Undefined 0 (2018) 1–0
IOS Press

Combining RDF and SPARQL with CP-theories to reason about preferences in a Linked Data setting

Vito Walter Anelli [a], Renato De Leone [b], Tommaso Di Noia [a], Thomas Lukasiewicz [c], Jessica Rosati [a,b,*]
[a] *DEI, Polytechnic University of Bari, Italy*
E-mail: {vitowalter.anelli, tommaso.dinoia, jessica.rosati}@poliba.it
[b] *School of Science and Technology, University of Camerino, Italy*
E-mail: {renato.deleone, jessica.rosati}@unicam.it
[c] *Department of Computer Science, University of Oxford, UK*
E-mail: thomas.lukasiewicz@cs.ox.ac.uk

Abstract. Preference representation and reasoning play a central role in supporting users with complex and multi-factorial decision processes. In fact, user tastes can be used to filter information and data in a personalized way, thus maximizing their expected utility. Over the years, many frameworks and languages have been proposed to deal with user preferences. Among them, one of the most prominent formalism to represent and reason with (qualitative) conditional preferences (CPs) are *conditional preference theories* (*CP-theories*). In this paper, we show how to combine them with Semantic Web technologies in order to encode in a standard SPARQL 1.1 query the semantics of a set of CP statements representing user preferences by means of RDF triples that refer to a "preference" OWL ontology. In particular, here we focus on context-uniform conditional (cuc) acyclic CP-theories [44]. The framework that we propose allows a standard SPARQL client to query Linked Data datasets, and to order the results of such queries relative to a set of user preferences.

Keywords: preferences, ceteris paribus, CP-theories, CP-nets, Linked Data, preference queries, SPARQL, DBpedia

1 Introduction

Dealing with user preferences is an important aspect of every application designed to provide personalized information to the end-user. The original interest in preferences can be found in decision theory, as a way to support complex, multifactorial decision processes [21], and nowadays every personalized system needs a preference model to capture what the user likes or dislikes. Once the user model has been represented, it is then exploited to filter information coming from a data source, e.g., a database, in order to provide a ranked list of results matching the order encoded in the preferences of the user.

Query languages usually let us specify the information that we want to be returned (hard constraints), where the result set contains elements with no specific order with reference to user preferences. It contains all those resources that exactly match the constraints represented by the query. As a matter of fact, if just one of the requirements representing the query is not fulfilled, the result set can be empty. At the same time, returning huge and unordered sets of answers could be useless and even counter-productive. A possible way to bypass these issues is to allow the language to represent both hard constraints—used to return only relevant results—and soft ones, i.e., preferences—to rank the results by fulfilling a user's tastes. Approaches to preference representation can be either quantitative or qualitative [12]. The former are based on a total order-

*Corresponding author

ing of the outcomes, given by a scoring function, while the latter enable the representation of partial orders, since preferences are treated as independent dimensions. From a user perspective, a qualitative approach is more natural than the quantitative one [37]. Indeed, in the first case, the user has just to provide pairwise qualitative comparisons, while in the second case, she has to assign a value to many alternatives, which very often are represented in a multi-attribute setting. Regarding the Linked (Open) Data world, the notion of qualitative preferences in SPARQL queries was introduced in [40] by Siberski et al., whose *preference-based querying* language extends SPARQL through the introduction of solution modifiers (the PREFERRING clause). Their query formulation retrieves only items that are the most preferred ones, or equivalently *undominated*. The work [27] builds on the earlier approach of [40], but adds preferences at the level of filters, rather than as a solution modifier. The *Pref-SPARQL* syntax of [27] needs no additional solution modifier to express qualitative preferences, as it leverages the expressive power of SPARQL 1.1. However, the approaches proposed in [40] and [27] both have an important limitation: they are not able to provide an order of all the available outcomes that reflects user preferences. That is, they return only the undominated (a.k.a. Pareto-optimal) results, i.e., those outcomes best satisfying user preferences. Unfortunately, the size of the resulting answer set could be too small to be of practical use. This fosters moving beyond the Pareto-optimal set identification to a top-k scenario [37], where firstly available outcomes are ordered, even if with the ties implied by a qualitative approach, from best (most preferred) to worst (less preferred) according to a given user's preferences, and then the first k results are returned.

In this paper, the focus is on model-based preference reasoning, which relies on specific assumptions about the structure of the preference relation [23]. The simplest assumption that can be made is that the target ranking of a set of resources, described in terms of multiple attributes, can be represented as a lexicographical order [13]. Lexicographical preference models define orders of importance on the attributes that describe the objects of a domain. As an example, consider the choice of a movie. Typically, the most important attribute that one considers is the genre of the movie (e.g., drama, superhero, etc.). Then, among the movies of the preferred genre, the choice can rely on the movie's actor (e.g., Tom Hanks, Christian Bale, etc.). The assumption of a lexicographical order restricts significantly the hypothesis space, but induces a bias rarely justified in practical applications. In fact, preferences on individual attributes are generally not independent of each other. With reference to the movie domain, Tom Hanks may be preferred to Christian Bale, if the movie genre is drama, while Christian Bale may be preferred in case of a superhero film. *CP-nets* [3] offer a language to express preferences on the values of single attributes, and, at the same time, allow to model dependencies of this type. A CP-net is a qualitative graphical representation that reflects conditional dependence and independence of preferences under a *ceteris paribus* (all else being equal) interpretation. It is a compact representation of a complex preference relation (partial order), where each node refers to a single attribute and is associated with a function that assigns a preference relation on the values of that attribute to each combination of the values of the parent attributes. More precisely, CP-nets require that the user specifies (i) for any attribute A of interest, which other attributes can impact her preferences for values of A (the parents of A), and (ii) for each instantiation of the parent attributes, the preference ordering over values of A. Points (i) and (ii) could make CP-nets a rather rigid formalism, compared to the expressive needs of a user. Furthermore, some statements that are very natural for the user to assess cannot be represented within a CP-net. *Conditional preference theories* (or *CP-theories*) [44] are a more general and flexible formalism for qualitative preferences that allows to go beyond the expressiveness limitations of CP-nets.

In our previous work [39], we focused on the well-known CP-net graphical language and have addressed the problem of preference representation and reasoning with Linked Data from different perspectives. We have proposed a vocabulary to represent statements formulated according to the ceteris paribus semantics and have shown how to encode a CP-net by means of this vocabulary. Inspired by [27], we have also explained how to embed such a compact preference model into a SPARQL 1.1 query in order to access semantic data in a personalized way. The present work extends the leading motivation and the approach of our previous paper [39], but embraces the more general and flexible formalism of CP-theories. We point out that the approach proposed here only deals with context-uniform conditional (cuc) acyclic CP-theories [44], which are a special type of CP-theories exposing nice polynomial computational properties while comparing outcomes.

This paper heavily extends the approach presented in [39] in many points. First of all, here we deal with the much more expressive CP-theories instead of CP-nets. A new extended vocabulary as well as a completely new algorithm to encode CP-theories in SPARQL is also proposed. Moreover, an implementation of the overall framework is presented together with experimental evaluations targeted at assessing the users' experience in representing their preferences as CP-theories and the performance of the tool. The main contributions of this work can be summarized as follows:

- presentation of RDF vocabularies to represent qualitative preference statements over Linked Data, built on top of the vocabulary proposed in [39], but adjusted for more general preference statements;
- an encoding into RDF triples of the qualitative preferential information represented by a CP-theory and the exploitation of theoretical results of [43] to compute a partial order over items for cuc-acyclic cases;
- a procedure to translate conditional preference statements into a SPARQL 1.1 query able to retrieve a **ranked list** of resources whose order reflects the user preferences;
- an application framework that meets a user's needs while representing her preferences as a CP-theory encoded in RDF and that eventually allows a SPARQL-enabled software agent to retrieve a ranked list of resources according to the users's tastes;
- An experimental evaluation to verify the effectiveness of the proposed approach.

The rest of this paper is structured as follows. Section 2 presents the motivating scenario that fostered the overall approach. The semantics of CP-theories and a recap on some relevant results and theorems is provided in Section 3. In Section 4, we propose an RDF vocabulary to represent a CP-theory with the preferential statements of a user, and then we show how to embed the RDF version of the CP-theory into a SPARQL query able to retrieve a ranked list of results ordered according to user's preferences. Section 5 describes the tool that supports the user both in the formulation of her preferences under the CP-theories semantics and in retrieving the resources of interest ordered according to her preferences. The results on a user study are reported in Section 6, where we also measure the performance of the implemented system on a synthetic dataset. Section 7 provides an overview of related work about preference reasoning and enabling query languages with preferences. Conclusions close the paper.

2 Motivating scenario

The leading scenario behind the framework that we propose here is that of a distributed system where a user may pose a query to a SPARQL endpoint and have the returned results ordered with respect to a set of personal preferences on a specific knowledge domain. For instance, a user might be willing to get a list of books to read by querying DBpedia and then have it ranked according to a set of preferences hosted on their own Web page. A possible implementation of the approach that we propose is depicted in Fig. 1,[1] where the main building blocks required to implement the whole framework are shown:

- a reference model to encode and reason with preferences (CP-theory in our case);
- an ontological vocabulary to represent preferences by adopting Web languages;
- a tool able to handle and manage preferences as well as to encode them in a set of SPARQL queries.

In order to implement the whole framework, we propose the following deployment and interaction steps.

Deploy.

- *Model user preferences*. We adopt CP-theories [43] as reference model to represent user preferences. As we will see in Section 3, they are a formalism to represent and reason with sets of qualitative preferences.[2]
- *Encode preferences in RDF by means of a preference ontology*. We developed an OWL ontology to represent CP-theory statements encoding user preferences (see Section 4).
- *Publish user preferences publicly on the Web*. The aim is to foster a principled adoption of user preferences by systems interested in providing a personalized access to

[1] For ease of presentation, in this paper, we use DBpedia as the main dataset to query. The approach can be adapted to any Linked Data dataset.

[2] The interest here is not whether such preferences are automatically learned from data or manually modeled and set by a human agent.

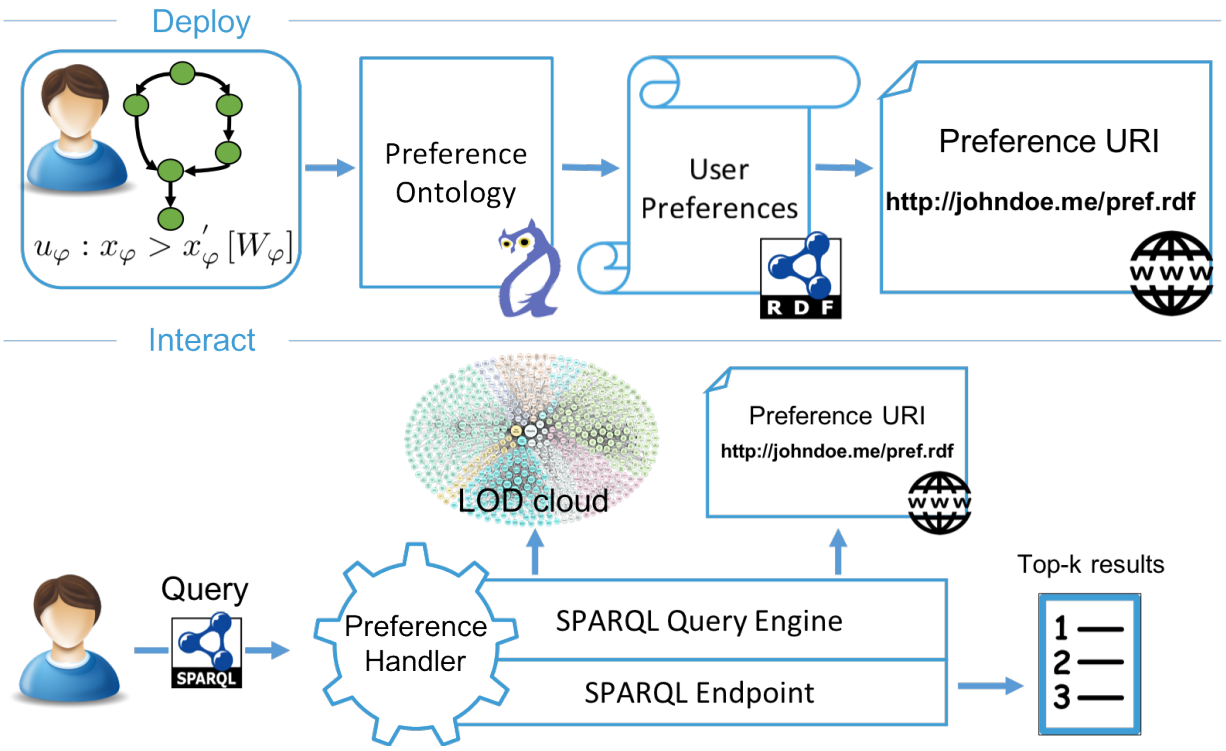

Fig. 1. A graphical representation of the proposed approach. The Deploy and Interact steps only rely on the adoption of standard technologies and languages.

data available in the Linked Data cloud. User preferences can be encoded and published in an RDF file.

Interact.

- *Use SPARQL to get user preferences*. A preference handler loads[3] the preference model of the user encoded in RDF by means of a general purpose SPARQL query engine.
- *The preference handler formulates SPARQL 1.1 queries able to retrieve and order resources by taking into account user preferences*. Standard SPARQL queries are formulated in order to rank the result set with reference to the preferences expressed by the user.

A strong requirement that we had in mind while developing our solution was to use only standard Web technologies and languages to implement the overall framework. Indeed, we selected RDF to model user preferences and the power of SPARQL 1.1 to perform preference-based reasoning in order to rank the results of a query. In fact, as we will see in Section 4.2, advanced features of the last version of the SPARQL query language can be employed to perform preference reasoning over a model based on CP-theories. In Section 2, all the details needed to implement the Deploy and Interact steps in a pure Linked Data setting will be provided.

3 CP-theories

Utility functions can be considered as the ideal tool for representing and reasoning with preferences, but the total order that they allow to represent does not always reflect the actual user model. Partial orders among preferences are a more natural way to represent a user's tastes. Qualitative statements, e.g., "*given u, I prefer x_i over x'_i*", permit a system to encode partial orders among user preferences, thus granting the representation of a more realistic user model. Let us consider the following example. The previous example is a typical conditional statement where the core notion

[3] See *SPARQL 1.1 Update* specification at https://www.w3.org/TR/sparql11-update/#load.

of a CP-statement is explicitly represented. We have that when a particular condition u is true, a user prefers to enjoy items where also x_i is true, rather than items where x_i' is true, given that x_i and x_i' cannot be true at the same time.

Example 1 (Books)
"Giorgio has just finished his exams and wants to relax with a book. Giorgio can read both English and French, but he would like to improve his French to enrich his curriculum and so he prefers to read French books. Giorgio prefers reading crime books over autobiographical ones for French books, since he believes that crime plots are more captivating and therefore more useful while learning a foreign language. The reverse order holds for English books. Giorgio is a good reader and, therefore, given an English book, he prefers those being part of a saga. The literary genre and the presence of a subsequent work have not the same importance to Giorgio: in case of English books, he considers the choice on the genre more important than the one dependant on sequels, while the opposite happens for French books. Finally, for books characterized by a sequel, Giorgio regards positively the presence of a cinematographic version". ■

By looking at Example 1, we find it quite hard (even impossible) to directly represent Giorgio's preferences by means of a score assigned to his preferential statements. In fact, Giorgio's preferences can be better expressed as qualitative (pairwise) comparisons.

Relevant frameworks to represent and reason with qualitative preferences are built according to the *ceteris paribus* semantics [28] and specifically consist of conditional preference networks (or CP-nets) [3] and a formalism along similar lines to CP-nets, but with a richer language of preference statements, namely, conditional preference theories (or CP-theories) [44].

Syntax. Formally, given a set of variables V, a CP-theory Γ is a set of preference statements φ of the general form:

$$u_\varphi : x_\varphi > x_\varphi^{'} \, [W_\varphi],$$

where u_φ is an assignment to a set of variables $U_\varphi \subset V$, x_φ and $x_\varphi^{'}$ are different assignments to some variable $X_\varphi \notin U_\varphi$, and W_φ is some subset of $V - U_\varphi - \{X_\varphi\}$.

Semantics. The interpretation of φ is that, given u_φ, x_φ is strictly preferred to $x_\varphi^{'}$, all else being equal, but irrespective of the values of variables in W_φ.

φ_1	$\top : C_F > C_{UK}[\emptyset]$
φ_2	$C_{UK} : LG_A > LG_C[SW]$
φ_3	$C_F : SW_{No} > SW_{Yes}[LG]$
φ_4	$C_{UK} : SW_{Yes} > SW_{No}[\emptyset]$
φ_5	$C_F : LG_C > LG_A[\emptyset]$
φ_6	$SW_{Yes} : F_{Yes} > F_{No}[\emptyset]$
φ_7	$SW_{No} : F_{No} > F_{Yes}[\emptyset]$

Table 1
The CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$.

That is, φ compactly states that for all assignments w, $w^{'}$ to W_φ and assignments t to $T_\varphi = V - U_\varphi - \{X_\varphi\} - W_\varphi$, $tu_\varphi x_\varphi w$ is preferred to $tu_\varphi x_\varphi^{'} w^{'}$.

In what follows, we will use the word *outcome* to indicate a complete assignment to all the variables in V and denote the set of all outcomes as $\mathcal{O}$. For the statement φ, we denote by φ^* the set of pairs of outcomes $(tu_\varphi x_\varphi w, tu_\varphi x_\varphi^{'} w^{'})$, where t is an assignment to T_φ, and $w, w^{'}$ are assignments to W_φ. Further defining $\Gamma^* = \cup_{\varphi \in \Gamma} \varphi^*$, it is then natural to define for the CP-theory Γ a strict partial order $>_\Gamma$, induced by Γ on the set of outcomes $\mathcal{O}$, as the transitive closure of Γ^*. The CP-theory formalism allows to express the usual CP-net ceteris paribus statements by simply considering $W_\varphi = \emptyset$ and identifying U_φ with $Pa(X)$, the parents of a variable X, that is, variables which the preferences on X depend on. However, as anticipated in Section 1, under the stricter CP-net formalism, for each variable X, a parent set $Pa(X)$ must be defined and instantiated when the preference order over values of X is established. This is not required in CP-theories, where you can find more statements related to the same variable X, but with different sets U. In addition, CP-theories allow stronger conditional preference statements than CP-nets, which are natural for users to express. For example, they represent a formalism even more general than TCP-nets [4], an enhancement of CP-nets where (conditional) relative importance between variables can be expressed. TCP-nets can be represented through statements with W containing at most one variable. Moreover, there are statements, such as *I prefer x_i over $x_i^{'}$ irrespective of the values of all other variables*, that cannot be expressed in CP-nets or TCP-nets, but correspond in the new formalism of CP-theories to $\top : x > x^{'} \, [V - \{X\}]$, where $\top$ is the assignment to an empty set of variables.

Example 2 (Books cont'd)
The overall profile of Giorgio may be modelled by means of the CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$, that is, the

set of statements given in Table 1. There, a set of binary variables $V = \{Country, LiteraryGenre, SubsequentWork, FilmVersion\}$ (abbreviated as C, LG, SW, and F, respectively) is considered. Their domains are given by:

- $dom(Country) = \{C_F, C_{UK}\}$ (for *France* and *United Kingdom*),
- $dom(LiteraryGenre) = \{LG_C, LG_A\}$ (for $Crime\_fiction$ and $Autobiographical\_novel$),
- $dom(SubsequentWork) = \{SW_{Yes}, SW_{No}\}$ (indicating if a book has a sequel or not),
- $dom(FilmVersion) = \{F_{Yes}, F_{No}\}$ (indicating whether there is a cinematographic version of the book or not). ■

For a CP-theory Γ, the preference ranking over outcomes $>_\Gamma$ introduced above, can be equivalently induced under the *worsening swap* semantics. Hereafter, we use the notation $o(X_i) = x_i$ to indicate that the variable X_i is assigned the value x_i in o, and analogously $o(\{X_j, \ldots, X_{j+k}\}) = \{x_j, \ldots, x_{j+k}\}$ to state that $X_j = x_j, \ldots, X_{j+k} = x_{j+k}$ in o.

Given two outcomes o and $o^{'}$ of $\mathcal{O}$, there is a worsening swap from o to $o^{'}$, if there exist a variable $X_i \in V - \{X_j, \ldots, X_{j+k}\} - W, x_i, x_i' \in dom(X_i)$ and an assignment $x_j, \ldots, x_{j+k}$ to the variables set $\{X_j, \ldots, X_{j+k}\}$ such that:

(i) $o(X_i) = x_i$ and $o'(X_i) = x_i'$,
(ii) $o(\{X_j, \ldots, X_{j+k}\}) = o'(\{X_j, \ldots, X_{j+k}\}) = \{x_j, \ldots, x_{j+k}\}$,
(iii) $o(V - \{X_i\} - \{X_j, \ldots, X_{j+k}\} - W) = o'(V - \{X_i\} - \{X_j, \ldots, X_{j+k}\} - W)$, and
(iv) $x_j \ldots x_{j+k} : x_i > x_i'[W] \in \Gamma$.

The preference relation $>_\Gamma$ over $\mathcal{O}$ is therefore the transitive closure of worsening swaps.

A CP-theory Γ is *consistent*, if it has a model, i.e., if there exists a strict total order $>$ that *satisfies* Γ, which is equivalent to $> \supseteq \Gamma^*$, that is, $>$ extends $>_\Gamma$. In [43], it is proved that the irreflexivity of $>_\Gamma$ (or equivalently the acyclicity of Γ^*) is a necessary and sufficient condition for consistency.

The consistency of a CP-theory implies that there are no cycles generated by $>_\Gamma$. Avoiding cycles is of paramount importance, as they introduce conflicting information while ordering the outcomes in $\mathcal{O}$. Let us consider the ordering represented in Figure 2, where an edge from o_i to o_j represents $o_i >_\Gamma o_j$. As we cannot establish what is the correct ordering of the outcomes due to the cycle created by o_2 and o_3, we could even have situations where we cannot compute the most preferred (undominated) outcome. For example, suppose in Figure 2, we do not have o_1. What would then be the best solution for the user in this case?

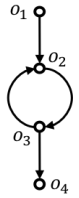

Fig. 2. An example of cycle among outcomes.

A necessary condition for consistency is local consistency. Consider a CP-theory Γ, a variable $X \in V$, and an assignment a to a set of variables $A \subseteq V$. An ordered pair $(x, x^{'})$ of X values is *validated* by a, if there exists a statement φ of the form $u : x > x^{'}\,[W]$ in Γ, such that a extends u, that is, a projected to U_φ gives u.

The *local ordering* $\succ_a^X\ (\Gamma)$ (abbreviated as $\succ_a^X$) on X values is defined as the transitive closure of the set of pairs $(x, x^{'})$ validated by a. Γ is *locally consistent*, if $\succ_\alpha^X$ is irreflexive for all variables X and outcomes α. Local consistency is a necessary condition for consistency, since if Γ is not locally consistent, then there exist an outcome α, a variable X, and a sequence $x_1, \ldots, x_k$ of values of X with associated statements $u_i : x_i > x_{i+1}[W_i] \in \Gamma$ such that α extends u_i and $\alpha(X) = x_1 = x_k$.

This gives a worsening swapping sequence from α to α (only involving changing variable X), thus implying that $>_\Gamma$ is not irreflexive, or equivalently that Γ is not consistent. In general, deciding whether a CP-theory is locally consistent is coNP-complete, but it can be shown that if the size of the parent sets and the size of the domain sets are bounded by a constant, then deciding local consistency is polynomial [44]. Moreover, for CP-nets and TCP-nets, local consistency is always guaranteed [44].

Given a CP-theory Γ, there are several kinds of directed graphs that can be defined on the set of variables V. For $S, T \subset V$, we indicate with $S \to T$ the set of edges $\{(X, Y) : X \in S, Y \in T\}$, omitting the set brackets, if S or T is a singleton set, e.g., abbreviating $S \to \{Y\}$ with $S \to Y$.

The *dependency graph* $H(\Gamma)$ consists of edges $U_\varphi \to X_\varphi$ for all φ in Γ, that is, all the pairs of the form (Y, X_φ), with $\varphi \in \Gamma$ and $Y \in U_\varphi$. That is, the

edge (Y, X) belongs to $H(\Gamma)$ iff there is some conditional preference statement $\varphi \in \Gamma$ that makes the preferences for X conditional on Y. Relative importance is not encoded in $H(\Gamma)$.

On the other side, we define $G(\Gamma)$ to contain $U_\varphi \rightarrow X_\varphi$ and $X_\varphi \rightarrow W_\varphi$ for all φ in Γ, i.e., $G(\Gamma) = H(\Gamma) \cup \{X_\varphi \rightarrow W_\varphi \mid \varphi \in \Gamma\}$. $G(\Gamma)$ contains both dependency and relative importance information: it is $H(\Gamma)$ with the addition of edges (X, Z), if there is any statements φ representing a preference on values of X irrespective of the value of Z (then, X is *more important* than Z, with importance meant as in the TCP-net formalism [4]).

A CP-theory Γ is *fully acyclic*, if $G(\Gamma)$ is acyclic. For fully acyclic CP-theories, consistency and local consistency are equivalent [44].

For a CP-theory Γ and assignment a to a set of variables $A \subseteq V$, we can define another directed graph $J_a(\Gamma)$ on V made of the set of edges $U_\varphi \rightarrow \{X_\varphi\} \cup W_\varphi$ for all $\varphi \in \Gamma$ and also the set $\{X_\varphi\} \rightarrow W_\varphi$ for all $\varphi \in \Gamma$ such that $U_\varphi \subset A$ and a extends u_φ.

A CP-theory Γ is *context-uniformly conditionally acyclic* (or *cuc-acyclic*), if it is locally consistent, and for each outcome $o \in \mathcal{O}$, $J_o(\Gamma)$ is acyclic. It can be proved that a cuc-acyclic CP-theory is always consistent [43]. The condition of cuc-acyclicity is weaker than the full acyclic one, and it requires only $H(\Gamma)$ to be acyclic [44] instead of $G(\Gamma)$.

Cuc-acyclicity implies a less expressive power in terms of preferences that the user may express, but, as we will see in the following, it grants a nicer computational complexity when ranking a set of outcomes (along with guaranteeing always consistency, while deciding the consistency of general CP-nets and CP-theories is PSPACE-complete [24]).

Example 3
To better clarify the expressive limits of cuc-acyclic CP-theories, we consider the following CP-theory $\hat{\Gamma}$ whose $H(\hat{\Gamma})$ is shown in Figure 3.

(1) Crime Fiction : Agatha Christie > Andrea Camilleri [∅]
(2) Isaac Asimov : Science Fiction > Crime Fiction [∅]

Due to the interrelation between the two preferences, $H(\hat{\Gamma})$ is cyclic, and so $\hat{\Gamma}$ cannot be cuc-acyclic.

Although there might be cases where cuc-acyclicity is a strong limitation in the representation of user preferences, it represents a limited subset of all possible CP-theories that can be used to model a user profile. ■

In what follows, we show two results, both proved in [44], which determine a strict partial order extending

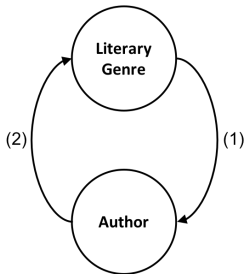

Fig. 3. The graph $H(\hat{\Gamma})$ representing preferences in Example 3.

$>_\Gamma$. The approaches proposed to compare outcomes are strongly related to the ordering queries defined in [3] and already discussed in [39] for CP-nets, and can be seen as a generalization of Corollary 4 and Theorem 5 of [3]. The first result is applicable for a fully acyclic CP-theory Γ. It proposes to compare two outcomes by looking (using the appropriate local ordering) at their value on each of the most important variables on which they differ, where importance is defined according to the graph $G(\Gamma)$.

In Theorem 1, we denote with $\Delta(\alpha, \beta)$ the set of variables of V on which outcomes α and β differ, i.e., $\Delta(\alpha, \beta) = \{Y \in V | \alpha(Y) \neq \beta(Y)\}$. If $\alpha \neq \beta$, we build $\Theta(\alpha, \beta)$ as the set of $G^{'}$-maximal elements of $\Delta(\alpha, \beta)$, being $G^{'}$ the transitive closure of $G(\Gamma)$. $\Theta(\alpha, \beta)$ is therefore the set of variables $Y \in \Delta(\alpha, \beta)$ such that there exists no $Z \in \Delta(\alpha, \beta)$ with $(Z, Y) \in G^{'}$.

Theorem 1
Let Γ be a locally consistent and fully acyclic CP-theory, and let the binary relation $\succ_{p(\Gamma)}$ on $\mathcal{O}$ be defined as follows, for any pair of outcomes α and β: $\alpha \succ_{p(\Gamma)} \beta$ iff $\alpha \neq \beta$ and $\alpha(Y) \succ_\alpha^Y \beta(Y)$ for all $Y \in \Theta(\alpha, \beta)$. Then, $\succ_{p(\Gamma)}$ is a strict partial order extending $>_\Gamma$, and the comparison between any pair of outcomes requires polynomial time.

The second result deals with cuc-acyclic CP-theories [44]. In Theorem 2, $\Delta(\alpha, \beta)$ is still used to denote the set of different variables in outcomes α and β. If $\alpha \neq \beta$, $\Theta^{'}(\alpha, \beta)$ is defined as the set of $\triangleright_\alpha$-undominated elements of $\Delta(\alpha, \beta)$, being $\triangleright_\alpha$ the transitive closure of $J_\alpha(\Gamma)$: $Y \in \Theta^{'}(\alpha, \beta)$ iff there exists no Z in $\Theta^{'}(\alpha, \beta)$ with $(Z, Y) \in \triangleright_\alpha$.

Theorem 2
Let Γ be a cuc-acyclic CP-theory, and let the binary relation $\gg_\Gamma$ on $\mathcal{O}$ be defined as follows, for any pair of outcomes α and β: $\alpha \gg_\Gamma \beta$ iff $\alpha \neq \beta$ and $\alpha(Y) \succ_\alpha^Y \beta(Y)$ for all $Y \in \Theta^{'}(\alpha, \beta)$. Then, $\gg_\Gamma$ is

a strict partial order extending $\succ_\Gamma$, and the comparison between any pair of outcomes requires polynomial time.

Theorem 2 proposes a more general approach to generate a strict partial order on $\mathcal{O}$, which although requiring a pretty strong condition on the CP-theory, i.e., cuc-acyclicity, does not need full acyclicity. In particular, if Γ is fully acyclic, and α and β are two outcomes to compare, then $J_\alpha(\Gamma) \subseteq G(\Gamma)$ and so $\rhd_\alpha \subseteq G^{'}$. Therefore, $\Theta^{'}(\alpha, \beta) \supseteq \Theta(\alpha, \beta)$. This implies that if $\alpha \gg_\Gamma \beta$ then $\alpha \succ_{p(\Gamma)} \beta$, so that $\gg_\Gamma$ is a closer approximation of $\succ_\Gamma$ than $\succ_{p(\Gamma)}$.

Section 4.2 will ground on this more general Theorem 2 to formulate a `SPARQL` query able to rank outcomes according to user preferences encoded in a CP-theory model.

Example 4 (Books cont'd)
Relative to the CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$ with Giorgio's preferences (see Table 1), the use of Theorem 2 produces the following sound ranking solution:

$$\Big\langle\, C_F LG_C SW_{No} F_{No},$$
$$\Big(C_F LG_C SW_{No} F_{Yes}, C_F LG_A SW_{No} F_{No}\Big),$$
$$C_F LG_A SW_{No} F_{Yes}, C_F LG_C SW_{Yes} F_{Yes},$$
$$\Big(C_F LG_C SW_{Yes} F_{No}, C_F LG_A SW_{Yes} F_{Yes}\Big),$$
$$C_F LG_A SW_{Yes} F_{No}, C_{UK} LG_A SW_{Yes} F_{Yes},$$
$$C_{UK} LG_A SW_{Yes} F_{No}, C_{UK} LG_A SW_{No} F_{No},$$
$$C_{UK} LG_A SW_{No} F_{Yes}, C_{UK} LG_C SW_{Yes} F_{Yes},$$
$$C_{UK} LG_C SW_{Yes} F_{No}, C_{UK} LG_C SW_{No} F_{No},$$
$$C_{UK} LG_C SW_{No} F_{Yes} \,\Big\rangle,$$

where outcomes within round parentheses are not comparable.

As an example, we may consider the outcome $C_F LG_C SW_{No} F_{No}$which is favoured in the comparisons made according to the order $\gg_\Gamma$ over every other outcome. In fact, the set

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{No} F_{Yes})$$

coincides with the set of distinct variables in the compared outcomes, namely,

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{No} F_{Yes}),$$

being both equal to $\{F\}$, and the preference φ_7 can be used to locally order the first outcome over the second one. Analogously,

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{No} F_{No})$$

is equal to

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{No} F_{No})$$
$$= \{LG\},$$

and the preference φ_5 can be exploited. The set

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{No} F_{Yes})$$

coincides with

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{No} F_{Yes}$$
$$= \{LG, F\},$$

and the preferences φ_5 and φ_7 can be exploited for the variables LG and F, respectively. While

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{Yes} F_{Yes})$$
$$= \{SW, F\},$$

it holds that

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{Yes} F_{Yes})$$

is composed of the only variable SW, and the preference φ_3 can be used for this comparison. The preference φ_3 can also be used when dealing with

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{Yes} F_{No}),$$

which coincides with the set

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_C SW_{Yes} F_{No})$$
$$= \{SW\}.$$

As for Giorgio, *SubsequentWork* takes priority over *LiteraryGenre* for French books, according to φ_3, and takes priority over *FilmVersion*, because of the preference φ_6 (or φ_7), it follows that the set of distinct variables

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{Yes} F_{Yes})$$

has three elements $\{LG, SW, F\}$, while

$$\Theta^{'}(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{Yes} F_{Yes})$$
$$= \{SW\},$$

and the preference φ_3 can be used in this case. For the last comparison involving France, the preference φ_3 is still determinant, as

$$\Delta(C_F LG_C SW_{No} F_{No}, C_F LG_A SW_{Yes} F_{No})$$
$$= \{LG, SW\},$$

but $\Theta'(C_F LG_C SW_{No}\, F_{No}, C_F LG_A\, SW_{Yes} F_{No}) = \{SW\}$. When comparing the outcome $C_F LG_C SW_{No}\, F_{No}$ with any outcome o' in which C_{UK} appears, Θ' contains only the undominated variable $Country$, and the preference φ_1 can be used to advantage $C_F LG_C SW_{No} F_{No}$ over o'. ■

4 CP-theories and Linked Open Data

As we stated in Section 1, the target of this work is twofold. On the one hand, we want to supply the user with a vocabulary to represent qualitative statements formulated in terms of ceteris paribus semantics. On the other hand, we aim to provide an encoding of user preferences that can be used in a top-k query answering scenario. In this section, we start by proposing a first ontology that allows a system to represent preferential statements according to CP-theories in a very straightforward way and an extended version to manage the directed graph $J_o(\Gamma)$ on V introduced in Section 3 and exploited in Theorem 2. Note that `RDF` triples encoding the directed graph can be automatically derived from the original preferential statements. In Section 5, we provide a description of an implemented tool to infer the `RDF` version of the directed graph, starting from a set of conditional preferences. We deal with the complementary target in Section 4.2, where we show how to employ a user profile represented as an instantiation of the extended ontology to encode the corresponding preferences in a standard `SPARQL` query able to retrieve and rank resources in a personalized way.

Figure 4 shows the ontology that we modeled to express user profiles in terms of CP-theory statements[4]. The main idea behind the modeling of the ontology is that we may express preferences on properties of items that the user is looking for, e.g., `dbo:literaryGenre`, `dbo:country`, `dbo:subsequentWork`, or potentially `dbo:filmVersion`. Hereafter, the ontology in Figure 4 will be referred to as the *lite* ontology. The aim of the *lite* ontology is that of creating an ontological vocabulary providing all the elements to syntactically represent conditional preference statements in a theory Γ. By means of this ontology, it is possible to encode whatever preference φ in its general form $u_\varphi : x_\varphi > x'_\varphi\,[W_\varphi]$.

[4]The corresponding OWL file is available at `http://sisinflab.poliba.it/semanticweb/lod/ontologies/cpt_light.owl`

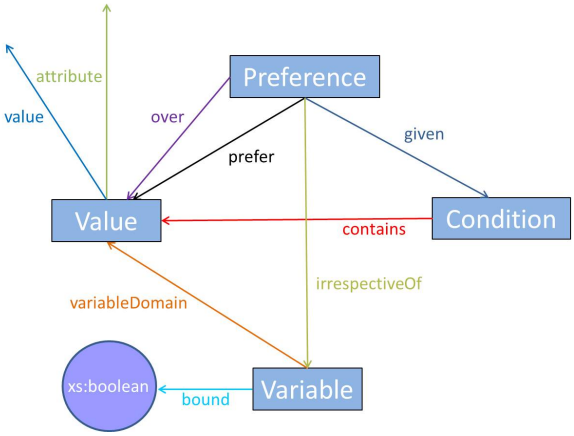

Fig. 4. A graphical representation of the *lite* version of the ontology proposed to represent conditional statements.

The ontology is composed by four main classes and nine properties. The class `Value` represents possible values of a variable. If we look at the book *Good-Knyght!* in Example 5, we see that the "actual values" for which the user expresses a preference are composed by both a *property* (e.g., `dbo:country`) and its related *object* (e.g., `db:United_Kingdom`). This is the reason why the class `Value` is the domain of the two properties `attribute` and `value`. The former mapping the property, the latter mapping the object. `Condition` is used to express the conditional part u_φ of a preference statement $u_\varphi : x_\varphi > x'_\varphi\,[W_\varphi]$, which is also the condition for the relative importance [4] of the variable X_φ over variables in W_φ in case $W_\varphi \neq \emptyset$. It is the domain of the property `contains`, whose range is `Value`. The class `Preference` represents the whole conditional statement φ. The properties having `Preference` as domain reflect the structure of the preferential statement "`given` a `Condition`, I `prefer` a `Value` `over` another `Value`, optionally `irrespectiveOf` some other variables". The class `Variable` is used to model the variables of a CP-theory, and it is domain of `variableDomain`, whose range is `Value`. Finally, we have the `bound` property needed to explicitly state if the `value` associated to an `attribute` is an actual value (as for `dbo:country` and `dbo:literaryGenre`) or if it represents the situation that we (do not) have a triple involving the `attribute`, as for `dbo:subsequentWork` or `dbo:filmVersion`. Here, we adopted the modeling choice of representing directly the conditions generated by the combination for the values of variables in U_φ instead of relating the variables them-

selves. As an example, in case we have "*Given an English book that is part of a saga, I prefer...*", with reference to the previous example, the corresponding encoding will be[5]

```
cpl:combined-cond a cpl:Condition ;
                  cpl:contains  cpl:c1 ;
                  cpl:contains  cpl:s1 .
```

Here, `cpl:combined-cond` represents u_φ as a whole, while `cpl:c1` and `cpl:s1` represent the values x_1 and x_2 composing $u_\varphi = x_1x_2$.
We will see how this modeling choice will be useful when embedding the CP-theory into a SPARQL query.

Notice that such classes and predicates are sufficient for the user to express the CP-theory with her preferential statements, and to facilitate the user experience even further, we built a user-friendly tool, where preferences related to a specific domain (e.g., books or movies) can be added. The aforementioned tool will be extensively described in Section 5.

Example 5 (Books cont'd)
With respect to Giorgio's preference "*given an English book, he prefers those being part of a saga*", if we look in DBpedia, we may find, for instance, the book *GoodKnyght!*. Indeed, we have:

```
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://dbpedia.org/ontology/>

db:GoodKnyght! a dbo:Book ;
dbo:country db:United_Kingdom ;
dbo:subsequentWork db:Whizzard! .
```

From the previous RDF statements, we see that Giorgio's preference refers to values of objects in a triple with reference to a specific predicate. Indeed, given a set of resources of type `dbo:Book` such that the value for `dbo:country` is `db:United_Kingdom`, he prefers those with an associated triple whose predicate is `dbo:subsequentWork`. In order to be fully compliant with the Linked Data technological stack, we need a vocabulary/ontology that allows users to represent their preferences on different attributes of resources that they might be interested in. Hence, with reference to the ontology in Fig. 4, we have the following RDF triples modeling the preference introduced at the beginning of this example.

```
@prefix cpl: <http://sisinflab.poliba.it/
 semanticweb/lod/ontologies/cpt_light.owl#>
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://dbpedia.org/ontology/>

### Variables ###
cpl:country a cpl:Variable;
cpl:bound true;
cpl:variableDomain cpl:c1,cpl:c2.

cpl:subsequentWork a cpl:Variable;
cpl:bound false;
cpl:variableDomain cpl:s1, cpl:s2.

### Values allowed for each variable ###
cpl:c1 a cpl:Value;
cpl:attribute dbo:country;
cpl:value db:United_Kingdom.

cpl:c2 a cpl:Value;
cpl:attribute dbo:country;
cpl:value db:France.

cpl:s1 a cpl:Value;
cpl:attribute dbo:subsequentWork;
cpl:value cpl:subsequentWorkYes.

cpl:s2 a cpl:Value;
cpl:attribute dbo:subsequentWork;
cpl:value cpl:subsequentWorkNo.

### Condition ###
cpl:cond a cpl:Condition;
cpl:contains  cpl:c1.

### Preference ###
cpl:pref a cpl:Preference;
cpl:given cpl:cond;
cpl:prefer cpl:s1;
cpl:over cpl:s2.
```

■

Once we have defined and modeled Γ in RDF, in order to compare two outcomes α and β as in Theorem 2, we may build the RDF version of the directed graph $\triangleright_\alpha$ on V (see Section 3).

To this aim, the *lite* ontology is extended as shown in Figure 5[6] to what we call the *full* ontology. Hence, starting from a set of preferences represented via the

[5]Here, we use the prefix `cpl` to denote `<http://sisinflab.poliba.it/semanticweb/lod/ontologies/cpt_light.owl#>`

[6]The corresponding OWL file is available at `http://sisinflab.poliba.it/semanticweb/lod/ontologies/cpt_full.owl`

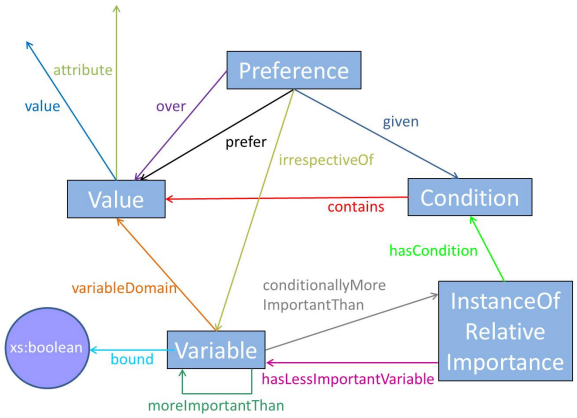

Fig. 5. A graphical representation of the *full* ontology proposed to represent conditional statements.

lite ontology, we derive its *full* version such that, once instantiated with an outcome α, it represents $\triangleright_\alpha$. The derivation step is performed by adding new statements via the following relations:

`moreImportantThan` has `Variable` both as domain and as range, and it is used to model the transitive closure of dependency and (unconditional) relative importance information, i.e., for the transitive closure of edges $U_\varphi \rightarrow \{X_\varphi\} \cup W_\varphi$, and, if $U_\varphi = \emptyset$, for $\{X_\varphi\} \rightarrow W_\varphi$, for any φ.

`conditionallyMoreImportantThan` takes into account conditional relative importance information. Its range is an instance of the new class `InstanceOfRelativeImportance`, which is used to represent a pair (u_φ, Z), $Z \in W_\varphi$, for a statement φ with $U_\varphi \neq \emptyset$. In fact, the class `InstanceOfRelativeImportance` is the domain of the property `hasCondition`, whose range is the `Condition` instance representing u_φ, and of the `hasLessImportantVariable` property, whose range is the `Variable` instance for Z. In the following, we say "X is `conditionallyMoreImportantThan` Y under the `Condition` C" when X is linked via `conditionallyMoreImportantThan` to an instance of the class `InstanceOfRelativeImportance`, which, in turn, is linked by `hasCondition` to a `Condition` C and by `hasLessImportantVariable` to a `Variable` Y.

Both properties must act as transitive relations. Therefore, we add more statements to the original preferences by means of the following rules involving the transitive closure of property `moreImportantThan` and then, for pairs of variables (Y, Z) not linked by it, the transitive closure of property `conditionallyMoreImportantThan`.

- if variable Y is `moreImportantThan` a variable X and at the same time variable X is `conditionallyMoreImportantThan` a variable Z, under a condition C not involving values of Y, then we add the RDF statements representing that "Y is `conditionallyMoreImportantThan` Z under the condition C";
- if Y is `conditionallyMoreImportantThan` a variable X under a condition C, X is at the same time `moreImportantThan` a variable Z, and the condition C does not involve any value of Z, then the additional fact we add is "Y is `conditionallyMoreImportantThan` Z under the condition C";
- if Y is `conditionallyMoreImportantThan` a variable X under a condition C, and X is `conditionallyMoreImportantThan` a variable Z under a condition C', then the additional fact added to the knowledge base is "Y is `conditionallyMoreImportantThan` Z under the condition C''", where $C'' = C \wedge C'$ is the condition joining C and C', but only if $C \wedge C'$ does not contains values of Y, Z or two different values of any other variable.

According to the properties just introduced, given a pair of outcomes (α, β), if there exist no variable $X' \in \Delta(\alpha, \beta)$ that is `moreImportantThan` X and no variable $X'' \in \Delta(\alpha, \beta)$ that is `conditionallyMoreImportantThan` (u, X), with α extending u, then a variable X is in $\Theta'(\alpha, \beta)$.

Example 6 (Books cont'd)
The *full* encoding corresponding to $\Gamma_{\mathrm{C-LG-SW-F}}$ for Giorgio's preferences in Table 1, is represented in Listing 1[7]. ■

```
@prefix cpt: <http://sisinflab.poliba.it/semanticweb/
 lod/ontologies/cpt_full.owl#>
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://dbpedia.org/ontology/>

cpt:country1 a cpt:Value;
             cpt:attribute dbo:country;
             cpt:value db:United_Kingdom.
cpt:country2 a cpt:Value;
             cpt:attribute dbo:country;
```

[7]For conciseness, the prefix `cpt` is always assumed in the following for `<http://sisinflab.poliba.it/semanticweb/lod/ontologies/cpt_full.owl#>`.

```
            cpt:value db:France.
cpt:genre1 a cpt:Value;
           cpt:attribute dbo:literaryGenre ;
           cpt:value db:Crime_fiction.
cpt:genre2 a cpt:Value;
           cpt:attribute dbo:literaryGenre ;
           cpt:value db:Autobiographical_novel.
cpt:sw1 a cpt:Value;
        cpt:attribute dbo:subsequentWork;
        cpt:value cpt:subsequentWorkYes.
cpt:sw2 a cpt:Value;
        cpt:attribute dbo:subsequentWork;
        cpt:value cpt:subsequentWorkNo.
cpt:film1 a cpt:Value;
          cpt:attribute dbo:filmVersion;
          cpt:value cpt:filmVersionYes.
cpt:film2 a cpt:Value;
          cpt:attribute dbo:filmVersion;
          cpt:value cpt:filmVersionNo.

cpt:conditionC1 a cpt:Condition;
                cpt:contains  cpt:country1.
cpt:conditionC2 a cpt:Condition;
                cpt:contains  cpt:country2.
cpt:conditionSW1 a cpt:Condition;
                 cpt:contains  cpt:sw1.
cpt:conditionSW2 a cpt:Condition;
                 cpt:contains  cpt:sw2.

cpt:country a cpt:Variable;
            cpt:bound true;
            cpt:variableDomain cpt:country1,cpt:country2;
            cpt:moreImportantThan cpt:literaryGenre;
            cpt:moreImportantThan cpt:subsequentWork;
            cpt:moreImportantThan cpt:filmVersion.
cpt:literaryGenre a cpt:Variable;
                  cpt:bound true;
                  cpt:variableDomain cpt:genre1,cpt:genre2;
                  cpt:conditionallyMoreImportantThan
                        cpt:instanceOfRelativeImportance1;
                  cpt:conditionallyMoreImportantThan
                        cpt:instanceOfRelativeImportance3.
cpt:subsequentWork a cpt:Variable;
                   cpt:bound false;
                   cpt:variableDomain cpt:sw1, cpt:sw2;
                   cpt:moreImportantThan cpt:filmVersion;
                   cpt:conditionallyMoreImportantThan
                        cpt:instanceOfRelativeImportance2.
cpt:filmVersion a cpt:Variable;
                cpt:bound false;
                cpt:variableDomain cpt:film1, cpt:film2.

cpt:instanceOfRelativeImportance1
      a cpt:instanceOfRelativeImportance;
      cpt:hasCondition cpt:conditionC1;
      cpt:hasLessImportantVariable cpt:subsequentWork.
cpt:instanceOfRelativeImportance2
      a cpt:instanceOfRelativeImportance;
      cpt:hasCondition cpt:conditionC2;
      cpt:hasLessImportantVariable cpt:literaryGenre.
cpt:instanceOfRelativeImportance3
      a cpt:instanceOfRelativeImportance;
      cpt:hasCondition cpt:conditionC1;
      cpt:hasLessImportantVariable cpt:filmVersion.

cpt:preference1 a cpt:Preference;
                cpt:prefer cpt:country2;
                cpt:over cpt:country1.
cpt:preference2 a cpt:Preference;
                cpt:given cpt:conditionC1;
                cpt:prefer cpt:genre2;
                cpt:over cpt:genre1;
                cpt:irrespectiveOf cpt:subsequentWork.
cpt:preference3 a cpt:Preference;
                cpt:given cpt:conditionC2;
                cpt:prefer cpt:sw2;
                cpt:over cpt:sw1;
                cpt:irrespectiveOf cpt:literaryGenre.
cpt:preference4 a cpt:Preference;
                cpt:given cpt:conditionC1;
                cpt:prefer cpt:sw1;
                cpt:over cpt:sw2.
cpt:preference5 a cpt:Preference;
                cpt:given cpt:conditionC2;
                cpt:prefer cpt:genre1;
                cpt:over cpt:genre2.
cpt:preference6 a cpt:Preference;
                cpt:given cpt:conditionSW1;
                cpt:prefer cpt:film1;
                cpt:over cpt:film2.
cpt:preference7 a cpt:Preference;
                cpt:given cpt:conditionSW2;
                cpt:prefer cpt:film2;
                cpt:over cpt:film1.
```

Listing 1: The RDF version of the CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$ in Table 1, according to the *full* ontology.

In Section 4.2, we will see how to pose a SPARQL query against the *full* version of Γ in order to compute $\gg_\Gamma$, thus retrieving a ranked list of semantic resources ordered according to a user's preferences.

4.1 The special case of CP-nets

Before moving to the description of how to encode a CP-theory in a SPARQL query for personalized results ranking, we point out that the ontological model just described subsumes the vocabulary introduced in [39] to represent CP-nets models. There, we proposed how to model the information encoded in a CP-net through an ontology and how to formulate the query able to order outcomes accordingly in a consistent way. On the other hand, we know that CP-nets are a special and simple case of CP-theories and, therefore, we want to show how the new ontology in Figure 5 can deal with a CP-net.

The theoretical result exploited in [39] (namely Corollary 4 of [3]) orders an outcome o over another one o', consistently with a CP-net $\mathcal{N}$, if there exists a variable X such that o and o' assign the same values to all ancestors of X in $\mathcal{N}$ and given the assignment provided by o (and o') to the parents of X, i.e., $Pa(X)$, o assigns a more preferred value to X than that assigned by o' (according to the conditional preference table of X). The sufficient condition of Corollary 4 of [3] may be reformulated asking for a variable X such that there does not exist any variable moreImportantThan X different in o and o', and, given the assignment provided by o (and o') to $Pa(X)$, o assigns a more preferred value to X than that assigned by o'. The predicate moreImportantThan, in fact, covers dependency information and, when applied to a CP-net, allows to define a set of variables coincident with the ancestor set. Moreover, for a CP-net, the predicate given for any instance of Preference ordering the values of a variable may be used to define the parent set of that variable. The ontological model proposed in [39] for CP-nets is hence subsumed by the

full ontology of Figure 5. On the other hand, if one wants to represent a CP-net according to the *full* ontology, one has to consider that the dependency information is the only kind of information required for CP-nets, because there is no relative importance encoded. This implies that the RDF version of the CP-net, in terms of the *full* ontology, would not contain any predicate conditionallyMoreImportantThan and irrespectiveOf or any instance of the class InstanceOfRelativeImportance.

4.2 Ordering SPARQL results via CP-theories

In the following, we assume that users are looking for the best k items satisfying some requirements and that the choice for the best ones is led by their preferences, formulated according to a CP-theory Γ, on a set of variables $V = \{X_1,...,X_n\}$. Hence, we aim at solving a top-k query answering problem, where the ordering criterion is encoded in Γ. In the presented approach, we concentrate on cuc-acyclic CP-theories, to preserve the nice computational properties introduced in Section 3 and exploit the algorithmic approach suggested by Theorem 2.

The ultimate goal of our proposal can be summarized by the following query formulated in a meta-language on top of SPARQL:

```
SELECT ?item
WHERE {
    ?item satisfies user requirements
}
ORDER BY Γ
LIMIT  k
```

Here, *user requirements* are represented by a SPARQL graph pattern where at least one triple has ?item as subject. In the following, we use the notation $\mathcal{R}(?\texttt{item})$ to denote the user requirements associated with the variable ?item.

Example 7 (Books cont'd)
"*Giorgio really wants to relax, and so he is looking only for books with more than 300 pages*". In this case, Giorgio's requirements $\mathcal{R}(?\texttt{item})$ are represented by:

```
?item a dbo:Book.
?item dbo:numberOfPages ?page.
FILTER(?page>300).
```
■

The computation of an answer to the previous query, goes through the exploitation of the *full* version of Γ. The overall approach is composed of two main steps.

Step 1. Here, we compute a representation of α and β, starting from each $\varphi \in \Gamma$ and, by exploiting Theorem 2, an ordering based on $\gg_\Gamma$ for all possible outcomes is eventually returned. The representation of α and β as URIs goes through a string concatenation (we use the GROUP_CONCAT aggregate function of SPARQL).

Step 2. This step deals with ranking the items matching $\mathcal{R}(?\texttt{item})$ according to $\gg_\Gamma$ as computed in the previous step.

Both steps are detailed in the following.

Ordering the Outcomes (Step 1). From Theorem 2 in Section 3, we know how to build a strict partial order on a set of outcomes $\mathcal{O}$ extending $>_\Gamma$ by comparing outcomes via $\gg_\Gamma$. By means of the same meta-language that we used before, the ordering of outcomes can be done via the following query. There, we see the outcomes are ordered according to a counter representing the number of outcomes that they are able to $\gg_\Gamma$-dominate.

```
SELECT ?outcome-Dominating
       (COUNT(?outcome-dominated) AS ?counter)
WHERE {
  FILTER { ?outcome-Dominating ≫Γ ?outcome-dominated }
}
GROUP BY ?outcome-Dominating
ORDER BY  DESC(?counter)
```

In order to compute values for the two variables ?outcome-Dominating and ?counter, the previous query should act on one pair (α, β) per time by checking if $\alpha \gg_\Gamma \beta$.

The preference-based reasoning is performed exclusively by means of the SPARQL 1.1 query *Ordering-Query* whose generation is detailed in Algorithm 1 of Appendix A. The algorithm takes as input the *full* version of Γ and computes a query able to return a list of outcomes ordered according to ?counter. In particular, the query returns for each outcome, a numerical score representing its position in the ranking imposed by $\gg_\Gamma$.

For a better clarification, the reasoning procedure under the comparison between the pair (α, β) is summarized in the following:

1. The query computes the set $\Theta'(\alpha, \beta)$ by considering the variables X in the set $\Delta(\alpha, \beta)$ for which there do not exist: (i) a variable $X' \in \Delta(\alpha, \beta)$ linked to X by property cpt:moreImportant-

Than and (ii) a variable $X'' \in \Delta(\alpha, \beta)$ which is cpt:conditionallyMoreImportantThan than X under a condition extended by α.
2. It then counts the number of variables X from the set $\Theta'(\alpha, \beta)$ that let to state $\alpha(X) \succ_\alpha^X \beta(X)$ and compares it to the cardinality of $\Theta'(\alpha, \beta)$;
3. If the numerical values coincide, which means that for each variable X in $\Theta'(\alpha, \beta)$, $\alpha(X) \succ_\alpha^X \beta(X)$ holds, then the query concludes that $\alpha \gg_\Gamma \beta$.

In order to get all the information needed to check $\alpha(X) \succ_\alpha^X \beta(X)$ from the *full* version of Γ, *OrderingQuery* embeds Query 1 and Query 2 reported in the following. They return a set of quadruples $\langle$?V, ?ConcatenatedParent, ?Prefer, ?Over$\rangle$, with ?ConcatenatedParent optionally not instantiated, able to locally order α over β with respect to variable ?V.

Given a variable $X_i \in V$ with $dom(X_i) = \{x_{i1}, x_{i2}, ..., x_{in}\}$, we use the following notation relative to the corresponding instances cpt:x_{i1}, cpt:x_{i2},..., cpt:x_{in} of the class cpt:Value:

- $value(x_{ij})$ is the object of the triple cpt:x_{ij} cpt:value object;
- $attribute(x_{ij})$ denotes the object of the triple cpt:x_{ij} cpt:attribute object;
- we call *representative string* of x_{ij} the concatenation of the two strings represented by $attribute(x_{ij})$ and $value(x_{ij})$ respectively. The combination of $attribute(x_{ij})$ and $value(x_{ij})$ is used to represent x_{ij}, as they uniquely identify a value in the domain of a variable. Indeed, in case we used only $value(x_{ij})$, ambiguous situations could arise when it is used in combination with different attributes.

Finally, for an instance cpt:c of the class cp:Condition, we call *conditional values* of cpt:c all the objects of the triples cpt:c cpt:contains object.

Query 1

```
1  SELECT ?V
2    (concat(str(?attrPrefer),str(?valuePrefer)) as ?Prefer)
3    (concat(str(?attrPrefer),str(?valueOver)) as ?Over)
4  WHERE
5  {
6   ?preference cpt:prefer ?p;
7               cpt:over ?o.
8   FILTER NOT EXISTS {?preference cpt:given ?condition.}
9   ?V cpt:variableDomain ?p.
10  ?p cpt:attribute ?attrPrefer;
11     cpt:value ?valuePrefer.
12  ?o cpt:value ?valueOver.
13 }
```

Query 1 processes elements φ of Γ with $u_\varphi = \top$. Within the query, they are represented by the variable ?preference. The selection is made possible by the FILTER NOT EXISTS on the pattern {?preference cpt:given ?condition.} (line 8). Considering that the objects of properties cpt:prefer and cpt:over must be distinct values of the same variable, the query firstly extracts the variable that the preference acts on, i.e., ?V (line 1 and line 9). Then, it computes the *representative strings*, ?Prefer and ?Over (lines 2–3) for the objects ?p and ?o of the two triples involving cpt:prefer and cpt:over (lines 10–12).

Query 2

```
1  SELECT DISTINCT ?V  ?ConcatenatedParent ?Prefer ?Over  WHERE{
2    SELECT ?condition ?V
3     (GROUP_CONCAT(concat(str(?attr),str(?value)); separator="")
4      as ?ConcatenatedParent)
5     (concat(str(?attrPrefer),str(?valuePrefer)) as ?Prefer)
6     (concat(str(?attrPrefer),str(?valueOver)) as ?Over)
7    WHERE
8    {
9     ?preference cpt:given ?condition;
10                cpt:prefer ?p;
11                cpt:over ?o.
12    ?V cpt:variableDomain ?p.
13    ?p cpt:attribute ?attrPrefer;
14       cpt:value ?valuePrefer.
15    ?o cpt:value ?valueOver.
16    ?condition cpt:contains ?c.
17    ?c cpt:attribute ?attr;
18       cpt:value ?value.
19   }
20   GROUP BY ?condition ?V ?attrPrefer ?valuePrefer ?valueOver
21 }
```

Differently from the previous query, Query 2 is used to process statements φ belonging to Γ with $u_\varphi \neq \top$. The selection is made via the pattern {?preference cpt:given ?condition.} (line 9). Let us consider first the nested subquery in lines 2–20. For each instance of class Preference, such query extracts the variable ?V that the preference is about (lines 2 and 12) and considers the cpt:given condition ?condition (line 9), extracting its corresponding *conditional values* (line 16). The *representative strings* of such *conditional values* are then computed (lines 17–18) and concatenated at lines 3–4 in ?ConcatenatedParent, grouping by condition. The variables ?Prefer and ?Over are defined similarly to Query 1. The external query is just used to restrict the result set to variables ?V, ?ConcatenatedParent, ?Prefer, ?Over.

Ordering the Items (Step 2). Given the information on outcomes returned by the *OrderingQuery* at the previous step, on both values of variables and position in the ranking, an external RDF dataset, e.g., DBpedia,

may be queried, asking for items satisfying the hard constraints ($\mathcal{R}(?\texttt{item})$) imposed by the user and such that, when limiting the attention on variables in V, they match the description of an outcome. Items are then ordered according to the ranking over corresponding outcomes.

We are well aware that the one we propose is just a possible rewriting of a CP-theory in a SPARQL query and other encodings are possible, ever more efficient from a computational perspective. Moreover, we may see that the performance of the overall approach decreases when the size of variable domains grows and, in its current version, the approach is not able to handle continuous domains as for distance and time. Nevertheless, we believe that the proposed approach is a good starting point to reason with preferences in a pure Linked Data environment, as it is a straight implementation of theoretical results coming form previous works [44].

4.3 Instantiation of the framework

The procedure to retrieve items ordered according to user's preferences is made up of four phases:

- the *loading* of user's preferences;
- an *insert* to add information about outcomes;
- the execution of a *federated query*;
- the (optional) *dropping* of user's preferences.

First of all, the user's preferences file representing the *full* version of Γ is loaded in the SPARQL server through a LOAD operation and becomes the user's graph of preferences $\mathcal{G}_{user}$.

Example 8 (Book cont'd)
If the path to the RDF file encoding Giorgio's preferences (see Listing 1) is generally denoted as *path_to_ttl_file*, the load operation is executed as follows:

```
prefix g:<http://sisinflab.poliba.it/semanticweb/graphs/>

LOAD path_to_ttl_file INTO GRAPH g:Giorgio_preferences
```

■

The *OrderingQuery* able to order the outcomes according to $\gg_\Gamma$ is then executed. The information returned by the *OrderingQuery* is used to integrate the graph of user preferences $\mathcal{G}_{user}$ with additional triples on outcomes. Specifically, we add information about the score of an outcome and its description. Hence, the following triples are defined for each outcome:

- a triple satisfying the pattern

  `?URIOutcome cpt:hasScore ?score`

- a set of triples instantiating the pattern

  `?URIOutcome cpt:hasValueForX ?ValueForX`

  for every variable X of Γ.

Such information are added to $\mathcal{G}_{user}$ through an INSERT query.

Example 9 (Book cont'd)
For the CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$ with Giorgio's preferences, the INSERT operation would behave as follows:

```
prefix cpt:<http://sisinflab.poliba.it/semanticweb/
 lod/ontologies/cpt_full.owl#>
prefix dbo:<http://dbpedia.org/ontology/>
prefix db:<http://dbpedia.org/resource/>
prefix g:<http://sisinflab.poliba.it/semanticweb/graphs/>

INSERT { GRAPH g:Giorgio_preferences
  {?URIOutcome  cpt:hasScore  ?counter .
   ?URIOutcome  cpt:hasValueForCountry ?country_D;
                cpt:hasValueForLiteraryGenre ?genre_D;
                cpt:hasValueForSubsequentWork ?subwork_D;
                cpt:hasValueForFilmVersion ?filmVersion_D.
  }
}
where { GRAPH g:Giorgio_preferences {

  OrderingQuery

  }
}
```

Here, *OrderingQuery* denotes the ordering query over outcomes returned by Algorithm 1 in Appendix A. The output of Algorithm 1 applied to Listing 1 is available in Appendix B. ■

The next step is the execution of a federated query, composed by two subqueries. The first subquery retrieves the items satisfying the requirements imposed by the user, i.e., $\mathcal{R}(?\texttt{item})$, and for each item it looks for the values of variables in Γ. The retrieval of values grounds on the VALUES construct for variables that are `cpt:bound true` and on the combination of BIND, IF and EXISTS otherwise. The second subquery on the user's preferences graph $\mathcal{G}_{user}$, retrieving for each outcome its score and the variables values. A matching between items and outcomes is hence performed through such values and the items are finally ordered according to the position in the ranking of the relative outcome.

The main reason behind the federation of two (or more) endpoints is that: while the graph containing the RDF version of the user's preferences is encoded in the corresponding document (available at *Preference*

URI in Fig. 1), all the information about the items that we want to retrieve and rank is encoded in a separate dataset, e.g., DBpedia. The main assumption here is that the user's preferences are expressed with respect to a reference dataset/vocabulary, which can queried via a SPARQL endpoint.

Example 10 (Book cont'd)
Suppose that Giorgio is interested in the top-5 list of books matching his hard constraints (see Example 7), ordered according to his preferences encoded in the CP-theory $\Gamma_{\mathrm{C-LG-SW-F}}$ of Table 1. The federated query to carry out the searching task would be as follows:

```
prefix cpt:<http://sisinflab.poliba.it/semanticweb/
 lod/ontologies/cpt_full.owl#>
prefix dbo:<http://dbpedia.org/ontology/>
prefix db:<http://dbpedia.org/resource/>
prefix g:<http://sisinflab.poliba.it/semanticweb/graphs/>

SELECT ?item_D ?score WHERE {
  {SERVICE <http://dbpedia.org/sparql>  {
      SELECT DISTINCT ?item_D ?genre_D ?country_D
       ?subwork_D?filmVersion_D WHERE{
         ?item_D  a  dbo:Book;
                  dbo:numberOfPages ?page_D.
         FILTER(?page_D>300).
         ?item_D  dbo:literaryGenre ?genre_D;
                  dbo:country ?country_D.
         VALUES (?genre_D)   {
           (db:Crime_fiction)
           (db:Autobiographical_novel)
         }
         VALUES (?country_D) {
          (db:France)
          (db:United_Kingdom)
         }
         BIND(IF(EXISTS{?item_D dbo:subsequentWork ?object},
            cpt:subsequentWorkYes, cpt:subsequentWorkNo
            AS ?subwork_D).
         BIND(IF(EXISTS{?item_D dbo:filmVersion ?object},
            cpt:filmVersionYes,cpt:filmVersionNo)
            AS ?filmVersion_D).
      }
   }
  }
  {graph g:Giorgio_preferences {
    SELECT  ?score ?genre_D ?country_D ?subwork_D
            ?filmVersion_D  WHERE{
      ?s  cpt:hasScore ?score;
          cpt:hasValueForCountry ?country_D;
          cpt:hasValueForLiteraryGenre ?genre_D;
          cpt:hasValueForSubsequentWork ?subwork_D;
          cpt:hasValueForFilmVersion ?filmVersion_D.
    }
   }
  }
}
ORDER BY DESC(?score)
LIMIT 5
```

■

Finally, the graph with the user's preferences $\mathcal{G}_{user}$ can be optionally eliminated with a DROP operation.

Please note that all SPARQL queries are executed in a *simple entailment between RDF graphs* on the *full* version of Γ as well as on the external dataset for the federated query. Hence, all the queries are executed under the *RDF Entailment Regime* of SPARQL.

Example 11 (Book cont'd)
The graph related to Giorgio's preferences may be dropped as follows:

```
prefix g:<http://sisinflab.poliba.it/semanticweb/graphs/>

DROP GRAPH  g:Giorgio_preferences
```

■

5 Application

We now describe a tool[8] implementing the framework described in previous sections and aimed at supporting the end-user in retrieving a list of semantic resources ordered according to her preferences formulated under the CP-theory formalism. The tool just asks for preferential statements formulated under the CP-theory formalism, i.e., "*given u_φ, x_φ is strictly preferred to x'_φ, all else being equal, but irrespective of the values of variables in W_φ*". With reference to the ontologies introduced in Section 4, this means that for this preliminary step of preference definition, the interested user only has to deal with classes and properties of the *lite* ontology of Figure 4. In particular, after the selection of the domain of interest, the user inserts her preferences as depicted in Figure 6. The interface manages both instances of variables `cpt:bound true` and `cpt:bound false`, as introduced in Section 4. In the former case, the variable which the preference is "about" and the couple of values separated by the word "over" must be introduced; in the latter case, the user has to specify whether the presence or the absence of a variable is preferred. Optionally, she can insert a "Condition" under which the above order holds and make explicit, in the "Irrespective" section, the set of variables for the (conditional) relative importance. She can insert as much preferences as she wants with the "add Another Preference" button or complete the insertion procedure with the "Insert Preferences" button.

When this second button is pressed, the tool takes care that the user has defined the transitive closure of those preferential statements related to multiple values of a variable. More specifically, if $\varphi_1 = u_{\varphi_1} : x > \hat{x}\,[W_{\varphi_1}]$ and $\varphi_2 = u_{\varphi_2} : \hat{x} > \bar{x}\,[W_{\varphi_2}]$ have

[8] The tool is available at `http://cptheorysparql.cloudapp.net:10002/`

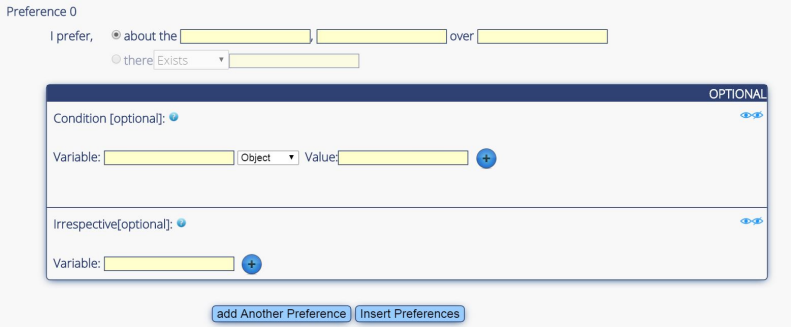

Fig. 6. Preferences Insertion.

been inserted, with x, $\hat{x}$, $\bar{x} \in dom(X)$, and u_{φ_1} and u_{φ_2} do not contain two different values of the same variable, then the rule $\varphi_3 = u_{\varphi_3} : x > \bar{x}\,[W_{\varphi_3}]$ is added, if missing (where u_{φ_3} is the condition joining u_{φ_1} and u_{φ_2}, and W_{φ_3} is the intersection of sets W_{φ_1} and W_{φ_2}). As an example in the movie domain, one may state that (φ_1) the actor Hugh Grant is preferred over Colin Firth for comedy films irrespective of the country of production and that (φ_2) Colin Firth is preferred over Joaquin Phoenix for movies directed by Woody Allen. In this case, the additional fact to add would be that (φ_3) Hugh Grant is preferred over Joaquin Phoenix for comedy movies directed by Woody Allen ceteris paribus and with no relative importance specification, since the intersection of sets W_{φ_1} and W_{φ_2} is empty. The tool then exploits the *lite* version of Γ to generate its *full* version by managing the transitive closure of `moreImportantThan` and `conditionallyMoreImportantThan`, as described in Section 4 [9].

The tool also helps the user to understand if her CP-theory is cuc-acyclic or not. It returns an error message to the user if the CP-theory is not locally consistent or the directed graph $J_{u_\varphi}(\Gamma)$ on V, for any u_φ introduced by the user in her preferential statements, is cyclic. Otherwise, it returns the encoding that can be used to query the `DBpedia` dataset.

As an alternative to the manual insertion of preferences, the user can decide to upload a file of preferences written according to the *lite* ontology, using the specific top right button. The transitivity and cuc-acyclicity checking and the introduction of the additional class and properties of the *full* ontology is performed in this case as well.

(a) (b)

Fig. 7. The buttons to display the Full `RDF` File (a) or to formulate a preference-based query (b).

At this point, a file with the *full* version of Γ is available and can be visualized by pushing the button in Figure 7 (a) or exploited directly to formulate a query against `DBpedia` through the button in Figure 7 (b).

More specifically, when the button of Figure 7 (b) is pressed, an interface as the one depicted in Figure 8 appears. The interface mimics the query presented at the beginning of Section 4.2. There, the users may insert their own requirements $\mathcal{R}(?\texttt{item})$ and specify the number k of results to use in the `LIMIT` modifier, which by default is set to 10. The URL displayed after the `ORDER BY` clause represents the location of the RDF file containing the *full* version of Γ.

The query returns the top-k list of items belonging to the domain of interest (e.g., books as in Figure 8), satisfying $\mathcal{R}(?\texttt{item})$ and ordered according to user's preferences.

Example 12 (Book cont'd)
Introducing Giorgio's preferences, contained in the CP-theory of Table 1, in the proposed tool would produce the top-5 list of results shown in Table 2. The results refer to the release of `DBpedia` 2015-04[10] (also known as: 2015 A).

By looking in `DBpedia`, one can observe an exact matching with the expected order shown in Example 4, according to the following triples:

[9] The rules that allow the system to manage the definition and the transitive closure of both properties `moreImportantThan` and `conditionallyMoreImportantThan` have been implemented in Prolog and are available at `http://sisinflab.poliba.it/semanticweb/lod/ontologies/rules.pl`.

[10] `http://wiki.dbpedia.org/dbpedia-data-set-2015-04`

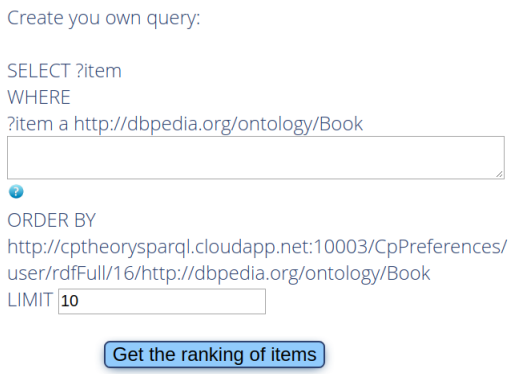

Fig. 8. The interface to build the query.

```
@prefix db: <http://dbpedia.org/resource/>
@prefix dbo: <http://dbpedia.org/ontology/>

db:An_Uncertain_Place  dbo:country  db:France ;
                       dbo:literayGenre  db:Crime_fiction .
db:Requiem_for_a_Fish  dbo:country  db:France ;
                       dbo:literayGenre  db:Crime_fiction .
db:Blood_Red_Rivers    dbo:country  db:France ;
                       dbo:literayGenre  db:Crime_fiction .
db:Tropic_of_Capricorn_(novel) dbo:country  db:France ;
                               dbo:literayGenre
                               db:Autobiographical_novel .
db:Have_Mercy_on_Us_All  dbo:country  db:France ;
                         dbo:literayGenre  db:Crime_fiction ;
                         dbo:subsequentWork
                         db:Wash_This_Blood_Clean_from_My_Hand.
```

■

?item_D	?score
db:An_Uncertain_Place	15
db:Requiem_for_a_Fish	15
db:Blood_Red_Rivers	15
db:Tropic_of_Capricorn_(novel)	13
db:Have_Mercy_on_Us_All	9

Table 2

The top-5 list of items retrieved for preferences in the CP-theory of Table 1.

6 Experiments

In order to asses the effectiveness of the presented approach and the implemented tool, we set up two different experiments.

The first experiment consisted of 20 real users using the tool to express their preferences. After the test, users were asked to fill up a questionnaire (reported in Table 5). The dataset adopted consisted of a subset of DBpedia 2015-04 related to the four popular domains of: Movies, Food, Music and Books. The statistics of the dataset used for experiments are detailed in Table 3.

Classes	Instances	Properties	1 hop resources
dbo:Book	31172	36	12964
dbo:Film	90063	31	82922
dbo:Food	6003	21	2367
dbo:Song	7195	27	2220
Total	**134433**	**115**	**100473**

Table 3

Dataset Statistics

It is worth noticing that CP-statements can also be automatically extracted from users data [35,36,31,34]. Nevertheless, we set up the previous experiment to have a hint on the average number of CP-statements φ needed to model a user profile as well as on what is, from a user perspective, the most tricky version of φ to represent among:

- $\top : x_\varphi > \hat{x}_\varphi[\emptyset]$,
- $u_\varphi : x_\varphi > \hat{x}_\varphi[\emptyset]$,
- $u_\varphi : x_\varphi > \hat{x}_\varphi[W_\varphi]$.

The second experiment consisted of simulating 168 users using the platform and expressing an overall number of 6720 preferences and 6720 queries to retrieve the resources ranked by taking into their preferences. The aim of this experiment was that of evaluating the response time of the overall system in retrieving a list of resources based on a set of user preferences.

6.1 *Test on Real Users*

In order to test the capability of a user to exploit the platform and even to test if human users unaware of CP-theories were able to express their preferences, we selected 20 users that did not know anything about CP-theories and, after a 5 minutes tutorial, we asked them to express their preferences by using our tool. We asked them to insert as many preferences as they wanted for each domain on the platform, and we then asked them to fill up a post-experience questionnaire in order to acquire some feedback about the experience. The motivation of this experiment is twofold: the first information that we wanted to collect was the number of preferences that a user is prone to explicitely express. The result for this evaluation is shown in Table 4. The users provided an overall number of 322 preferences. The average numbers per user are quite similar among the different domains (between 4 and 6) with a little higher propensity to express preferences over

books w.r.t. songs. The similar average values, and the similar standard deviations, suggest that there exists a commonality in the number of expressed preferences over a specific domain.

The second relevant information that we wanted to collect is how much the CP-theories expressiveness may fit a "natural" way of expressing preferences by a human being. To this aim, we submitted a small questionnaire with 10 questions whose relative answers in aggregate form are shown in Table 5 and Fig. 9. All the questions but Q.2 needed to express a value in a 5-star rating scale, with 1 being the worst answer and 5 the best one.

Users felt that representing preferences was not a trivial operation (3.2 corresponds to the lowest value of the overall questionnaire), but this perceived difficulty is clearly dependent on the type of preference (it is worth to notice that for every specific kind of preference, the score is higher than the overall score).

Thanks to the survey, we can list in an increasing order of difficulty the different kinds of preferences:

- *"About a property, I prefer a Value over another Value"* corresponding to $\top : x_\varphi > \hat{x}_\varphi[\emptyset]$.
- *"Given a condition, I prefer a Value over another Value"* corresponding to $u_\varphi : x_\varphi > \hat{x}_\varphi[\emptyset]$.
- *"About a property, I prefer a Value over another Value irrespectively to a property"* corresponding to $u_\varphi : x_\varphi > \hat{x}_\varphi[W_\varphi]$.

Moreover, if we look at the pie chart in Fig. 9, it emerges that the most difficult part of the process was to detect the properties (variables V) on which the preferences should be expressed. Another information that we wanted to collect was if the possible difficulty in expressing preferences is stable or it progressively vanishes as the number of expressed preferences increases. Questions 7, 8 and 9 show that the first preference was quite hard to express, but, as the experience goes on, it becomes much easier reaching an average value of 4.1.

The last relevant information that we wanted to collect is how much the expressiveness of CP-theories can correspond to a perceived "natural" way to express preferences. Also in this case, the result is interesting, because CP-theories are perceived as a quite good way of expressing preferences with a high value of 3.8.

6.2 *Test on Simulated Users*

In order to closely simulate the behavior of a real user, we designed a tool able to perform the classical operations of expressing a preference and asking the system for an ordered list of relevant resources. For each domain of interest, the simulated users randomly extract (with a uniform distribution) a property that they might be interested in, and then randomly select the other components of the preference (e.g., in case of a simple preference, $\top : x > \hat{x}[\emptyset]$, they select either the more liked resource x and the less liked one $\hat{x}$). The composed preference is then sent to the server to be processed and stored. The system checks if the preference produced a cycle, eventually warning the user (in case of a cycle, a new preference is produced). Once the preference is correctly inserted, the simulated user performs a query to the system to retrieve an ordered list of the 100 most relevant resources. The system continues, inserting a new preference for the same domain, and asking the system for a new list. The process ends when 10 preferences are inserted for each domain and the 10 related queries accomplished. Based on the previous experiment, we considered 10 as a representative number of preferences per user. Fig. 10 shows the average execution time for an increasing number of preferences related

	Total	Min	Max	Mean	Std Dev
dbo:Book	109	1	11	5.7368	2.1562
dbo:Film	78	1	17	4.5882	3.8900
dbo:Food	75	1	12	4.1667	2.6844
dbo:Song	60	1	12	3.5294	2.6485

Table 4
User Experiments Statistics

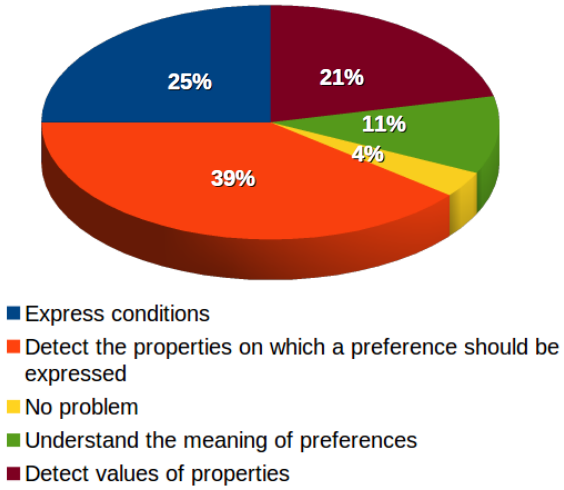

Fig. 9. Pie chart depicting the results of the second question of the survey.

Q.N.	Questions	Min	Max	Mean	St Dev
Q.1	*How easy has been to represent your preferences?*	1	5	3.1667	1.1100
Q.2	*Which among these did you consider the hardest?*	See Fig. 9			
Q.3	*How easy is to represent a preference like "About a property I prefer a Value over another Value"?*	1	5	4.1667	1.3048
Q.4	*How easy is to represent a preference like "I prefer a resource that has a certain property"?*	2	5	3.5556	1.0966
Q.5	*How easy is to represent a preference like "Given a condition I prefer a Value over another Value"?*	2	5	3.8889	0.9852
Q.6	*How easy is to represent a preference like "About a property I prefer a Value over another Value irrespectively to a property"?*	2	5	3.6111	1.2005
Q.7	*How easy was to represent the first preference?*	1	5	3.2222	1.3492
Q.8	*After the first preference how easy was to represent the next two ones?*	2	5	3.9444	0.9852
Q.9	*After the first three preferences how easy was to represent the next ones?*	1	5	4.1111	1.2783
Q.10	*How much this way of expressing preferences is similar to your own?*	1	5	3.7778	1.1448

Table 5
User Survey Statistics

to the simulated users.[11] The SPARQL engine adopted for the experimental evaluation is Jena Fuseki v. 2.3.1 running on a Linux server (kernel v. 4.4.0-28-generic) with an Intel Xeon @ 2.30GHz CPU and 8 GB RAM, while the local version of DBpedia had been loaded in a Virtuoso Server (v. 07.20.3212), running on a Linux server (kernel v. 4.2.0-23-generic) with an Intel Xeon @ 2.40 GHz CPU and 56 GB RAM.

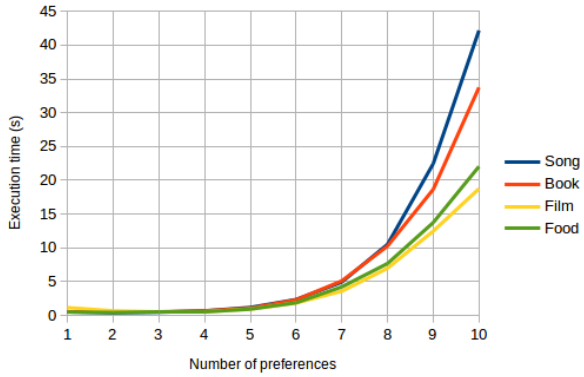

Fig. 10. Average execution time for increasing number of preferences (1 to 10) in the four domains of: Song, Book, Film and Food.

The results show that queries based on a number of preferences lower than six take approximately less than one second to return results to the user. This is even more interesting if we consider results of the previous experiments, where we saw that users tend to express an average number of preferences between 4 and 6.

[11] For those interested in a more fine-grained view of the data, a report of the execution times is publicly available at `https://github.com/sisinflab/CP-theories-SPARQL/blob/master/evaluation/evaluationResults.tsv`

7 Related works

The ability to infer, model, and reason with user preferences has been recognized as a prominent research direction in many fields, especially artificial intelligence (AI) [18,38]. Preferences are generally classified as quantitative, if they make use of a scoring function to assess an order over the available resources, resulting in a total order, or qualitative, if they are treated independently, resulting in incomparable resources and a partial preference order. Much work has focused on qualitative approaches, since these are closer to how people express their preferences; among the earliest logic-based approaches is von Wright's [42]. Following the overview over qualitative multi-attribute preference reasoning approaches provided by [37], in AI, there are, in particular, (i) methods adopting graphical structures to represent and reason about preferences, e.g., CP-nets [3] and TCP-nets [4]; (ii) methods that extend constraints satisfaction problems and incorporate soft constraints, as in the approximation of CP-nets with soft constraints described in [17,16]; and (iii) methods that use specific logic-based languages to represent qualitative preferences and derive utility functions, exploiting, e.g., machine learning techniques, such as support vector machines [14,15].

Conceptually close in spirit to the present paper is in particular [10], where ontological knowledge

expressed via existential rules in Datalog$^{\pm}$ is combined with CP-theories to represent qualitative conditional preferences along with domain knowledge, and to perform preference-based answering of conjunctive queries. Another related work [9] combines Datalog with CP-theories, but only considers atomic queries. The present paper, in contrast, focuses on SPARQL queries in a more restricted ontological context and conditional preferences specified via cuc-acyclic CP-theories. There is also a large body of work on handling preferences in logic programming, e.g., asprin [6], which is a framework for handling preferences among the stable models of a logic program. Similarly, the qualitative choice logic [5] is a propositional logic for representing a preference relation among models, which allows to specify alternative, ranked options for problem solutions. The above two and similar works on handling preferences in logic programming are fundamentally different from the present paper, as they are about preferences for ordering models of a logic program, rather than preferences for ordering the answers to a query subject to all models of a knowledge base.

Databases are another research area where preferences have been investigated. In a relational database management system, for example, the *top-k* (or *ranking*) queries represent a quantitative approach, since they return the k best matches according to a numerical score. In [33], a formalism supporting ranking queries for a relational database is presented. With reference to the qualitative approach instead, *skyline* queries [2] extend the notion of best matching to contexts, where multiple independent scores have to be taken into account. The result of a *skyline* query is a set of objects that are no worse than any other across all dimensions of a set of independent Boolean or numerical preferences [2]. Within the database community, both Chomicki [7,8] and independently Kießling and colleagues [29,30] formalized the first examples of *preference-based querying* languages, that is, extensions of SQL that support the specification of quantitative and qualitative queries.

The notion of preference is of primary importance also in the Linked Open Data context. The provision of means to enable users to look for data sources (e.g., SPARQL endpoints) and data content that is tailored to their individual preferences is one of the target of the original project by Tim Berners-Lee et al. Even the motivating example proposed in the introductory article about the Semantic Web [1] can be interpreted as a preference-based search, as extensively discussed in [40]. Based on this insight, in [40], the authors add preference-based querying capabilities to the most known Semantic Web query language, SPARQL. However, when the paper was published, it was not possible to specify multiple (independent) preference dimensions in SPARQL, and consequently the authors had to introduce the PREFERRING solution modifier. For example, the following query provides a preference-enabled SPARQL query for a user who is searching for an appointment, preferring excellent therapist, appointments out of the rush hour and later appointments over earlier ones, if both are equal with respect to the rush hour constraint.

```
1 SELECT ?appointment WHERE {
2  ?terapist :rated ?rating;
3            :offers ?appointment.
4  ?appointment :starts ?start;
5               :ends ?end.
6  PREFERRING (?rating = excellent AND
7   ?end < 1600 || ?start > 1800
8   CASCADE HIGHEST(?start))
9 }
```

At line 6, the PREFERRING clause behaves as a solution modifier, and the AND keyword separates independent preference dimensions. The CASCADE keyword at line 8 allows to give higher priority to the left-hand preference over the right-hand one. In their paper, the authors state that within the same SPARQL query, the use of a LIMIT k statement in combination with PREFERRING ones could inform the query evaluator to go deeper in the retrieval of skyline solutions, thus allowing the system to return a set of results ordered by user preferences.

A mapping operation between an OWL ontology, called *OWLPref*, and the SPARQL Preference syntax of [40] has been proposed by [32]. OWLPref allows for representing in a declarative, domain-independent and machine-interpretable way several kinds of preferences, namely, *SimplePreference*, *CompositePreference* (which makes compositions of the former), and *ConditionalPreference* (which models preferences that vary according to the context, thanks to a property *OnCondition*). However, considering the unavailability of conditional preferences in the SPARQL Preference syntax of [40], the use of *ConditionalPreference* in OWLPref seems of marginal utility.

The *PrefSPARQL* syntax of [27] keeps the goal of identifying the Pareto-optimal set, but introduces preferences at the level of filters. It still uses the AND to separate independent dimensions and to build what the authors call *MultidimesionalPref*. Each "dimension" is either a *conditional* preference (IF-THEN-ELSE) or an *atomic* preference, which in turn can be a sim-

ple expression or can involve more complex constructs [27]. Besides the support for conditional preferences and the substitution of CASCADE with PRIOR TO, the main innovative point of [27] with respect to [40] is perhaps that the proposed preference-enabled query can be completely rewritten using SPARQL 1.1 characteristics. In particular, [27] uses the FILTER NOT EXISTS. The translation of the previous query according to the *PrefSPARQL* query rewriting is given below.

```
1 SELECT ?appointmentA WHERE {
2  ?terapistA :rated ?ratingA;
3             :offers ?appointmentA.
4  ?appointmentA :starts ?startA;
5                :ends ?endA.
6  BIND ((?ratingA = :excellent) AS ?Pref1A)
7  BIND ((?endA < 16 || ?startA > 18:00) AS ?Pref2A)
8  BIND ((?startA) AS ?Pref3A)
9  FILTER NOT EXISTS {
10  ?terapistB :rated ?ratingB;
11             :offers ?appointmentB.
12  ?appointmentB :starts ?startB;
13                :ends ?endB.
14  BIND ((?ratingB = :excellent) AS ?Pref1B)
15  BIND ((?endB < 1600 || ?startB > 1800) AS ?Pref2B)
16  BIND ((?startB) AS ?Pref3B)
17  FILTER (
18   ((?Pref1B > ?Pref1A)  &&
19   !((?Pref2B < ?Pref2A) ||
20   (?Pref3B < ?Pref3A && ?Pref2B = ?Pref2A)))
21   ||
22   (!(?Pref1B < ?Pref1A)  &&
23   ((?Pref2B > ?Pref2A) ||
24   (?Pref3B > ?Pref3A && ?Pref2B = ?Pref2A))))}
25 }
```

The query looks for appointments ?appointmentA satisfying a certain pattern expressed in lines 2–5. The research is carried out checking that there is no ?appointmentB that verifies the same pattern (lines 10–13) and dominates ?appointmentA in any preference dimension. The example refers to only two independent preference dimensions and the situations when ?appointmentB dominates ?appointmentA are represented in the branches of || symbol at line 21, that is, lines 18–20 and lines 22–24. For the sake of completeness, ?appointmentB would dominate ?appointmentA if it was better in one dimension (line 18 or line 23–24) and no worse in the other one (line 19–20 or line 22). The PRIOR TO preference relation is encoded in lines 19–20 and 23–24 through the || operator.

Although *PrefSPARQL* allows the user to encode conditional preferences in a SPARQL 1.1 query, it differs from the approach that we presented here in at least three main aspects: (i) in *PrefSPARQL*, the focus is on computing the most preferred solution (undominated outcome), given a set of conditional preferences, while we provide a list of results ordered by user preferences; (ii) they deal with conditional preferences in the form $u_\varphi : x_\varphi > x'_\varphi$, while our approach is able to manage CP-statements in the form $u_\varphi : x_\varphi > x'_\varphi [W_\varphi]$, which result to be much more expressive for finite and discrete domains even in their cuc-acyclic version that we consider here; (iii) we provide an ontological vocabulary and a procedure to automatically encode preferences in a SPARQL query. However, thanks to its more agile structure, differently from our approach, *PrefSPARQL* allows the user to express preferences on variables with continuous domains as well as the usage of comparison operators.

In [41], the authors present SPREFQL, an extension of the SPARQL language that allows for appending a "PREFER" clause, which expresses soft preferences over the query results obtained by the main body of the query. The main ideas behind the approach are to associate relations of tuples with preference formulas, and to select the relations' most preferred tuples via a so-called winnow operator. Consequently, the approach does not allow for expressing conditional ceteris paribus preferences as in CP-theories.

As a general remark, previous SPARQL-related works on preference reasoning have been mainly devoted to preference representation and the retrieval of undominated outcomes. In principle, one may encode each of them in a procedural approach able to compute the first level of undominated solutions, then the second one and so one. At each iteration step, the procedure should be able to filter out the results coming from the "higher levels" of results computed in the previous steps.

Less closely related are approaches to preference-based query answering over graph databases. In particular, [22] presents regular languages for graph queries, where answers are partially ordered via a partial order on the strings of the languages. In the same vein, [25] introduces preferential regular path queries for enhanced querying of semi-structured databases. Query symbols are annotated with preference weights for scaling up or down the importance of matching a symbol against a database edge label. The paper studies (progressive) query answering, (certain) query answering in LAV data-integration systems, and query containment and equivalence. A similar approach in [20] introduces a graph query language that enables to declaratively express preferences. None of the above approaches to preference-based query answering over graph databases (which are intuitively based on (potentially recursive) pattern-recognition-style regular expressions), however, allows for expressing conditional

ceteris paribus preferences as in CP-theories. Less closely related are also information retrieval systems based on manipulating fuzzy truth values (which may also be interpreted as quantitative preferences), such as the fuzzy multimedia retrieval system in [19].

8 Conclusion and future work

In this paper, we have shown how user preferences can be taken into account while querying Linked Open Data datasets. Having realized that the Pareto-optimal set identification is not enough, we moved beyond it, proposing an approach to retrieve a ranked list of semantic resources, ordered according to a user's soft constraints. We focused on qualitative preferences, which are closer to how a user makes decisions, especially in a multi-attribute context, and integrated the partial order implied by a qualitative approach into a top-k scenario, that is, returning to the user, who is formulating qualitative preferential statements, a ranked list of resources, optionally limited in its size, ordered according to her preferences. Among qualitative approaches to preference reasoning, we relied on CP-theories, a general and well-known formalism based on the ceteris paribus semantics. We proposed an ontological vocabulary to model CP-theories by means of `RDF` statements under the ceteris paribus semantics. Then, we presented an algorithm able to build a standard `SPARQL` 1.1 query encoding the CP-theory and able to retrieve a ranked set of resources satisfying the corresponding preferential constraints. To our knowledge, this is the first attempt to encode the semantics of a CP-theory into a `SPARQL` query and, along with [39], the first approach that lets `SPARQL` to retrieve a ranked list of resources ordered according to a user's preferences.

We intend the proposed approach as a starting point for many future directions to reason with preferences in a pure Linked Data setting. More efficient encodings for the proposed queries could be investigated, able to mitigate some performance problems, related, for example, to the increase in the size of variables domains. Moreover, in its current version, the algorithm first computes the complete partial order of the outcomes, and then it matches them with items satisfying the users' hard requirements. As the computation of the partial order is computationally expensive, an improvement could surely be to compute and order only those outcomes matching the users' requirements, thus reducing the number of comparisons needed to return query results to the user. As a future direction of our research, we are also working on approaches proposed for automated CP-nets and CP-theories elicitation [11,26]. Another interesting topic for future research is to explore how our present work can be extended by integrating approaches to preference-based query answering over graph databases, such as the ones in [22,25,20], as well as how to deal with variables having continuous domains.

Acknowledgments

The authors wish to thank Francesco Paolo Albano for his unvaluable help in the implementation of the full framework and for his continuous support in its maintenance and further development.

This work was supported by The Alan Turing Institute under the UK EPSRC grant EP/N510129/1, and by the EPSRC grants EP/R013667/1, EP/L012138/1, and EP/M025268/1.

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Sci. Am.*, 284(5):34–43, 2001.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE 2001*, pages 421–430. IEEE Computer Society, 2001.

[3] C. Boutilier, R. I. Brafman, C. Domshlak, H. H. Hoos, and D. Poole. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.*, 21:135–191, 2004.

[4] R. I. Brafman and C. Domshlak. Introducing variable importance tradeoffs into CP-nets. In *Proc. of UAI 2002*, pages 69–76. Morgan Kaufmann, 2002.

[5] G. Brewka, S. Benferhat, and D. L. Berre. Qualitative choice logic. *Artif. Intell.*, 157(1/2):203–237, 2004.

[6] G. Brewka, J. P. Delgrande, J. Romero, and T. Schaub. asprin: Customizing answer set preferences without a headache. In *Proc. of AAAI 2015*, pages 1467–1474. AAAI Press, 2015.

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

[8] J. Chomicki. Logical foundations of preference queries. *IEEE Data Eng. Bull.*, 34(2):3–10, 2011.

[9] C. Cornelio, A. Loreggia, and V. A. Saraswat. Logical conditional preference theories. In *Proc. of MPREF 2015*, 2015.

[10] T. Di Noia, T. Lukasiewicz, M. V. Martínez, G. I. Simari, and O. Tifrea-Marciuska. Combining existential rules with the power of CP-theories. In *Proc. of IJCAI 2015*, pages 2918–2925. AAAI Press, 2015.

[11] Y. Dimopoulos, L. Michael, and F. Athienitou. Ceteris paribus preference elicitation with predictive guarantees. In *Proc. of IJCAI 2009*, pages 1890–1895, 2009.

[12] C. Domshlak, E. Hüllermeier, S. Kaci, and H. Prade. Preferences in AI: An overview. *Artif. Intell.*, 175:1037–1052, 2011.

[13] C. Domshlak, E. Hüllermeier, S. Kaci, H. Prade, F. Yaman, T. J. Walsh, M. L. Littman, and M. desJardins. Representing, processing, and learning preferences: Theoretical and practical challenges democratic approximation of lexicographic preference models. *Artif. Intell.*, 175(7):1290–1307, 2011.

[14] C. Domshlak and T. Joachims. Efficient and non-parametric reasoning over user preferences. *User Model. User-Adapt. Interact.*, 17(1/2):41–69, 2007.

[15] C. Domshlak and T. Joachims. Unstructuring user preferences: Efficient non-parametric utility revelation. *CoRR*, abs/1207.1390, 2012.

[16] C. Domshlak, S. Prestwich, F. Rossi, K. Venable, and T. Walsh. Hard and soft constraints for reasoning about qualitative conditional preferences. *J. Heuristics*, 12(4/5):263–285, 2006.

[17] C. Domshlak, F. Rossi, K. B. Venable, and T. Walsh. Reasoning about soft constraints and conditional preferences: Complexity results and approximation techniques. In *Proc. of IJCAI 2003*, pages 215–220. Morgan Kaufmann, 2003.

[18] J. Doyle. Prospects for preferences. *Comput. Intell.*, 20(2):111–136, 2004.

[19] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[20] V. Fionda and G. Pirrò. Querying graphs with preferences. In *Proc. of CIKM 2013*, pages 929–938. ACM Press, 2013.

[21] P. C. Fishburn. *Utility theory for decision making*. Publications in operations research. J. Wiley, 1970.

[22] S. Flesca and S. Greco. Partially ordered regular languages for graph queries. *J. Comput. Syst. Sci.*, 70(1):1–25, 2005.

[23] J. Fürnkranz and E. Hüllermeier. *Preference Learning*. Springer, 1st edition, 2010.

[24] J. Goldsmith, J. Lang, M. Truszczynski, and N. Wilson. The computational complexity of dominance and consistency in CP-nets. *J. Artif. Intell. Res.*, 33:403–432, 2008.

[25] G. Grahne, A. Thomo, and W. W. Wadge. Preferential regular path queries. *Fundam. Inform.*, 89(2-3):259–288, 2008.

[26] J. T. Guerin, T. E. Allen, and J. Goldsmith. Learning CP-net preferences online from user queries. In P. Perny, M. Pirlot, and A. Tsoukiàs, editors, *Algorithmic Decision Theory*, pages 208–220. Springer, 2013.

[27] M. Gueroussova, A. Polleres, and S. A. McIlraith. SPARQL with qualitative and quantitative preferences. In *Proc. of OrdRing 2013*, volume 1059 of *CEUR Workshop Proceedings*, pages 2–8. CEUR-WS.org, 2013.

[28] S. O. Hansson. What is ceteris paribus preference? *J. Philos. Logic*, 25(3):307–332, 1996.

[29] W. Kießling. Foundations of preferences in database systems. In *Proc. of VLDB 2002*, pages 311–322. VLDB Endowment, 2002.

[30] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL system—An overview. *IEEE Data Eng. Bull.*, 34(2):11–18, 2011.

[31] F. Koriche and B. Zanuttini. Learning conditional preference networks. *Artif. Intell.*, 174(11):685–703, 2010.

[32] A. Leonardo and F. Vasco. OWLPref: Uma representação declarativa de preferências para web semântica. Anais do XXVII Congresso da SBC, pages 1411–1420, 2007.

[33] C. Li, M. A. Soliman, K. C.-C. Chang, and I. F. Ilyas. RankSQL: Supporting ranking queries in relational database management systems. In *Proc. of VLDB 2005*, pages 1342–1345. VLDB Endowment, 2005.

[34] J. Liu, Y. Xiong, C. Wu, Z. Yao, and W. Liu. Learning conditional preference networks from inconsistent examples. *IEEE T. Knowl. Data En.*, 26(2):376–390, 2014.

[35] J. Liu, Z. Yao, Y. Xiong, W. Liu, and C. Wu. Learning conditional preference network from noisy samples using hypothesis testing. *Knowl.-Based Syst.*, 40:7–16, 2013.

[36] W. Liu, C. Wu, B. Feng, and J. Liu. Conditional preference in recommender systems. *Expert Syst. Appl.*, 42(2):774–788, 2015.

[37] I. Nunes, S. Miles, M. Luck, and C. J. P. de Lucena. An introduction to reasoning over qualitative multi-attribute preferences. *Knowl. Eng. Rev.*, 30:342–372, 5 2015.

[38] G. Pigozzi, A. Tsoukiàs, and P. Viappiani. Preferences in artificial intelligence. *Ann. Math. Artif. Intell.*, 77(3/4):361–401, 2016.

[39] J. Rosati, T. Di Noia, T. Lukasiewicz, R. De Leone, and A. Maurino. Preference queries with ceteris paribus semantics for Linked Data. In *OTM Conferences*, volume 9415 of *LNCS*, pages 423–442. Springer, 2015.

[40] W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In *Proc. of ISWC 2006*, pages 612–624. Springer, 2006.

[41] A. Troumpoukis, S. Konstantopoulos, and A. Charalambidis. An extension of SPARQL for expressing qualitative preferences. In *Proc. of ISWC 2017, Part I*, volume 10587 of *LNCS*, pages 711–727. Springer, 2017.

[42] G. H. Von Wright. *The Logic of Preference*. Edinburgh University Press, 1963.

[43] N. Wilson. Extending CP-nets with stronger conditional preference statements. In *Proc. of AAAI*, pages 735–741, 2004.

[44] N. Wilson. Computational techniques for a simple theory of conditional preferences. *Artif. Intell.*, 175:1053–1091, 2011.

Appendix

Appendix A Query formulation algorithm for CP-theories

Algorithm 1 has the user's preferences graph $\mathcal{G}_{user}$ as input, that is, the `RDF` version of the user's CP-theory Γ in terms of the *full* ontology and returns the `SPARQL` query able to order outcomes according to $\gg_\Gamma$, a strict partial order extending $>_\Gamma$ (see Theorem 2). Line 5 computes, for each outcome o, the values (of variables in V) that it is composed of, through the string $Outcome\_D\_values$ built with the for cycle of lines 2–4. The string $Outcome\_D\_values$ contains just the names of variables in V with a suffix $\_D$. Line 5 computes also the number of outcomes o' that o dominates according to $\gg_\Gamma$, referred to as `?counter`. The counting is made possible by the combination of the `COUNT` in line 5 and the `GROUP BY` in line 26. The `?counter` variable is then used by the `ORDER BY` in line 27 to rank the result set. It is worth to notice that line 5 asks for `URI(?outcome_D)` and not just for `?outcome_D`, since this entity will be em-

Algorithm 1

1: **procedure** GENERATEORDERINGQUERYFORCP-THEORIES($\mathcal{G}_{user}$)
2: **for all** $X_i \in V$ **do**
3: $Outcome\_D\_values$+=?X$_i$_D
4: **end for**
5: OrderingQuery = SELECT (URI(?outcome_D) AS ?URIOutcome) $Outcome\_D\_values$ (COUNT(DISTINCT ?outcome_d) AS ?counter) WHERE{ ;
6: OrderingQuery += {SELECT DISTINCT ?outcome_D ?outcome_d $Outcome\_D\_values$ (count(DISTINCT ?V) AS ?counterVundominated) (sum((?counterBind)) AS ?counterV) WHERE{ ;
7: OrderingQuery += {SELECT DISTINCT ?outcome_D ?outcome_d ?V $Outcome\_D\_values$ WHERE{ ;
8: **for** $y \in$ {D, d} **do**
9: **for all** $X_i \in V$ **do**
10: OrderingQuery += VALUES(?X$_i$_y) { ($value(x_{i1})$) ($value(x_{i2})$) } ;
11: **end for**
12: OrderingQuery += BIND(CONCAT(STR(;
13: **for** $i = 1, \ldots, |W| - 1$ **do**
14: OrderingQuery += $attribute(X_i)$), STR(?X$_i$_y), ;
15: **end for**
16: OrderingQuery += STR($attribute(X_{|W|})$), STR(?X$_{|W|}$_y)) AS ?$outcome\_y$). ;
17: **end for**
18: OrderingQuery += FILTER(?outcome_D!=?outcome_d). ;
19: OrderingQuery += ?V a cpt:Variable. ?V cpt:variableDomain ?variable1. ?variable1 cpt:value ?value1. ?V cpt:variableDomain ?variable2. ?variable2 cpt:value ?value2. FILTER (!(?value1=?value2)&& contains(?outcome_D,str(?value1))&& contains(?outcome_d,str(?value2))). ;
20: OrderingQuery += FILTER NOT EXISTS{ ?V2 cpt:moreImportantThan ?V. ?V2 cpt:variableDomain ?vd1. ?vd1 cpt:value ?v1. ?V2 cpt:variableDomain ?vd2. ?vd2 cpt:value ?v2. FILTER((!(?v1=?v2))&& contains(?outcome_D,str(?v1)) && contains(?outcome_d,str(?v2))). } ;
21: OrderingQuery += FILTER NOT EXISTS{ ?V3 cpt:conditionallyMoreImportantThan ?instanceOfRelativeImportance. ?instanceOfRelativeImportance cpt:hasCondition ?C. ?instanceOfRelativeImportance cpt:hasLessImportantVariable ?V. ?V3 cpt:variableDomain ?vd13. ?vd13 cpt:value ?v13. ?V3 cpt:variableDomain ?vd23. ?vd23 cpt:value ?v23. {Select distinct ?C (GROUP_CONCAT(CONCAT(str(?attr),str(?value)) ; separator ="") as ?Concatenated) where{ ?C cpt:contains ?c. ?c cpt:attribute ?attr; cpt:value ?value. } GROUP BY ?C } FILTER(CONTAINS(?outcome_D,?Concatenated)). FILTER(!(?v13=?v23)&& contains(?outcome_D,str(?v13)) && contains(?outcome_d,str(?v23))). } }} ;
22: OrderingQuery += {SELECT DISTINCT ?V ?ConcatenatedParent ?Prefer ?Over WHERE{ {Query 1} UNION {Query 2} }} ;
23: OrderingQuery += BIND(IF(((!BOUND(?ConcatenatedParent) && contains(?outcome_D,?Prefer)&& contains(?outcome_d,?Over)) || (BOUND(?ConcatenatedParent) && ?ConcatenatedParent!="" && contains(?outcome_D,?ConcatenatedParent) && contains(?outcome_D,?Prefer) && contains(?outcome_d,?Over))) ,1,0) as ?counterBind) ;
24: OrderingQuery += } GROUP BY ?outcome_D $Outcome\_D\_values$?outcome_d } ;
25: OrderingQuery += FILTER(?counterV=?counterVundominated)} ;
26: OrderingQuery += GROUP BY ?outcome_D $Outcome\_D\_values$;
27: OrderingQuery += ORDER BY DESC (?counter) ;
28: **return** OrderingQuery
29: **end procedure**

ployed as the subject of triples at the beginning of **Step 2** described in Section 4.2. Given a pair of outcomes (o, o'), lines 6 to 24 are used to identify the set of variables $\Theta^{'}(o, o')$ and among them the set of vari-

ables $\{X \in \Theta'(o, o')$: $o(X) \succ_o^X o'(X)\}$. The first nested subquery (lines 7 to 21) considers one pair of outcomes at a time, `?outcome_D` and `?outcome_d`, where `D` and `d` stand respectively for ***D**ominating* and ***d**ominated*. The for loop of lines 8–17 allows to consider first `?outcome_D` and then `?outcome_d`. For each of them, the nested loop of lines 9–11 introduces the values corresponding to the variables in V. For each variable $X_i \in V$, Algorithm 1 looks for values `?X`$_i$`_`y filtering only elements of the set $\{$*value*(x_{i1}), *value*$(x_{i2})\}$ in the binary case, or $\{$*value*$(x_{i1}), \ldots,$ *value*$(x_{in})\}$ elsewhere, through the `VALUES` assignment at line 10. The algorithm requires that a list W of variables is defined from the set of variables V of Γ: the variables that refer to each instance of `Condition` must appear in the same order in W, optionally allowing some recurrence. At lines 12-16, the outcome is explicitly built by concatenating, according to the order imposed by W, the values extracted for various `?X`$_i$`_`y together with *attribute*(X_i), for all members of W. Line 18 is added to verify that the pair of outcomes to compare is made of distinct elements.

At line 19, the patterns and the `FILTER` are used to identify the variables `?V` with different values `?value1` and `?value2` in the outcomes `?outcome_D` and `?outcome_d`, namely the variables in the set Δ(`?outcome_D`, `?outcome_d`). As imposed by the couple of `FILTER NOT EXISTS` of lines 20 and 21, these variables `?V` are such that:

- there does not exist any variable `?V2` in the set Δ(`?outcome_D`, `?outcome_d`) that is `moreImportantThan ?V`;
- there does not exist any variable `?V3` in the set Δ(`?outcome_D`, `?outcome_d`) that is `conditionallyMoreImportantThan ?V` under a `Condition` extended by `?outcome_D`.

In particular, the nested subquery appearing within the `FILTER NOT EXISTS` of line 21 extracts only the instances of `Condition` extended by `?outcome_D`, building the representative strings of the conditional values, concatenating them in `?Concatenated` and checking the inclusion of the string `?Concatenated` in `?outcome_D` through `FILTER` and `CONTAINS`. The pair of `FILTER NOT EXISTS` of lines 20 and 21 allows therefore to identify the set of variables Θ'(`?outcome_D`, `?outcome_d`).

The `UNION` of Query 1 and Query 2 is added at line 22. It returns a set of quadruples of the general form ⟨`?V`, `?ConcatenatedParent`, `?Prefer`, `?Over`⟩, able to order an outcome over another one, locally with respect to `?V`. For each variable `?V` within the set Θ'(`?outcome_D`, `?outcome_d`), the `IF` of line 23 verifies if one of the quadruples on variable `?V` can be used to order `?outcome_D` over `?outcome_d` locally with respect to `?V`. In particular, for quadruples with a missing value for `?ConcatenatedParent`, it is sufficient to verify if `?outcome_D` contains the better value of a preference, i.e., `?Prefer`, and `?outcome_d` contains the relative worse value `?Over`. Instead, for quadruples with a bound value for `?ConcatenatedParent`, it must happens that `?outcome_D` contains the value of `?ConcatenatedParent`, as well as `?Prefer` and `?outcome_d` contain the value of `?Over`. If one of the `||` (or) conditions happens, the `BIND` instantiate the value of `?counterBind` to 1, otherwise to 0. The `?counterBind` value is summed up across all instantiation of `?V` in Θ'(`?outcome_D`, `?outcome_d`), and it resolves into `?counterV` at line 6. The same line computes also the cardinality of Θ'(`?outcome_D`, `?outcome_d`), namely, the value `?counterVundominated`. The `FILTER` at line 25 verifies if the pair of values `?counterVundominated` and `?counterV` coincides, that is, if `?outcome_D`$(X) \succ^X_{\texttt{?outcome\_D}}$ `?outcome_d`(X) for all variables X in Θ'(`?outcome_D`, `?outcome_d`). In conclusion, if the `FILTER` of line 25 returns `true` then `?outcome_D` dominates `?outcome_d` with respect to $\gg_\Gamma$, and its `?counter` value is incremented of a unit. Only distinct dominated outcomes are counted, through the solution modifier `DISTINCT` at line 5.

If we consider the query in Appendix B resulting from the running example on Giorgio's preferences, we see that the variables involved in CP-statements as well as the corresponding values are encoded in the initial part of the query (line 9-50) and in the `GROUP BY` statement (line 130). The remaining of the query is quite standard and does not depend on the underlying CP-theory Γ. As for the initial part of the query, it contains a number of $2 \cdot |V|$ `VALUES` statements, where we assign all the allowed values to variables in V and a `BIND` statement used to compose the strings representing all possible outcomes. We emphasize that the whole query is automatically generated by Algorithm 1, starting from the *full* version of Γ, and then the process is completely transparent to the user.

Appendix B Ordering Query for the Book Example

```
prefix cpt:<http://sisinflab.poliba.it/semanticweb/lod/ontologies/cpt_full.owl#>
prefix dbpedia-owl:<http://dbpedia.org/ontology/>
prefix dbpedia:<http://dbpedia.org/resource/>
prefix g:<http://sisinflab.poliba.it/semanticweb/graphs/>
SELECT  (URI(?outcome_D) AS ?URIOutcome) ?genre_D ?country_D ?subwork_D
        ?filmVersion_D (COUNT(DISTINCT ?outcome_d) AS ?counter)
WHERE
{
   { SELECT DISTINCT ?outcome_D ?outcome_d ?genre_D ?country_D ?subwork_D
      ?filmVersion_D (COUNT(DISTINCT ?V) AS ?counterVundominated)
      (SUM((?counterBind))AS ?counterV)
     WHERE
     {
        {SELECT DISTINCT ?outcome_D ?outcome_d ?V ?genre_D ?country_D ?subwork_D
                         ?filmVersion_D
         WHERE
         {
           VALUES (?genre_D) {
             (dbpedia:Crime_fiction)   (dbpedia:Autobiographical_novel)
           }
           VALUES (?country_D) {
            (dbpedia:France)  (dbpedia:United_Kingdom)
           }
           VALUES (?subwork_D) {
             (cpt:subsequentWorkYes)   (cpt:subsequentWorkNo)
           }
           VALUES (?filmVersion_D) {
             (cpt:filmVersionYes) (cpt:filmVersionNo)
           }
           BIND (CONCAT(STR(dbpedia-owl:country),STR(?country_D),
                  STR(dbpedia-owl:literaryGenre),STR(?genre_D),
                  STR(dbpedia-owl:subsequentWork),STR(?subwork_D),
                  STR(dbpedia-owl:filmVersion),STR(?filmVersion_D)) AS ?outcome_D).
           VALUES (?genre_d) {
              (dbpedia:Crime_fiction)  (dbpedia:Autobiographical_novel)
           }
           VALUES (?country_d) {
              (dbpedia:France)   (dbpedia:United_Kingdom)
           }
           VALUES (?subwork_d) {
              (cpt:subsequentWorkYes)   (cpt:subsequentWorkNo)
           }
           VALUES (?filmVersion_d) {
               (cpt:filmVersionYes)  (cpt:filmVersionNo)
           }
           BIND (CONCAT(STR(dbpedia-owl:country),STR(?country_d),
                  STR(dbpedia-owl::literaryGenre),STR(?genre_d),
                  STR(dbpedia-owl:subsequentWork),STR(?subwork_d),
                  STR(dbpedia-owl:filmVersion),STR(?filmVersion_d)) AS ?outcome_d).
           FILTER(?outcome_D!=?outcome_d).

           ?V a cpt:Variable.
           ?V cpt:variableDomain ?variable1. ?variable1 cpt:value ?value1.
           ?V cpt:variableDomain ?variable2. ?variable2 cpt:value ?value2.
           FILTER (!(?value1=?value2)&& CONTAINS(?outcome_D,STR(?value1))
                    && CONTAINS(?outcome_d,STR(?value2))).
           FILTER NOT EXISTS{
             ?V2 cpt:moreImportantThan ?V.
             ?V2 cpt:variableDomain ?vd1. ?vd1 cpt:value ?v1.
             ?V2 cpt:variableDomain ?vd2. ?vd2 cpt:value ?v2.
             FILTER((!(?v1=?v2))&& CONTAINS(?outcome_D,STR(?v1)) &&
                    CONTAINS(?outcome_d,STR(?v2))).
           }
           FILTER NOT EXISTS{
            ?V3 cpt:conditionallyMoreImportantThan ?instanceOfRelativeImportance.
            ?instanceOfRelativeImportance cpt:hasCondition ?C.
            ?instanceOfRelativeImportance cpt:hasLessImportantVariable ?V.
            ?V3 cpt:variableDomain ?vd13. ?vd13 cpt:value ?v13.
            ?V3 cpt:variableDomain ?vd23. ?vd23 cpt:value ?v23.
            {  SELECT DISTINCT ?C
                 (GROUP_CONCAT(CONCAT(STR(?attr),STR(?value));separator ="")
                   AS ?Concatenated) WHERE{
               ?C cpt:contains ?c.
```

```
                ?c cpt:attribute ?attr;  cpt:value ?value.
                }
                GROUP BY ?C
             }
             FILTER(CONTAINS(?outcome_D,?Concatenated)).
             FILTER(!(?v13=?v23)&& CONTAINS(?outcome_D,STR(?v13)) &&
                    CONTAINS(?outcome_d,STR(?v23))).
           }
         }
        }
        { SELECT DISTINCT ?V  ?ConcatenatedParent ?Prefer ?Over {
           {  SELECT ?V
                (CONCAT(STR(?attrPrefer),STR(?valuePrefer)) AS ?Prefer)
                (CONCAT(STR(?attrPrefer),STR(?valueOver)) AS ?Over)  WHERE
              {?preference cpt:prefer ?p;
                           cpt:over ?o.
               FILTER NOT EXISTS{?preference cpt:given ?condition.}
               ?V cpt:variableDomain ?p.
               ?p cpt:attribute ?attrPrefer;
                  cpt:value ?valuePrefer.
               ?o cpt:value ?valueOver.
              }
           }
           UNION
           {  SELECT DISTINCT ?V  ?ConcatenatedParent ?Prefer ?Over  WHERE
              {
                  SELECT DISTINCT  ?condition ?V
                    (CONCAT(STR(?attrPrefer),STR(?valuePrefer)) AS ?Prefer)
                    (CONCAT(STR(?attrPrefer),STR(?valueOver)) AS ?Over)
                    (GROUP_CONCAT(CONCAT(STR(?attr),STR(?value));separator ="")
                      AS ?ConcatenatedParent) WHERE
                  { ?preference cpt:given ?condition.
                    ?preference cpt:prefer ?p;
                                cpt:over ?o.
                    ?V cpt:variableDomain ?p.
                    ?p cpt:attribute ?attrPrefer;
                       cpt:value ?valuePrefer.
                    ?o cpt:value ?valueOver.
                    ?condition cpt:contains ?c.
                    ?c cpt:attribute ?attr;
                       cpt:value ?value.
                  }
                  GROUP BY ?condition ?V ?attrPrefer ?valuePrefer ?valueOver
              }
           }
          }
        }
        BIND(IF(  ( (!BOUND(?ConcatenatedParent) &&
             CONTAINS(?outcome_D,?Prefer)&&
             CONTAINS(?outcome_d,?Over))
            ||
            (BOUND(?ConcatenatedParent) && ?ConcatenatedParent!="" &&
             CONTAINS(?outcome_D,?ConcatenatedParent) &&
             CONTAINS(?outcome_D,?Prefer) &&
             CONTAINS(?outcome_d,?Over)) )  ,1,0) AS ?counterBind )
     }
     GROUP BY  ?outcome_D ?genre_D ?country_D ?subwork_D ?filmVersion_D ?outcome_d
   }
 FILTER(?counterV=?counterVundominated)
}
GROUP BY  ?outcome_D ?genre_D ?country_D ?subwork_D ?filmVersion_D
ORDER BY DESC (?counter)
```