A New Matchmaking Approach Based on Abductive Conjunctive Query Answering

Jianfeng Du[1,2], Shuai Wang[1], Guilin Qi[3], Jeff Z. Pan[4], and Yong Hu[1]

[1] Guangdong University of Foreign Studies, Guangzhou 510006, China
jfdu@mail.gdufs.edu.cn
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[3] Southeast University, Nanjing 211189, China
[4] The University of Aberdeen, Aberdeen AB243UE, UK

Abstract. To perform matchmaking in Web-based scenarios where data are often incomplete, we propose an extended conjunctive query answering (CQA) problem, called *abductive CQA problem*, in Description Logic ontologies. Given a consistent ontology and a conjunctive query, the abductive CQA problem computes all *abductive answers* to the query in the ontology. An abductive answer is an answer to the query in some consistent ontology enlarged from the given one by adding a bounded number of individual assertions, where the individual assertions that can be added are confined by user-specified concept or role names. We also propose a new approach to matchmaking based on the abductive CQA semantics, in which offer information is expressed as individual assertions, request information is expressed as conjunctive queries, and matches for a request are defined as abductive answers to a conjunctive query that expresses the request. We propose a sound and complete method for computing all abductive answers to a conjunctive query in an ontology expressed in the Description Logic Program fragment of OWL 2 DL with the Unique Name Assumption. The feasibility of this method is demonstrated by a real-life application, rental matchmaking, which handles requests for renting houses.

1 Introduction

Matchmaking is a useful facility for comparing offers with requests. It determines whether an offer matches a request or whether a request matches an offer, and has been widely used in Web service discovery [18, 3, 10], skill matching [6], marriage matching [2] and product matching in e-marketplaces [14, 15, 5, 16].

Existing approaches to matchmaking can be divided into two categories, namely syntactic ones and semantic ones. Syntactic approaches usually exploit keyword-based search methods to compare offers with requests. These approaches make little use of the background knowledge and the semantics about offers or requests, and will easily miss right matches or yield wrong matches. Semantic approaches usually use ontologies to formalize offers and requests. Since an ontology provides the background knowledge and formalizes offers and requests

with certain semantics, semantic approaches can make the matchmaking results more sound and complete than syntactic approaches.

The World Wide Web Consortium (W3C) has proposed the Web Ontology Language (OWL), for which the newest version is OWL 2 [11], to model ontologies. OWL is based on a family of formal languages, called *Description Logics* (DLs) [1]. In particular, the most expressive and decidable species of OWL 2, OWL 2 DL, corresponds to the DL $\mathcal{SROIQ}$ [13]. The proposal of OWL has motivated the industry to upgrade many applications to DL-based ones. Recent semantic approaches to matchmaking are also based on DLs. These approaches can be roughly divided into two sorts, described below.

The first sort approaches, such as [3, 10, 18], exploit a DL ontology to compute semantic distances between offers and requests, where offers and requests are expressed as DL concept descriptions (simply DL concepts). These approaches mainly focus on defining a reasonable distance function between two DL concepts. The primary drawback is that they cannot guarantee that the computed distances adhere to the DL semantics. For example, when an offer matches a request, i.e., the offer is subsumed by the request under the DL semantics, the computed distance may not be zero.

The second sort approaches, such as [14, 15, 5], also express offers and requests as DL concepts, but they exploit DL inference methods to compute different matches. In the approaches proposed in [14, 15], a popular DL inference method, namely concept subsumption checking, is used to compute several kinds of matches. Two kinds that are most related to this work are respectively the *potential match* proposed in [15] and the *intersection match* proposed in [14], where a potential match for a request is an offer matching a portion of the request, while an intersection match for a request is an offer consistent with the request. In the approach proposed in [5], two non-standard DL inference methods, namely concept abduction and concept contraction, are used to compute *possible matches*. A possible match for a request is an offer that gets subsumed by the request after adding some information to the offer (i.e. abduction) and removing some information from the request (i.e. contraction).

All the aforementioned semantic approaches are not easy to scale to real-life applications that involve a large number of offers and requests, because composing the DL concepts for offers and requests is time consuming and laborious. A more practical approach should alleviate human efforts to formalize offers or requests. Hence, we use for reference another approach which is based on conjunctive query answering (CQA). In this approach, offer information is expressed as individual assertions in the back-end ontology, request information is expressed as conjunctive queries posed upon the back-end ontology, and matches for a request are defined as answers to a conjunctive query that expresses the request. This approach enables an efficient way to construct a matchmaking system. That is, a large portion of the back-end ontology, which stores data about offers (i.e. offer information), can be automatically extracted from Web sources using ontology population techniques [4, 8], while a very small portion of the back-end ontology, which stores background knowledge, can be manually built

using ontology editors. This approach has been used to solve the fuzzy matchmarking problem [16], where the back-end ontology is expressed by a Datalog program with *scoring atoms* that calculate the match degrees. It has also been used to support Semantic Web search [9] for DL back-end ontologies.

However, the CQA based approach is unsuitable in Web-based scenarios where data of the back-end ontology come from the World Wide Web and are often incomplete. Since an offer that is not originally an answer to a conjunctive query can be turned into an answer to after missing data are added, the offer can also be considered as a match. To capture this idea, we propose an extended CQA problem, called *abductive CQA problem*. Given a consistent ontology and a conjunctive query, the abductive CQA problem computes all *abductive answers* to the query in the ontology. An abductive answer is an answer to the query in a certain consistent ontology enlarged from the given one by adding a bounded number of individual assertions, where the individual assertions that can be added are confined by two disjoint sets of concept or role names. The names in the first set are called *abducible predicates*. The possibly added individual assertions can be on abducible predicates only. The names in the second set are called *closed predicates*. The possibly added individual assertions cannot make the enlarged ontology entail any individual assertion that is on closed predicates and is not entailed by the given ontology.

Based on the abducible CQA semantics, we propose a new semantic approach to computing all matches for a given request, where these matches are abductive answers to a conjunctive query that expresses the request. This notion of match is similar to the notion of potential match [15] and the notion of intersection match [14] — all of them regard matches for a request as offers satisfying a certain portion of the request. We also propose a method for computing all abductive answers to a conjunctive query. The method encodes the abductive CQA problem into a Prolog program and solves it with Prolog engines. To ensure that the method is sound and complete, we assume that the given ontology is expressed in the Description Logic Program (DLP) [12] fragment of OWL 2 DL and adopts the Unique Name Assumption [1]. The DLP fragment of OWL 2 DL underpins the OWL 2 RL profile of OWL 2 [11] and is often used in applications that require efficient reasoning facilities. The Unique Name Assumption, which explicitly declares that any two different individual names correspond to different elements in the interpretation domain, is often used with DLs.

We conducted experiments in a real-life application, rental matchmaking, which handles requests for renting houses in an ontology with more than one million individual assertions. We carefully designed ten benchmark queries for this application. Experimental results show that the proposed method is rather efficient in computing abductive answers to the benchmark queries.

The remainder of the paper is organized as follows. After providing preliminaries in the next section, in Sect. 3 we give more details on the abductive CQA problem. Then in Sect. 4, we describe the proposed method for computing all abductive answers to a conjunctive query. Before making a conclusion, we present our experimental evaluation in Sect. 5.

2 Preliminaries

2.1 OWL 2 and DLP

We assume that the reader is familiar with OWL 2 [11] and briefly introduce the most expressive and decidable species of OWL 2, OWL 2 DL, which corresponds to the DL $\mathcal{SROIQ}$ [13] with datatypes. An OWL 2 DL ontology consists of an RBox, a TBox and an ABox. The RBox consists of a finite set of role inclusion axioms and role assertions. The TBox consists of a finite set of concept inclusion axioms. The ABox consists of a finite set of individual assertions. OWL 2 DL (i.e. $\mathcal{SROIQ}$) is a syntactic variant of a fragment of First-order Logic, and the semantics of a $\mathcal{SROIQ}$ ontology $\mathcal{O}$ can be defined by translating to a formula $\pi(\mathcal{O})$ of First-order Logic with equality, where π is defined in Table 1. We use the traditional rule form to represent $\pi(\mathcal{O})$. For example, consider the ontology (called the *rental ontology*) used in our experiments, the following two axioms in the RBox of the rental ontology: $\mathsf{Tra}(\mathsf{isA})$ and $\mathsf{hasFacility} \circ \mathsf{isA} \sqsubseteq \mathsf{hasFacility}$, and the following two axioms in the TBox: $\mathsf{House} \sqsubseteq \mathsf{Building}$ and $\mathsf{Building} \sqcap \mathsf{TrafficLine} \sqsubseteq \bot$, can be translated to the following First-order rules.

$R_1 = \forall x, y, z : \mathsf{isA}(x, y) \wedge \mathsf{isA}(y, z) \rightarrow \mathsf{isA}(x, z)$
$R_2 = \forall x, y, z : \mathsf{hasFacility}(x, y) \wedge \mathsf{isA}(y, z) \rightarrow \mathsf{hasFacility}(x, z)$
$R_3 = \forall x : \mathsf{House}(x) \rightarrow \mathsf{Building}(x)$
$R_4 = \forall x : \mathsf{House}(x) \wedge \mathsf{TrafficLine}(x) \rightarrow \bot$

Rule R_1 tells that if x is more specific than y and y is more specific than z, then x is more specific than z. Rule R_2 tells that if x has a facility y and y is more specific than z, then x also has a facility z. Rule R_3 tells that if x is a renting house, then it is also a building. Rule R_4 tells if x is a building, then it cannot be a traffic line, where the symbol $\bot$ in rule consequences denotes a contradiction.

A *model* of an OWL 2 DL ontology $\mathcal{O}$ is a model of $\pi(\mathcal{O})$ under the traditional First-order semantics. $\mathcal{O}$ is said to be *consistent* if it admits at least one model. An individual assertion α is said to be *entailed* by $\mathcal{O}$, denoted by $\mathcal{O} \models \alpha$, if α is satisfied by all models of $\mathcal{O}$. The *Unique Name Assumption* [1] in $\mathcal{O}$ is an assumption that $\mathcal{O} \models a \not\approx b$ for any two different individual names a and b occurring in $\mathcal{O}$.

The DLP [12] fragment of OWL 2 DL is the intersection of OWL 2 DL and Horn Logic, another fragment of First-order Logic in which all rules have no existential quantifiers or disjunctions in the consequence part. We simply call an ontology *DLP ontology* if it is expressed in the DLP fragment of OWL 2 DL and adopts the Unique Name Assumption.

2.2 Conjunctive Query Answering

A *conjunctive query* is an expression of the form $\exists \overrightarrow{y} : \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$, where $\overrightarrow{x}$ is a vector of *distinguished variables*, $\overrightarrow{y}$ a vector of *non-distinguished variables* and $\overrightarrow{c}$ a vector of individuals or constants. $\mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ denotes a conjunction of *atoms* of the form $A(v)$ or $r(v_1, v_2)$, where A is an *atomic concept* (i.e. a concept

Table 1. The semantics of $\mathcal{SROIQ}$ by mapping to First-order Logic

Translating $\mathcal{SROIQ}$ concepts to First-order Logic	
$\pi_x(\top) = \top$	$\pi_x(\bot) = \bot$
$\pi_x(A) = A(x)$	$\pi_x(\neg C) = \neg\pi_x(C)$
$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$	$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$
$\pi_x(\exists r.C) = \exists y : \mathsf{ar}(r, x, y) \wedge \pi_y(C)$	$\pi_x(\forall r.C) = \forall y : \mathsf{ar}(r, x, y) \rightarrow \pi_y(C)$
$\pi_x(\exists r.\mathsf{Self}) = \mathsf{ar}(r, x, x)$	$\pi_x(\{a\}) = x \approx a$
$\pi_x(\geq_n r.C) = \exists y_1, ..., y_n : \bigwedge_{i=1}^{n}(\mathsf{ar}(r, x, y_i) \wedge \pi_{y_i}(C)) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \not\approx y_j$	
$\pi_x(\leq_n r.C) = \forall y_1, ..., y_{n+1} : \bigwedge_{i=1}^{n+1}(\mathsf{ar}(r, x, y_i) \wedge \pi_{y_i}(C)) \rightarrow \bigvee_{1 \leq i < j \leq n+1} y_i \approx y_j$	
Translating axioms to First-order Logic	
RBox: $\pi(r_1 \circ \ldots \circ r_n \sqsubseteq s) = \forall x_1, ..., x_{n+1} : \bigwedge_{i=1}^{n} \mathsf{ar}(r_i, x_i, x_{i+1}) \rightarrow \mathsf{ar}(s, x_1, x_{n+1})$	
$\pi(\mathsf{Tra}(r)) = \forall x, y, z : \mathsf{ar}(r, x, y) \wedge \mathsf{ar}(r, y, z) \rightarrow \mathsf{ar}(r, x, z)$	
$\pi(\mathsf{Sym}(r)) = \forall x, y : \mathsf{ar}(r, x, y) \rightarrow \mathsf{ar}(r, y, x)$	$\pi(\mathsf{Ref}(r)) = \forall x : \mathsf{ar}(r, x, x)$
$\pi(\mathsf{Dis}(r, s)) = \forall x, y : \neg\mathsf{ar}(r, x, y) \vee \neg\mathsf{ar}(s, x, y)$	$\pi(\mathsf{Irr}(r)) = \forall x : \neg\mathsf{ar}(r, x, x)$
TBox: $\pi(C \sqsubseteq D) = \forall x : \pi_x(C) \rightarrow \pi_x(D)$	
ABox: $\pi(C(a)) = \pi_x(C)[x \mapsto a]$	$\pi(r(a, b)) = \mathsf{ar}(r, a, b)$
$\pi(\neg r(a, b)) = \neg\mathsf{ar}(r, a, b)$	$\pi(a \not\approx b) = a \not\approx b$
Translating $\mathcal{SROIQ}$ ontologies to First-order Logic	
$\pi(\mathcal{O}) = \bigwedge_{\alpha \in \mathcal{O}} \pi(\alpha)$	

Note: A denotes a concept name; $\mathsf{ar}(r, x, y)$ denotes $s(y, x)$ if r is an inverse role s^-, or denotes $r(x, y)$ otherwise.

name), r is an *atomic role* (i.e. a role name) or a built-in predicate, and v, v_1 and v_2 are variables in $\overrightarrow{x}$ and $\overrightarrow{y}$, or individuals or constants in $\overrightarrow{c}$. A *Boolean conjunctive query* is a conjunctive query without distinguished variables.

Given an OWL 2 DL ontology $\mathcal{O}$ and a Boolean conjunctive query $Q = \exists \overrightarrow{y} : \mathsf{conj}(\overrightarrow{y}, \overrightarrow{c})$, a model $\mathcal{I}$ of $\mathcal{O}$ is said to *satisfy* Q if there exists a tuple of (possibly anonymous) individuals or constants whose substitution for the variables in $\overrightarrow{y}$ makes every atom in $\mathsf{conj}(\overrightarrow{y}, \overrightarrow{c})$ satisfied by $\mathcal{I}$. Q is said to be *entailed* by $\mathcal{O}$, denoted by $\mathcal{O} \models Q$, if every model of $\mathcal{O}$ satisfies Q. A tuple $\overrightarrow{t}$ of individuals is called an *answer* to a conjunctive query $Q(\overrightarrow{x}) = \exists \overrightarrow{y} : \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ in $\mathcal{O}$ if $\mathcal{O} \models Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$, where $Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$ denotes a Boolean conjunctive query obtained from $Q(\overrightarrow{x})$ by replacing every variable in $\overrightarrow{x}$ with its corresponding individual in $\overrightarrow{t}$. The *conjunctive query answering* (*CQA*) *problem* is to compute all answers to a conjunctive query in an ontology.

3 The Abductive CQA Problem

By expressing offer information as individual assertions and request information as conjunctive queries, the matchmaking problem can be treated as the CQA problem. For example, when we want to find all renting houses whose price is up to 4000 yuan per month, we can pose the following conjunctive query upon the rental ontology: $\exists y : \mathsf{House}(x) \wedge \mathsf{rent}(x, y) \wedge y \leq 4000$. Then the answers to this query correspond to renting houses to be found. However, the answers to a

conjunctive query may not provide all choices to a requester. For example, when the price of a renting house is missing, possibly due to incomplete extraction from Web sources, this renting house will not be an answer of the aforementioned query, though its rental price is 3000 yuan per month in reality. Hence those answers in a certain enlarged ontology may also correspond to offers that match the request in reality. This enlarged ontology can be seen as the result of adding missing data about offers to the original ontology. Since offer information is expressed as individual assertions, the missing data should be restricted to individual assertions. Moreover, the number of added assertions should be bounded by some constant that reflects the incompleteness of the original ontology, while the added assertions should be confined by certain concept or role names according to the actual situation. Hence, we introduce an extended CQA problem, called *abductive CQA problem*, defined below.

Definition 1 (Abductive CQA). Given a consistent ontology $\mathcal{O}$, a conjunctive query Q, a non-negative integer k, two disjoint sets of concept or role names S_A and S_C, where the concept or role names in S_A (resp. S_C) are called *abducible predicates* (resp. *closed predicates*), a tuple $\overrightarrow{t}$ of individuals is called an *abductive answer* to Q in $\mathcal{O}$ w.r.t. k, S_A and S_C, if there exists a set $\mathcal{A}$ of individual assertions on the predicates in S_A such that $|\mathcal{A}| \leq k$, $\mathcal{O} \cup \mathcal{A} \models Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$, $\mathcal{O} \cup \mathcal{A}$ is consistent, and $\mathcal{O} \models \alpha$ for all individual assertions α on the predicates in S_C such that $\mathcal{O} \cup \mathcal{A} \models \alpha$, where $|S|$ denotes the cardinality of a set S, and $\mathcal{A}$ is said to be *attached with* $\overrightarrow{t}$. The *abductive CQA problem* is to compute all abductive answers to Q in $\mathcal{O}$ w.r.t. k, S_A and S_C.

In the above definition, $\mathcal{A}$ can be seen as the missing data about a certain offer. It consists of at most k individual assertions and should be a set such that the union of it and the given ontology $\mathcal{O}$ is consistent. Here, we call $\mathcal{A}$ a *candidate complement set* if $|\mathcal{A}| \leq k$ and $\mathcal{O} \cup \mathcal{A}$ is consistent. The set of abducible predicates, S_A, restricts all concepts or roles appearing in a candidate complement set to concept or role names in S_A. The set of closed predicates, S_C, further restricts that the union of a candidate complement set and the given ontology does not entail any individual assertion which is on a closed predicate but is not entailed by the given ontology. In general, a concept or role name can be set as an abducible predicate if some of its instances are possibly missing; it can be set as a closed predicate if all individual assertions on it are ensured to be entailed by the given ontology. However, it is not necessary that a concept or role name is set to be either abducible or closed.

The abductive CQA problem is similar to the ABox abduction problem proposed in [7], which computes all minimal sets $\mathcal{A}$ of individual assertions on a set S of predicates such that $\mathcal{O} \cup \mathcal{A}$ is consistent, $\mathcal{A} \not\models G$ and $\mathcal{O} \cup \mathcal{A} \models G$, for a consistent ontology $\mathcal{O}$ and a set G of individual assertions. Compared to the ABox abduction problem, the abductive CQA problem also restricts $\mathcal{A}$ to a set that consists of individual assertions on certain predicates and does not introduce inconsistency. But it computes abductive answers to a conjunctive query by considering all possible $\mathcal{A}$ instead of computing certain $\mathcal{A}$. Moreover, it introduces the use of the bounded number k and a set of closed predicates. The

bounded number k is used to control the extensiveness of abducible answers, while the usefulness of closed predicates is described below.

One use of the closed predicates is to simulate the use of disjoint concept or role axioms, which declare that two concepts or two roles are disjoint. For example, suppose $\mathsf{City}(a)$ is entailed by $\mathcal{O}$ and $\mathsf{House}(a)$ is not. When $\mathcal{O}$ does not contain the axiom declaring that House and City are disjoint, $\mathsf{House}(a)$ can possibly be entailed by $\mathcal{O} \cup \mathcal{A}$ for some candidate complement set $\mathcal{A}$. But we actually do not expect $\mathsf{House}(a)$ to be entailed by $\mathcal{O} \cup \mathcal{A}$ as $\mathsf{House}(a)$ does not hold in reality. Hence we can define House as a closed predicate, so that all candidate complement sets $\mathcal{A}$ making $\mathcal{O} \cup \mathcal{A}$ entail $\mathsf{House}(a)$ are not considered in computing abductive answers. Since disjoint concept or role axioms can easily be neglected when manually constructing or maintaining ontologies, the closed predicates are often needed to substitute the use of these axioms. Another use of the closed predicates is to enable some optimizations in computing abductive answers; this will be shown in the next section.

It should be mentioned that closed predicates are not predicates in an NBox (Negation-as-failure Box) [17]. Defining concept or role names in the NBox of an ontology impacts the semantics of the ontology. For example, defining House as a concept name in the NBox amounts to adding the axiom $\mathsf{House} \sqsubseteq \{a_1, ..., a_n\}$ to the ontology, where $a_1, ..., a_n$ are all (explicit and implicit) instances of House in the ontology. In contrast, defining concept or role names as closed predicates does not impact the semantics of the ontology; it only determines what kind of candidate complement sets can be considered. In fact, when $k = 0$, the set of abductive answers to Q in $\mathcal{O}$ w.r.t. k, S_A and S_C coincides with the set of answers to Q in $\mathcal{O}$ no matter how S_A and S_C are set.

For the abductive CQA problem, we assume that the given ontology $\mathcal{O}$ has been *extensionally reduced* by replacing concept assertions $C(a)$ with $Q_C(a)$ and role assertions $(\neg)r^-(a, b)$ with $(\neg)r(b, a)$ in the ABox, and by adding $Q_C \sqsubseteq C$ to the TBox, where C is a concept appearing in the ABox but is neither an atomic concept nor a negated atomic concept, Q_C is a new globally unique atomic concept corresponding to C, and r is an atomic role appearing in the ABox. Extensionally reducing an ontology does not impact abductive answers to conjunctive queries in the ontology.

4 A Method for the Abductive CQA Problem

The ABox abduction method proposed in [7] encodes the ABox abduction problem into a Prolog program and solves it with Prolog engines. We extend this method to solve the abductive CQA problem in a consistent extensionally reduced DLP ontology $\mathcal{O}$. It has been empirically shown in [7] that the ABox abduction method is feasible only when the translated First-order rules have no equality in heads. To guarantee this, expressing $\mathcal{O}$ in the DLP fragment of OWL 2 DL is not enough, thus we also assume that $\mathcal{O}$ adopts the Unique Name Assumption so that any translated rule with equational head atoms can be con-

verted to a semantically equivalent one by rewriting equational head atoms to inequational atoms and moving them to the rule body.

Compared to the ABox abduction method in [7], the proposed method for abductive CQA has significant extensions. First, the proposed method handles new parameters that are not considered in [7] (i.e. the bounded number k and the set S_C of closed predicates). Second, the proposed method introduces new optimizations based on abducible predicates and closed predicates.

Let $F(\mathcal{O})$ denote the set of First-order rules translated from $\mathcal{O}$ where there are no equational head atoms. The proposed method has the following six steps.

In the first step, for the purpose of optimization the unique minimal model of $F(\mathcal{O})$ is computed and all ground atoms in this model are added to $F(\mathcal{O})$ as *ground facts* (i.e. variable-free rules whose consequence part is not $\bot$), yielding $F'(\mathcal{O})$. We call a concept or role name P an *addable predicate* if there is a sequence of rules $r_1, ..., r_n$ in $F(\mathcal{O})$ (where $n \geq 1$) such that P occurs in the head of r_1, some abducible predicates occur in the body of r_n, and the body of r_i and the head of r_{i+1} have common predicates for all $i \in \{1, ..., n-1\}$. Intuitively, only individual assertions on addable predicates can possibly be added to the unique minimal model of $F(\mathcal{O})$ after individual assertions on abducible predicates are added to $\mathcal{O}$. Consider an arbitrary conjunctive query $Q(\overrightarrow{x}) = \exists \overrightarrow{y} : P(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ that consists of a single atom $P(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ where P is a non-addable or closed predicate. For an arbitrary set $\mathcal{A}$ of individual assertions that will be attached with some abductive answers of Q in $\mathcal{O}$, there is not any individual assertion α on P that is entailed by $\mathcal{O} \cup \mathcal{A}$ but not entailed by $\mathcal{O}$, hence the set of abductive answers of Q in $\mathcal{O}$ coincides with the set of answers of Q in $\mathcal{O}$ no matter how the bounded number k and closed predicates are set. Since the answers of Q in $\mathcal{O}$ can be retrieved from the unique minimal model of $F(\mathcal{O})$, the abductive answers of Q in $\mathcal{O}$ can be directly retrieved from the ground facts in $F'(\mathcal{O})$. Hence, the use of non-addable or closed predicates can yield a more efficient encoding of the abductive CQA problem. To achieve this encoding, the following steps consider $F'(\mathcal{O})$ instead of $F(\mathcal{O})$.

In what follows, we assume that the predicate House is closed and the predicates House, Building, isA, hasFacility, locatesIn and rent are addable.

In the second step, all ground facts in $F'(\mathcal{O})$ are encoded into Prolog rules. For example, the ground fact $\rightarrow \mathsf{isA}(a, b)$ is encoded into the following two Prolog rules, where the last rule is written once in the encoded Prolog program for all ground facts on isA. In Prolog atoms, the prefix "pf_" or "p_" is added to concept or role names, because in the Prolog syntax only for variables the first letter is capitalized, while concept or role names are not variables.

$\mathsf{pf\_isA}(a, b).$

$\mathsf{p\_isA}(X, Y, L_i, L_o) \ :- \ \mathsf{pf\_isA}(X, Y), L_o = L_i.$

In the third step, all *definite rules* (i.e. First-order rules whose consequence part is not $\bot$) in $F'(\mathcal{O})$ that have addable but not closed predicates in heads and have variables are encoded into Prolog rules. Note that any definite rule in $F'(\mathcal{O})$ whose head predicate is non-addable or closed is not encoded, because all instances of the head predicate can be directly retrieved from the ground facts in

$F'(\mathcal{O})$, i.e. retrieved through the Prolog rules generated in the previous step. To manage individual assertions that can be added to $\mathcal{O}$, every non-built-in atom in a definite rule in $F'(\mathcal{O})$ is encoded as a Prolog atom with an extra input argument and an extra output argument. The input (resp. output) argument is a list representing the original (resp. updated) set of added assertions if the atom is on an addable but not closed predicate, or is the empty list otherwise. A list L is of the form $[t_1, ..., t_n]$, where t_i is of the form $(a, \text{“}\mathsf{rdf:type}\text{”}, \mathsf{p\_A})$ or $(a, \mathsf{p\_r}, b)$; it is called *empty* if it is of the form $[\,]$. A list L can be decoded into a set of individual assertions $\{t'_1, ..., t'_n\}$, denoted by $\mathsf{decode}(L)$, where t'_i is rewritten from t_i by rewriting $(a, \text{“}\mathsf{rdf:type}\text{”}, \mathsf{p\_A})$ to a concept assertion $A(a)$ and $(a, \mathsf{p\_r}, b)$ to a role assertion $r(a, b)$. The two extra arguments of a Prolog atom, which is encoded from an atom on non-addable or closed predicate, are set as empty lists, because all instances of the predicate can be directly retrieved from the ground facts in $F'(\mathcal{O})$. In the encoded Prolog rule, the extra input argument of the head atom, L_i, is the extra input argument of the first body atom on addable but not closed predicates, whereas the extra output argument of the head atom, L_o, is the extra output argument of the last body atom on addable but not closed predicates; if there are no body atoms on addable but not closed predicates, an Prolog atom $L_o = L_i$ is added to the body to define what L_o is. For example, the rules R_1 and R_3 in Sect. 2 are encoded into the following two Prolog rules.

$\mathsf{p\_isA}(X, Z, L_i, L_o) \ :- \ \mathsf{p\_isA}(X, Y, L_i, L_1), \mathsf{p\_isA}(Y, Z, L_1, L_o).$
$\mathsf{p\_Building}(X, L_i, L_o) \ :- \ \mathsf{p\_House}(X, [\,], [\,]), L_o = L_i.$

It should be noted that all *constraints* (i.e. First-order rules whose consequence part is $\bot$), such as the rule R_4 in Sect. 2, are not encoded into Prolog rules. This is because constraints are only used to determine if the added assertions are consistent with $\mathcal{O}$. But this consistency checking in a Prolog engine is based on brute-force like search and is generally less efficient than calling an external consistency checker, so all constraints in $F'(\mathcal{O})$ are ignored.

In the fourth step, all Prolog predicates occurring in cycles in the set of Prolog rules generated in the previous step are declared to be *tabled* predicates. Setting a Prolog predicate tabled means that any Prolog atom on this predicate is prevented from calling multiple times. This is a crucial step for guaranteeing termination when calling Prolog atoms in the encoded Prolog program. It is similar to the first step in the ABox abduction method [7], except that Prolog predicates encoded from non-addable or closed predicates are not set as tabled predicates, because they cannot be head predicates of Prolog rules generated in the previous step and will not occur in cycles in the encoded Prolog program.

In the fifth step, the given conjunctive query Q is encoded into a Prolog rule with a nullary head atom go, where every non-built-in atom in Q is also encoded as a Prolog atom with an extra input argument and an extra output argument. Likewise, the input (resp. output) argument is a list representing the original (resp. updated) set of added assertions if the atom is on an addable but not closed predicate, or is the empty list otherwise. In order to output all abductive answers to Q, two Prolog atoms $\mathsf{output}(X_1, ..., X_n, L)$ and fail are added to the body of the encoded Prolog rule, where $X_1, ..., X_n$ are all distinguished variables

in Q, and L is the extra output argument of the last body atom on addable but not closed predicates; if there are no body atoms on addable but not closed predicates, L is set as the empty list. $\mathsf{output}(X_1, ..., X_n, L)$ directly returns true if the tuple $\langle X_1, ..., X_n \rangle$ has been output; otherwise, if L is the empty list, the atom outputs $\langle X_1, ..., X_n \rangle$ and returns true; otherwise, if $\mathcal{O} \cup \mathsf{decode}(L)$ is consistent and every ground atom on closed predicates in the unique minimal model of $F(\mathcal{O}) \cup \mathsf{decode}(L)$ is also in the unique minimal model of $F(\mathcal{O})$ (which is checked by calling an external consistency checker), the atom outputs $\langle X_1, ..., X_n \rangle$ and returns true, else the atom returns fail. In other words, an external consistency checker is called when and only when the tuple $\langle X_1, ..., X_n \rangle$ has not been output and L is not empty. The Prolog atom fail forces the Prolog engine to enumerate all possible instantiations for variables in the encoded Prolog rule when calling go, so as to obtain all abductive answers to Q. Suppose we want to find all renting houses that locate in GZ (Guangzhou) and have a rental price less than 3000 yuan per month, we can express it as the following conjunctive query.

$$Q(x) = \exists y : \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{GZ}) \wedge \mathsf{rent}(x, y) \wedge y < 3000$$

Then the above query can be encoded into the following Prolog rule.

$$\mathsf{go} \ :\!- \ \mathsf{p\_House}(X, [\,], [\,]), \mathsf{p\_locatesIn}(X, \text{“}\mathsf{GZ}\text{”}, [\,], L_1), \mathsf{p\_rent}(X, Y, L_1, L_2), \\ Y < 3000, \mathsf{output}(X, L_2), \mathsf{fail}.$$

In the last step, for every abducible predicate in S_A, two Prolog rules are added. The first rule says that the updated set of added assertions is the original one, if the individual assertion to be added is already in the original set of added assertions. The second rule says that the updated set of added assertions is obtained from the original one by inserting an individual assertion on P, if the number of added assertions in the original set is less than k. For example, suppose the predicate isA is abducible, then the following two Prolog rules are added, where the Prolog atom $\mathsf{dom}(X)$ ensures X to be an individual or a constant and returns true, $\mathsf{in}(t, L)$ sets t as a member of the list L and returns true if t can possibly be grounded to a member of L or returns false otherwise, $\mathsf{less}(L, k)$ returns true iff the the number of members in L is less than k, and $\mathsf{insert}(t, L, L')$ sets L' as the resulting list obtained by inserting t to L and turns true.

$$\mathsf{p\_isA}(X, Y, L_i, L_o) \ :\!- \ \mathsf{in}((X, \mathsf{p\_isA}, Y), L_i), L_o = L_i.$$
$$\mathsf{p\_isA}(X, Y, L_i, L_o) \ :\!- \ \mathsf{less}(L_i, k), \mathsf{dom}(X), \mathsf{dom}(Y), \mathsf{insert}((X, \mathsf{p\_isA}, Y), L_i, L_o).$$

The following theorem shows the correctness of the above encoding method.

Theorem 1. *The encoded Prolog program outputs exactly all abductive answers to* $Q(\overrightarrow{x}) = \exists \overrightarrow{y} : \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ *in* $\mathcal{O}$ *w.r.t.* k, S_A *and* S_C *when calling* go.

Proof. (1) Let $\overrightarrow{t}$ be a tuple of individuals or constants output by the encoded Prolog program, and $\mathcal{A}$ be the set of individual assertions attached with $\overrightarrow{t}$ when $\overrightarrow{t}$ is output. Since backward inference in Prolog is sound, it is clear that $F'(\mathcal{O}) \cup \mathcal{A} \models Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$. Furthermore, when $\overrightarrow{t}$ is output (during calling a Prolog atom on output), it is confirmed that $\mathcal{O} \cup \mathcal{A}$ is consistent, and every individual assertion on closed predicates is not entailed by $\mathcal{O} \cup \mathcal{A}$ unless it is entailed by $\mathcal{O}$. According to the last step of the encoding method, $\mathcal{A}$ should only

consist of individual assertions on abducible predicates and $|\mathcal{A}| \leq k$. Hence, by Definition 1, $\overrightarrow{t}$ is an abductive answer to $Q(\overrightarrow{x})$ in $\mathcal{O}$ w.r.t. k, S_A and S_C.

(2) Let the tuple $\overrightarrow{t}$ be an abductive answer to Q in $\mathcal{O}$ w.r.t. k, S_A and S_C, then there exists a set $\mathcal{A}$ of individual assertions on abducible predicates such that $|\mathcal{A}| \leq k$, $\mathcal{O} \cup \mathcal{A} \models Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$, $\mathcal{O} \cup \mathcal{A}$ is consistent and $\mathcal{O} \models \alpha$ for all individual assertions α on closed predicates such that $\mathcal{O} \cup \mathcal{A} \models \alpha$. The unique minimal model of $F(\mathcal{O}) \cup \mathcal{A}$ is equal to the set of ground atoms occurring in the least fixpoint of $\Pi^{(n)}$, where $\Pi^{(0)} = \emptyset$ and for $n \geq 1$, $\Pi^{(n)} = \{R\sigma \mid R \in F(\mathcal{O}) \cup \mathcal{A}, \sigma$ is a mapping from variables in R to constants in $F(O) \cup \mathcal{A}$ such that all body atoms of $R\sigma$ occur in $\Pi^{(n-1)}\}$. Let Δ_n $(n \geq 1)$ denote the set of ground atoms occurring in $\Pi^{(n)}$.

Consider an arbitrary ground atom α in the unique minimal model of $F(\mathcal{O}) \cup \mathcal{A}$. Let $p(a_1, ..., a_m, L_i, L_o)$ be encoded from α. In case α is on non-addable or closed predicates, it is clear that calling $p(a_1, ..., a_m, [\,], [\,])$ will return true according to the Prolog rules generated in the second step of the encoding method. In other cases, we show by induction on Δ_n that (*) calling the Prolog atom $p(a_1, ..., a_m, L_i, L_o)$ will return true with $\mathsf{decode}(L_o)$ set as a subset of $\mathcal{A}$ when L_o is given as a variable and L_i is given as a specific list such that $\mathsf{decode}(L_i) \subseteq \mathcal{A}$. In what follows, we assume that L_o is a variable and $\mathsf{decode}(L_i) \subseteq \mathcal{A}$. Consider the case where $\alpha \in \Delta_1$. If $\alpha \in \mathcal{A}$, then according to the Prolog rules generated in the last step of the encoding method, calling $p(a_1, ..., a_m, L_i, L_o)$ will return true with $\mathsf{decode}(L_o) \subseteq \mathsf{decode}(L_i) \cup \{\alpha\} \subseteq \mathcal{A}$; otherwise, according to the Prolog rules generated in the second step, calling $p(a_1, ..., a_m, L_i, L_o)$ will return true with $\mathsf{decode}(L_o) = \mathsf{decode}(L_i) \subseteq \mathcal{A}$. Suppose the result (*) holds for all ground atoms in Δ_k $(k \geq 1)$. Consider the case where $\alpha \in \Delta_{k+1} \setminus \Delta_k$. There exists a rule $R \in F(\mathcal{O})$ and a mapping σ from variables in R to constants in $F(\mathcal{O}) \cup \mathcal{A}$ such that all body atoms of $R\sigma$ belong to Δ_k. According to the Prolog rules generated in the third step, there exists an encoded Prolog rule R_e whose head atom is $p(a_1, ..., a_m, L_i, L_o)$ and whose body atoms except $L_o = L_i$ are encoded from body atoms of $R\sigma$. By inductive hypothesis, all body atoms of R_e except $L_o = L_i$, when being called, will return true with their extra output parameters set as some lists L such that $\mathsf{decode}(L) \subseteq \mathcal{A}$. Thus, calling $p(a_1, ..., a_m, L_i, L_o)$ will return true with $\mathsf{decode}(L_o) \subseteq \mathcal{A}$ by triggering R_e.

Since $\mathcal{O}$ is a DLP ontology, all arguments of ground atoms in the unique minimal model of $F(\mathcal{O}) \cup \mathcal{A}$ are constants in $F(\mathcal{O}) \cup \mathcal{A}$. Since $\mathcal{O} \cup \mathcal{A} \models Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}]$, there must be a tuple $\overrightarrow{s}$ of constants in $F(\mathcal{O}) \cup \mathcal{A}$ such that $F(\mathcal{O}) \cup \mathcal{A} \models \alpha$ for every ground atom α in $Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}, \overrightarrow{y} \mapsto \overrightarrow{s}]$. Let $\alpha_1, ..., \alpha_n$ be all ground atoms in $Q(\overrightarrow{x})[\overrightarrow{x} \mapsto \overrightarrow{t}, \overrightarrow{y} \mapsto \overrightarrow{s}]$, and β_i be a Prolog atom encoded from α_i as follows: If α_i is a built-in atom, then β_i is encoded as the corresponding built-in atom in Prolog; else if α_i is on non-addable or closed predicates, β_i is encoded to an atom of the form $p(a_1, ..., a_m, [\,], [\,])$; else β_i is encoded to an atom of the form $p(a_1, ..., a_m, L, L')$, where L is the last parameter of the previous β_j on addable but not closed predicates (if this β_j does not exist, L is the empty list). Consider the Prolog rule for encoding Q which is generated in the fifth step. $\beta_1, ..., \beta_n$ and $\mathsf{output}(\overrightarrow{t}, L)$ will be called one by one when calling go. By

the results proved in the previous paragraph, calling β_i will return true for all $i \in \{1, ..., n\}$, and L must be set as a list such that $\mathsf{decode}(L) \subseteq \mathcal{A}$ before calling $\mathsf{output}(\overrightarrow{t}, L)$. When $\mathsf{output}(\overrightarrow{t}, L)$ is called, if $\overrightarrow{t}$ has not been output, then since $\mathsf{decode}(L) \subseteq \mathcal{A}$, L should be the empty list or satisfy that $\mathcal{O} \cup \mathsf{decode}(L)$ is consistent and every ground atom on closed predicates in the unique minimal model of $F(\mathcal{O}) \cup \mathsf{decode}(L)$ is also in the unique minimal model of $F(\mathcal{O})$, hence $\overrightarrow{t}$ should be output. To conclude, $\overrightarrow{t}$ must be output when calling go. □

5 Experimental Evaluation

We conducted experiments in a real-life application, rental matchmaking, which is based on a rental ontology and handles requests for renting houses. The goal of this application is to provide a suitable rental matchmaking system for residents in China, under the current circumstances that many families in China cannot afford a house.

We manually constructed the RBox and the TBox of the rental ontology, which have 129 logical axioms, 36 concept names and 35 role names, using Protégé (version 4.1)[5], a well-known ontology editor. In addition, we built automatic tools to extract information from existing Websites, including a rental Website[6], a traffic Website[7] and an administrative region Website[8]. We manually wrote annotation rules in these tools to convert the extracted data to individual assertions. We also manually added some individual assertions on the role isA to define that some facilities are more specific than some other facilities. When adding individual assertions annotated from different Websites to the rental ontology, inconsistency occurs because some homonymous entities that belong to disjoint concepts are treated as the same individual. We resolved the inconsistency by using the method proposed in [8] to remove a cardinality-minimal set of axioms. Finally, we obtained a consistent ontology[9] with 32,954 individuals and 1,152,336 logical axioms, where the number of renting houses is 9,248. The ontology is an existentially reduced DLP ontology.

We implemented the proposed method in JAVA and used XSB[10] as the back-end Prolog engine since XSB supports tabled predicates. Considering that the matchmaking results returned by the proposed method have a formal semantics, we did not focus on the quality of the matchmaking results but on the efficiency in computing them, which is a crucial criterion for verifying the feasibility of the method. All experiments were were conducted on a PC with Pentium Dual Core 2.80GHz CPU and 16GB RAM, running Win 7 (64 bit).

We carefully designed ten benchmark queries, shown in Fig. 1, to test the efficiency of the proposed method, where all individuals in these queries are

[5] http://protege.stanford.edu/
[6] http://www.soufun.com/
[7] http://www.8684.cn/
[8] http://www.chinaquhua.cn/
[9] http://jfdu.limewebs.com/papers/rental.zip
[10] http://xsb.sourceforge.net/

$Q_1(x) = \mathsf{House}(x)$

Meaning: *Find all renting houses.*

$Q_2(x) = \exists y : \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{Liwan}) \wedge \mathsf{rent}(x, y) \wedge y \geq 1000 \wedge y \leq 2000$

Meaning: *Find all renting houses in the Liwan district (a district in Guangzhou) whose rental price is from 1000 to 2000 yuan per month.*

$Q_3(x) = \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{Baiyun}) \wedge \mathsf{numOfBedrooms}(x, 2) \wedge \mathsf{numOfLivingRooms}(x, 1) \wedge \mathsf{locatesNear}(x, \mathsf{GDUFS})$

Meaning: *Find all renting houses in the Baiyun district (a district in Guangzhou) that have two bedrooms and one living-room and locate near GDUFS (Guangdong University of Foreign Studies).*

$Q_4(x) = \exists y : \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{Liwan}) \wedge \mathsf{numOfBedrooms}(x, y) \wedge y \geq 1 \wedge y \leq 3 \wedge \mathsf{numOfLivingRooms}(x, 1) \wedge \mathsf{numOfKitchens}(x, 1) \wedge \mathsf{numOfBathrooms}(x, 1)$

Meaning: *Find all renting houses in the Liwan district that have one to three bedrooms, one living-room, one kitchen and one bathroom.*

$Q_5(x) = \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{ZhuJiangNewTown}) \wedge \mathsf{towards}(x, \mathsf{South})$

Meaning: *Find all southward renting houses in the Zhu Jiang New Town.*

$Q_6(x) = \exists y, z : \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{Liwan}) \wedge \mathsf{locatesNear}(x, y) \wedge \mathsf{isOriginalOf}(y, z) \wedge \mathsf{Bus}(z)$

Meaning: *Find all renting houses in the Liwan district that locate near the origin stop of some bus line.*

$Q_7(x) = \exists y_1, y_2 : \mathsf{House}(x) \wedge \mathsf{locatesNear}(x, y_1) \wedge \mathsf{locatesNear}(x, y_2) \wedge y_1 \neq y_2 \wedge \mathsf{isLineOf}(y_1, \mathsf{HEMC\_GDUFS}) \wedge \mathsf{isLineOf}(y_2, \mathsf{HEMC\_GDUFS})$

Meaning: *Find all renting houses locating near two different traffic lines both of which have a stop called HEMC_GDUFS (the section of Guangdong University of Foreign Studies which is in Higher Education Mega Center).*

$Q_8(x) = \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{Liwan}) \wedge \mathsf{hasFacility}(x, \mathsf{Club}) \wedge \mathsf{hasFacility}(x, \mathsf{SportsArea})$

Meaning: *Find all renting houses in the Liwan district that have clubs and sports areas.*

$Q_9(x) = \exists y : \mathsf{House}(x) \wedge \mathsf{numOfBedrooms}(x, 3) \wedge \mathsf{numOfLivingRooms}(x, 2) \wedge \mathsf{rent}(x, y) \wedge y \leq 3000 \wedge \mathsf{hasFacility}(x, \mathsf{Club}) \wedge \mathsf{hasFacility}(x, \mathsf{SportsArea})$

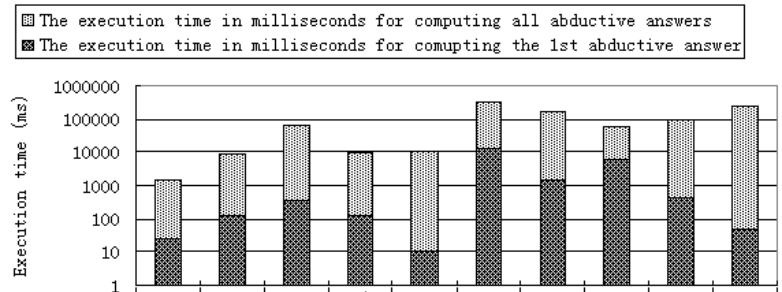
Meaning: *Find all renting houses that have three bedrooms and two living-rooms, have a rental price no more than 3000 yuan per month, and have clubs and sports areas.*

$Q_{10}(x) = \exists y : \mathsf{House}(x) \wedge \mathsf{locatesIn}(x, \mathsf{GZ}) \wedge \mathsf{floorNo}(x, y) \wedge \mathsf{numOfFloors}(x, z) \wedge y \neq z$

Meaning: *Find all renting houses in GZ (Guangzhou) that are not at the top floor of a building.*

Fig. 1. The benchmark queries for testing the proposed method

automatically generated URIs that correspond to names in Chinese, but they are shown by meaningful names here for readability. The first query is the basic one which can be directly answered by the original rental Website. The next four queries are complex queries about multiple aspects of renting houses. The 6th query and the 7th query are complex queries about renting houses and traffic



	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
#avg. time(ms)	0.14	110.21	230.18	38.19	21.82	764.73	613.04	363.27	210.21	27.30
#abd. answers	9248	81	277	235	468	388	269	146	426	8615
#cons. checks	0	0	184	0	0	179	112	120	196	1419

Fig. 2. The statistics for each benchmark query (Note: in the bottom table, row 1 shows the average execution time in milliseconds for computing one abductive answer, row 2 shows the number of abductive answers, and row 3 shows the number of calls to an external consistency checker.)

lines, and they are not supported by the original rental Website. The last three queries involve more complex reasoning. For example, the 8th query and the 9th query involve reasoning on the rules R_1 and R_2 in Sect. 2.

We tested the proposed method on computing all abductive answers to every benchmark query in the rental ontology w.r.t. k, S_A and S_C, where $k = 1$, $S_A = \{\mathsf{locatesNear}, \mathsf{hasFacility}, \mathsf{rent}, \mathsf{towards}, \mathsf{floorNo}, \mathsf{numOfFloors}\}$ (which consists of all predicates on which the information may be incomplete) and $S_C = \{\mathsf{House}\}$ (which consists of one predicate on which the information is surely complete).

Our implemented system works in two phases. In the first phase, all information except specific conjunctive queries is encoded into a Prolog program, which is then loaded into XSB. This phase is independent of any given query and is performed offline. In our experiments, this phase was done in 875 seconds. In the second phase, every benchmark query is encoded into a Prolog rule. Then this rule is added to the Prolog program obtained in the first phase and is evaluated by XSB. The statistics in this phase are shown in Fig. 2. For 7/1/2 benchmark queries, the first abductive answer was computed in 1/2/13 seconds. For 4/3/3 benchmark queries, all abductive answers were computed in 10/100/300 seconds. For all benchmark queries, each abductive answer was computed in one second on average. Note that all benchmark queries have abductive answers in the rental ontology and the evaluation of six benchmark queries needs to call an external consistency checker. This shows that the system is able to efficiently handle nontrivial requests for renting houses.

6 Conclusion and Future Work

We have proposed a new semantic approach to matchmaking based on the abductive CQA semantics. By considering that data are often incomplete in Web-based

scenarios, this approach defines matches for a request as abductive answers to a conjunctive query that expresses the request. Furthmore, we proposed a sound and complete method for computing all abductive answers in a consistent extensionally reduced DLP ontology. Experimental results on rental matchmaking demonstrated the feasibility of the proposed method.

For future work, we plan to define reasonable measures for ranking abductive answers. These measures can be computed according to the minimal sets of added assertions attached with abductive answers. We also plan to define another extended CQA semantics to perform matchmaking in an inconsistent and incomplete ontology without rendering the ontology consistent beforehand.

Acknowledgement

Jianfeng Du and Shuai Wang are partly supported by the NSFC under grant 61005043 and the Undergraduate Innovative Experiment Project in Guangdong University of Foreign Studies. Guilin Qi is partly supported by Excellent Youth Scholars Program of Southeast University under grant 4009001011, the NSFC under grant 61003157, Jiangsu Science Foundation under grant BK2010412, and the Key Laboratory of Computer Network and Information Integration (Southeast University). Jeff Z. Pan is partly supported by the EU K-Drive project and the RCUK dot.rural project. Yong Hu is partly supported by the NSFC under grant 70801020.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. Batabyal, A.A., DeAngelo, G.J.: To match or not to match: Aspects of marital matchmaking under uncertainty. Operations Research Letters 36(1), 94–98 (2008)
3. Bianchini, D., Antonellis, V.D., Melchiori, M.: Flexible semantic-based service matchmaking and discovery. World Wide Web 11(2), 227–251 (2008)
4. Cimiano, P., Völker, J.: Text2onto - a framework for ontology learning and data-driven change discovery. In: Proc. of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB). pp. 227–238 (2005)
5. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. Electronic Commerce Research and Applications 4(4), 345–361 (2005)
6. Colucci, S., Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M., Mottola, M.: A formal approach to ontology-based semantic match of skills descriptions. Journal of Universal Computer Science 9(12), 1437–1454 (2003)
7. Du, J., Qi, G., Shen, Y., Pan, J.Z.: Towards practical abox abduction in large OWL DL ontologies. In: Proc. of the 25th AAAI Conference on Artificial Intelligence (AAAI). pp. 1160–1165 (2011)

8. Du, J., Shen, Y.: Computing minimum cost diagnoses to repair populated DL-based ontologies. In: Proc. of the 17th International World Wide Web Conference (WWW). pp. 265–274 (2008)
9. Fazzinga, B., Gianforme, G., Gottlob, G., Lukasiewicz, T.: Semantic web search based on ontological conjunctive queries. In: Proc. of the 6th International Symposium on Foundations of Information and Knowledge Systems (FoIKS). pp. 153–172 (2010)
10. Fenza, G., Loia, V., Senatore, S.: A hybrid approach to semantic web services matchmaking. International Journal of Approximate Reasoning 48(3), 808–828 (2008)
11. Grau, B.C., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P.F., Sattler, U.: OWL 2: The next step for OWL. Journal of Web Semantics 6(4), 309–322 (2008)
12. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proc. of the 12th International World Wide Web Conference (WWW). pp. 48–57 (2003)
13. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR). pp. 57–67 (2006)
14. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Proc. of the 12th International World Wide Web Conference (WWW). pp. 331–339 (2003)
15. Noia, T.D., Sciascio, E.D., Donini, F.M., Mongiello, M.: A system for principled matchmaking in an electronic marketplace. In: Proc. of the 12th International World Wide Web Conference (WWW). pp. 321–330 (2003)
16. Ragone, A., Straccia, U., Noia, T.D., Sciascio, E.D., Donini, F.M.: Fuzzy matchmaking in e-marketplaces of peer entities using datalog. Fuzzy Sets and Systems 160(2), 251–268 (2009)
17. Ren, Y., Pan, J.Z., Zhao, Y.: Closed world reasoning for OWL2 with NBox. Journal of Tsinghua Science and Technology 15(6), 692–701 (2010)
18. Shu, G., Rana, O.F., Avis, N.J., Chen, D.: Ontology-based semantic matchmaking approach. Advances in Engineering Software 38(1), 59–67 (2007)