces a guarded command language, where a guard allows a statement to be execute only when a specified condition is true. Guards in *cp-exp* can be related to the work in Dijkstra (1975); however, in there they are used for a different purpose. Here, they constitute the central structure that the *DeLP*-interpreter can use to obtain the argument preference criterion that it will use in the inference process.

7. Conclusions and future work

The use of argumentation-based reasoning engines provides useful advantages in the implementation in a variety of user support systems such as expert systems, systems for automated negotiation, recommender systems and decision support systems (see for instance, Amgoud & Prade (2009), Chesñevar et al. (2009, chap. 20), Ferretti et al. (2008), Monteserin & Amandi (2011), Bedi & Vashisth (2015), Briguez et al. (2013, 2014)). As pointed out in the literature, having different argument comparison criteria available introduces an interesting degree of flexibility in these systems; however, in most of the proposed systems, the comparison criterion is either a fixed component or, if there exist multiplicity of criteria at the user disposal, there exist no way of changing it once a criterion is chosen. The contribution of our approach aims to offer an improvement in current systems, providing a concrete programmable mechanism for the user to select the comparison criteria, and a formalization of the semantics for the interaction of such mechanism with the elements of the system. In particular, the conditional-preference based query, introduced in Section 3, and the formalization introduced in Section 4, are two of the main strengths of this proposal.

An important advantage of introducing argumentation in a recommender system is to provide more transparency to the recommendation inference process. In particular, argumentation contributes the user confidence and trust on the system answers by giving explanations about the reasons favouring a recommendation and the reasons against it (Chesñevar et al., 2009, chap. 20; Tintarev & Masthoff, 2007). As we have explained, argument comparison criteria have an important role in this regard, since they determine which arguments (reasons) are preferred when conflicts arise; in other words, the comparison criterion used for making a recommendation should be part of that explanation. Our proposal introduces more clarity about that aspect, which as yet has not been the focus of the existing argumentative recommender systems. The formalism presented here allows the user, via a conditional-preference expression, to select the comparison criteria bringing about important benefits by clarifying the recommendation process. As we have shown in Section 5, this provides the user with insights about the information that will be prioritized in the reasoning process; in this sense, a recommendation is more compelling if the user, besides been aware of reasons, can understand *why* a reason is preferred over others. In the same section, we discussed how the user can guide the recommendation process according to its current preferences or needs, and this guidance can be clearly seen in the dialectical trees produced for a given recommendation. Thus, the proposed mechanism contributes to augment the trust of the user on the recommendations provided by the system.

In Section 4, we introduced a tree representation for conditional-preference expressions which provided ways to analyse several properties of such expressions. These properties are useful to identify when an expression can be optimized to avoid the computation of redundant inferences and characterizing when certain paths in the expressions will not be traversable. Also, using these results we have characterized whether an expression is sound, i.e., the expressions where every path to a criterion can be traversed. These properties are of special interest in our formalism since they allow to construct valid expression, i.e., expressions maintaining relations coherent between guards that justify the choice of a particular criterion.

As for future work, there are several lines of research already under consideration and others that we plan to follow. Also we are interested in studying new developments to tackle the limitations that we discuss in the following paragraphs.

In the formalism we have introduced is possible to specify a comparison criterion that can be seen as a combination of two or more criteria; however, our system does not provide a systematic way of doing this combination. For that, we can follow the approach proposed in Briguez et al. (2014) or in Deagustini et al. (2012), where a combination of two criteria (rule priority and generalized specificity) are handled by the systems as it were a single criteria. This, *ad hoc* and fixed solution presents several limitations regarding the extendibility and the modularity of the system, since changing one of the involved criteria, or the way in which they are combined, requires to revise the whole composed criterion. To provide a systematic solution to this limitation, as a future work, it would interesting to study special operators to combine comparison criteria and how these can be integrated with the conditional expressions that we have proposed in this article.

As we have mentioned, our proposal has an impact in multi-agent systems; however, we were not focused in achieving optimization regarding the time efficiency of the query answering process. There are several aspects of the process that can be studied in this regard: the argumentative dialectical process, the client–server interaction overhead, the knowledge revision made by the server with each query and the sequential processing (among others). For instance, the notion of multiple-contextual queries presented in Tucat et al. (2009) could be integrated to our approach in order to avoid the overhead of sequential processing. In particular, our conditional preference based queries could be extended to include several queried literals, which should be answered using the same *cp-exp* and the same context.

Finally, one of our future goals is to broaden the presented framework considering alternatives in the evaluation of the satisfaction of a guard, possibly using the notion of defeasible derivation or the notion of warrant. It is also interesting to note that *cp-exps* could be optimized, avoiding in that manner certain incoherence or contradictory situations, here we have studied just the case where repeated literals could not appear.

Acknowledgments

This research was funded by PGI-UNS (Grants 24/N035, 24/N030) and PIP-CONICET (Grant 112-201101-01000).

References

Amgoud, L., Cayrol, C., & Berre, D. L. (1996). Comparing arguments using preference ordering for argument-based reasoning. In *Eighth International conference on tools with artificial intelligence, ICTAI'96* (pp. 400–403). Toulouse, France, November 16–19.

Amgoud, L., Parsons, S., & Perrussel, L. (2000). An argumentation framework based on contextual preferences. In Proceedings of the 3rd International Conference on Formal and Applied Practical Reasoning, FAPR '00, 2000 (pp. 59–67).

Amgoud, L., Bonnefon, J.- F., & Prade, H. (2005). An argumentation-based approach to multiple criteria decision. In *symbolic and quantitative approaches to reasoning with uncertainty, 8th European conference, ECSQARU 2005* (pp. 269–280). Barcelona, Spain, July 6–8, Proceedings.

Amgoud, L., & Prade, H. (2009). Using arguments for making and explaining decisions. *Artificial Intelligence, 173*, 413–436.

Antoniou, G., Maher, M. J., & Billington, D. (2000). Defeasible logic versus logic programming without negation as failure. *Journal of Logic Programming, 42*, 47–57.

Bedi, P., & Vashisth, P. (2011). Interest based recommendations with argumentation. *Journal of Artificial Intelligence, ANSI*, 119–142.

Bedi, P., & Vashisth, P. B. (2015). Argumentation-enabled interest-based personalised recommender system. *Journal of Experimental & Theoretical Artificial Intelligence, 27*, 199–226.

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems, 46*, 109–132.

Boutilier, C., Brafman, R. I., Hoos, H. H., & Poole, D. (1999). Reasoning with conditional ceteris paribus preference statements. In *Proceedings of the fifteenth conference on uncertainty in artificial intelligence UAI'99* (pp. 71–80). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Briguez, C. E., Budán, M. C. D., Deagustini, C. A. D., Maguitman, A. G., Capobianco, M., & Simari, G. R. (2014). Argument-based mixed recommenders and their application to movie suggestion. *Expert Systems with Applications, 41*, 6467–6482.

Briguez, C. E., Capobianco, M., & Maguitman, A. G. (2013). A theoretical framework for trust-based news recommender systems and its implementation using defeasible argumentation. *International Journal on Artificial Intelligence Tools, 22*.

Budzynska, K., Rocci, A., & Yaskorska, O. (2014). Financial dialogue games: A protocol for earnings conference calls. In *Computational models of argument – Proceedings of COMMA 2014* (pp. 19–30), Atholl Palace Hotel, Scottish Highlands, UK, September 9–12.

Chesñevar, C. I., & Simari, G. R. (2007). A lattice-based approach to computing warranted beliefs in skeptical argumentation frameworks. In *IJCAI 2007, proceedings of the 20th international joint conference on artificial intelligence* (pp. 280–285), Hyderabad, India, January 6–12.

Chesñevar, C., Maguitman, A. G., & González, M. P. (2009). Empowering recommendation technologies through argumentation. *Argumentation in artificial intelligence* (pp. 403–422). Springer.

Deagustini, C. A. D., Dalibón, S. E. F., Gottifredi, S., Falappa, M. A., & Simari, G. R. (2012). Consistent query answering using relational databases through argumentation. In *Database and expert systems applications – 23rd international conference, DEXA 2012. Proceedings, Part II* (pp. 1–15), Vienna, Austria, September 3–6.

Deagustini, C. A. D., Fulladoza Dalibón, S. E., Gottifredi, S., Falappa, M. A., Chesñevar, C. I., & Simari, G. R. (2013). Relational databases as a massive information source for defeasible argumentation. *Knowledge-Based Systems, 51*, 91–109.

Dijkstra, E. W. (1975). Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM, 18*, 453–457.

Ferretti, E., Errecalde, M., García, A. J., & Simari, G. R. (2008). Decision rules and arguments in defeasible decision making. In *Computational models of argument: Proceedings of COMMA 2008* (pp. 171–182), Toulouse, France, May 28–30.

García, A. J., Chesñevar, C. I., Rotstein, N. D., & Simari, G. R. (2013). Formalizing dialectical explanation support for argument-based reasoning in knowledge-based systems. *Expert Systems with Applications, 40*, 3233–3247.

García, A. J., & Simari, G. R. (2014). Defeasible logic programming: Delp-servers, contextual queries, and explanations for answers. *Argument & Computation, 5*, 63–88.

García, A. J., Rotstein, N. D., Tucat, M., & Simari, G. R. (2007). An argumentative reasoning service for deliberative agents. In *Knowledge science, engineering and management, second international conference, KSEM 2007. Proceedings* (pp. 128–139). Melbourne, Australia, November 28–30.

García, A. J., & Simari, G. R. (2004). Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming (TPLP), 4*, 95–138.

Godo, L., Marchioni, E., & Pardo, P. (2012). Extending a temporal defeasible argumentation framework with possibilistic weights. In *Logics in artificial intelligence – 13th european conference, JELIA 2012. Proceedings* (pp. 242–254), Toulouse, France, September 26–28.

Heras, S., Atkinson, K., Botti, V., Grasso, F., Julián, V., & McBurney, P. (2013). Research opportunities for argumentation in social networks. *Artificial Intelligence Review, 39*, 39–62.

Kaci, S. (2011). Working with preferences: Less is more. *Cognitive technologies*. Springer.

Li, M., Vo, Q. B., & Kowalczyk, R. (2011). Majority-rule-based preference aggregation on multi-attribute domains with cp-nets. In *Proceedings of the tenth international joint conference on autonomous agents and multi-agent systems (AAMAS)* (pp. 659–666).

Martinez, M. V., García, A. J., & Simari, G. R. (2012). On the use of presumptions in structured defeasible reasoning. In *Computational models of argument – Proceedings of COMMA 2012* (pp. 185–196), Vienna, Austria, September 10–12.

Monteserin, A., & Amandi, A. (2011). Argumentation-based negotiation planning for autonomous agents. *Decision Support Systems, 51*, 532–548.

Prakken, H., & Sartor, G. (1997). Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics, 7*, 25–75.

Rahwan, I., & Simari, G. R. (2009). *Argumentation in Artificial Intelligence* (1st ed.). Springer Publishing Company, Incorporated.

Rajpal, A., & Khurana, P. (2014). Visualization in argument based recommender system. *International Journal of Computer Science & Information Technologies, 5*.

Simari, G. R., & Loui, R. P. (1992). A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence, 53*, 125–157.

Tang, Y., Hang, C.- W., Parsons, S., & Singh, M. P. (2012). Towards argumentation with symbolic dempster-shafer evidence. In *Computational models of argument – Proceedings of COMMA 2012* (pp. 462–469). Vienna, Austria, September 10–12.

Tintarev, N., & Masthoff, J. (2007). A survey of explanations in recommender systems. In *Proceedings of the 23rd international conference on data engineering workshops, ICDE 2007 (pp. 801–810)*. 15–20 April 2007. Istanbul, Turkey.

Tucat, M., Garcia, A. J., Simari, G. R. (2009). Using defeasible logic programming with contextual queries for developing recommender servers. In *AAAI Fall Symposium. The Uses of Computational Argumentation* AAAI Technical Report. AAAI.

Vreeswijk, G. (1997). Abstract argumentation systems. *Artificial Intelligence, 90*, 225–279.