I N F S Y S

R E S E A R C H

R E P O R T

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG WISSENSBASIERTE SYSTEME

A KNOWLEDGE-BASED APPROACH FOR SELECTING INFORMATION SOURCES

Thomas EITER Michael FINK Hans TOMPITS

INFSYS RESEARCH REPORT 1843-03-14

DECEMBER 2003

Institut für Informationssysteme
Abtg. Wissensbasierte Systeme
Technische Universität Wien
Favoritenstraße 9-11
A-1040 Wien, Austria
Tel: +43-1-58801-18405
Fax: +43-1-58801-18493
sek@kr.tuwien.ac.at
www.kr.tuwien.ac.at

TECHNISCHE UNIVERSITÄT WIEN

INFSYS RESEARCH REPORT
INFSYS RESEARCH REPORT 1843-03-14, DECEMBER 2003

A KNOWLEDGE-BASED APPROACH FOR SELECTING INFORMATION SOURCES

Thomas Eiter, Michael Fink, and Hans Tompits [1]

Abstract. Through the Internet and the World Wide Web, a vast number of information sources has become available, which offer information on various subjects by different providers, often in heterogeneous formats. This calls for tools and methods for building an advanced information processing infrastructure. One issue in this area is the selection of suitable information sources in query answering. In this paper, we present a knowledge-based approach to this problem, in the setting where one out of a set of information sources (prototypically, data repositories) should be selected for evaluating a user query. We use extended logic programs (ELPs) to represent rich descriptions of the information sources, an underlying domain theory, and user queries in a formal query language (here, XML-QL, but other languages can be handled as well). Moreover, we use ELPs for declarative query analysis and query abstraction. Central to our approach are declarative *source-selection programs*, for which we define syntax and semantics. Due to the structured nature of the considered data items, the semantics of such programs must carefully respect implicit context information in source-selection rules, and furthermore combine it with possible user preferences. A prototype implementation of our approach has been realized exploiting the `DLV` KR system and its `plp` front-end for prioritized ELPs. We describe here a representative example involving particular movie databases, and report about experimental results.

[1] Institut für Informationssysteme, Abteilung Wissensbasierte Systeme 184/3, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Wien, Austria. E-mail: [eiter, michael, tompits]@kr.tuwien.ac.at

Acknowledgements: This work was supported by the Austrian Science Fund (FWF) projects P13871-INF and N Z29-INF.
We acknowledge Compaq Equipment Corporation for providing the EachMovie Database.

This report replaces INFSYS Research Report 1843-02-08, entitled "A Knowledge-Based Approach To Information Site Selection", which contained material in preliminary form.

Copyright © 2003 by the authors

Contents

1 Introduction **1**

2 Overview of the Approach **2**
2.1 Source-selection capability . . . 3

3 Preliminaries **4**

4 Query Representation and Abstraction **6**
4.1 Low-level representation . . . 7
4.2 High-level predicates . . . 8
4.3 Query abstraction program . . . 9

5 Source Description **12**

6 Source Selection **13**
6.1 Syntax . . . 13
6.2 Semantics . . . 15

7 Properties of Source-Selection Programs **19**

8 Implementation and Application **23**
8.1 Application: Movie databases . . . 23
8.1.1 Movie sources . . . 23
8.1.2 Domain knowledge . . . 24
8.1.3 Source-selection program . . . 26

9 Experiments **27**
9.1 Results . . . 28
9.2 Results with Modified Selection Bases . . . 29

10 Related Work **31**
10.1 Information integration systems . . . 31
10.2 Other work . . . 33

11 Conclusion **33**

A Appendix: XML DTD for the Movie Databases **39**

B Appendix: Low-level Query-Representation Examples **40**

C Appendix: High-level Query-Description Examples **42**

D Appendix: Experimental Movie Doamin Knowledge **43**

E Appendix: Experimental Source-Description Program **55**

F Appendix: Experimental Source-Selection Program **58**

G Appendix: Experimental XML-QL Queries **64**

1 Introduction

Through the Internet and the World Wide Web (WWW), a wealth of information has become available to a large group of users. A huge number of documents, files, and data repositories on a range of subjects are offered by different providers, which may be non-profit individuals, organizations, or companies. Such data repositories are currently in heterogeneous formats, but the trend is that XML becomes a future de-facto standard for putting data on the Web, since this eases data exchange. Nonetheless, the quality of their contents may differ a lot with respect to aspects such as accuracy, coverage of resp. completeness for certain topics, or refresh period, to mention a few.

Accessing and processing data on the Web calls for developing tools and methods for an advanced information processing infrastructure. *Mediators* [62] and special *information agents* ("middle agents" [14]), which provide various services including finding, selecting, and querying relevant information sources, play an important role here. The potential of knowledge-based approaches—and in particular of logic programming—for developing reasoning components for intelligent information agents is recognized in the AI community and outlined, e.g., in [18, 23, 55].

In this paper, we pursue this issue further and present a declarative approach to information source selection in the following setting. Given a query by a user in some formal query language, and a suite of information sources on which this query might may be evaluated, which one is the best to answer the query, i.e., such that the utility of the answer, measured by the quality of the result and other criteria (e.g., costs), is as large as possible for the user? Note that in fact, this problem is not bound to information sources on the Web, and is of interest in any context where different candidate information sources (e.g., scientific databases, newspaper archives, stock exchange predictions, etc.) are available and one of them should be selected.

Obviously, a sensible solution to this problem is nontrivial and involves various aspects such as basic properties of the information sources, knowledge about their contents, and knowledge about the particular application domain. These aspects have to be suitably combined, and reasoning may be needed to elicit implicit knowledge. We stress that the general problem considered here is distinct from a simple keyword-based search as realized by Web-engines such as Google [33], and consequently we do not propose here a method for competing with these tools.[1]

Our approach, which incorporates aspects mentioned above, makes several contributions, which are briefly summarized as follows.

(1) We use extended logic programs (ELPs) under the answer-set semantics [31] to represent rich descriptions of the information sources, an underlying domain theory, and queries in a formal language. We perform query analysis by ELPs and compute *query abstractions*. Here, we consider XML-QL [17], but other formal query languages can be handled as well; e.g., [56] considers the ubiquitous SQL language for relational databases.

(2) At the heart, a declarative *source-selection program* represents both *qualitative* and *quantitative* criteria (e.g., source preference and costs). Its rules may access information supplied by the other programs, including object and value occurrences in the query. For example, a rule r_1 may state that a query about actor awards should be posed to source s_1.

(3) We consider the interesting and, to our knowledge, novel issue of *contexts* in logic programs. Structured data items require a careful definition of the selection semantics. If, besides r_1 as above, a

[1] Google does not index XML-files or databases behind a Web query interface, and hence could not be readily applied.

rule r_2 says that source s_2 is in general best for awards, then r_1 should have priority over r_2 and s_1 should be queried. Note that this priority is not based on inheritance, but on the *context* of the occurrence of awards, which is given by the query access path. Thus, inheritance-based approaches such as [9] do not apply obviously. Furthermore, implicit priorities, derived from context information, must be combined with explicit user preferences from the selection policy, and arising conflicts must be resolved.

(4) We have implemented a prototype, which we used to build an example application involving movie information sources. It comprises several XML databases, wrapped from movie databases on the Web, and handles queries in XML-QL. Experiments that we have conducted show that the system behaved intuitively on a number of natural queries, some of which require reasoning from the background knowledge to identify the proper selection.

Our motivation to use a knowledge-based approach—and in particular an answer-set programming approach—to source selection rather than a standard decision theoretic approach based on utility functions is driven by the following advantages:

- Source-selection programs are declarative and have a well-defined formal semantics, both under qualitative as well as quantitative criteria.
- The formalism is capable of handling incomplete information and performing nonmonotonic inferences, which, arguably, is an inherent feature of the current problem domain.
- Changes in the specification of the source-selection process are easily incorporated by modifying or adding suitable rules or constraints, without the need for re-designing the given program, as might be the case, e.g., in procedural languages.
- Finally, the declarative nature of the answer-set-semantics formalism permits also a coupling with sophisticated ontology tools, and with reasoning engines for them, providing advanced features for the domain knowledge.

The rest of this paper is organized as follows. The next section briefly outlines our approach, and Section 3 introduces some background terminology. In Section 4, we describe elementary query representation and abstraction to high level concepts. Section 5 addresses the modeling of sources, while Section 6 is devoted to source-selection programs. In Section 7, we then study some properties of source-selection programs. The implementation and the movie application are addressed in Section 8, while experiments are reported in Section 9. Section 10 discusses related work and Section 11 concludes the paper.

2 Overview of the Approach

A prototypical source-selection setup is shown in Figure 1, where data are stored in XML databases. Queries are posed in an XML query-language such as XML-QL [17], which we use here. Some of the databases may be wrapped from non-XML data sources. However, our approach is not committed to semi-structured data and XML; other models and languages (e.g., relational databases and SQL) may also be used.

A query is handed over to an information agent, which has to pick one of several databases that comply with the same (universal) schema to answer the query. The answers to queries may differ on aspects such as contents, quantity, cost, reliability, and so on. Merging data from different sources, query rewriting, etc.,

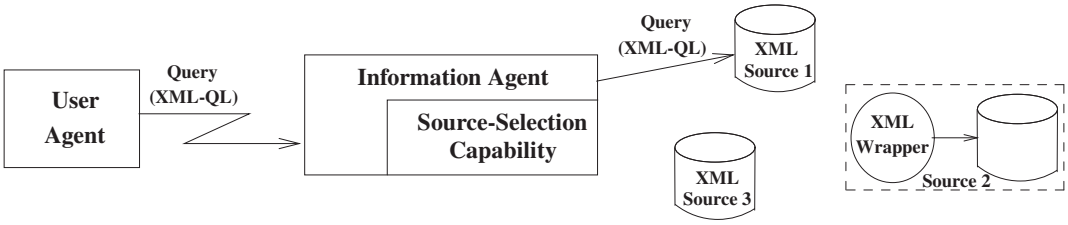

Figure 1: Possible architecture of a source-selection system

do not play a role here. The architecture in Figure 1 is only one of several possible architectures one can image. For example, there might be multiple such information agents in a system, avoiding a centralization bottleneck; or, the site selection capability may reside not with a special source-selection agent, but be part of a more powerful mediator agent.

2.1 Source-selection capability

Our approach is based on the *answer-set programming* paradigm [53], in which problems are represented by extended logic programs (ELPs) [31], possibly augmented with priorities (e.g., [8, 16, 39]) and weak constraints [10, 43], and solutions are obtained from the answer sets of the programs (see Section 3). The following scenario illustrates our ideas.

Example 1 Consider an agent which has access to XML information sources s_1, s_2, and s_3 about movies. Assume that the following XML-QL [17] query is handed to the agent, which informally asks a source for titles of all movies directed by Alfred Hitchcock:

```
FUNCTION HitchcockMovies($MovieDB:"Movie.dtd") {
  CONSTRUCT <MovieList> {
    WHERE <MovieDB> <Movie>
             <Title> $t </Title>
             <Director> <Personalia>
                 <FirstName> "Alfred" </FirstName>
                 <LastName> "Hitchcock" </LastName>
             </Personalia> </Director>
          </Movie> </MovieDB>
    IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
 } </MovieList> }
```

Suppose the agent knows that s_1 is a very good source for information about directors, while s_2 has usually good coverage about person data; all known about s_3 is that it is not much reliable. In this situation, we would expect the agent to select s_1 for querying.

While the above example is simple, it shows that the selection process uses different kinds of knowledge. In our approach, this is reflected by the following components of a *selection base* $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ (see Fig. 2):

Query description Π_{qd}: For any query Q as in Example 1, a high-level abstract description is extracted from a low-level (syntactic) representation $R(Q)$, given as a set of elementary facts, by applying a

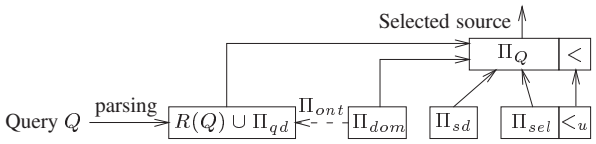

Figure 2: Using a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ for source selection

logic program Π_{qd} to $R(Q)$ and ontological knowledge Π_{ont}. Informally, the rules of Π_{qd} single out the essential parts of Q, such as occurrence of attributes and values in the query, comparison and joins, or access paths describing contexts.

Domain theory Π_{dom}: The agent's knowledge about the application domain (e.g., movies) is represented in a logic program Π_{dom}. It includes an ontology, Π_{ont}, and further background knowledge, permitting (modest) common-sense reasoning. The ontology may be established using meta-information about the data in the information sources (e.g., an XML DTD).

Source description Π_{sd}: Information about the sources is represented in a logic program Π_{sd}, using special predicates for query topics, cost aspects, and technical aspects.

Source-selection program Π_{sel}: The information source selection is specified by rules and constraints, which refer to predicates defined in the above programs. It comprises both *qualitative aspects* and *quantitative aspects* in terms of optimization criteria (concerning, e.g., cost or response time), which are expressed using weak constraints [10]. Furthermore, the user can define preferences between rules, in terms of a strict partial order, $<_u$. These preferences are combined with implicit priorities that emerge from the *context* in which source selection rules should be applied, and possible preference conflicts are resolved.

Given a query Q, the overall evaluation proceeds in three steps:

Step 1 (query representation): The input query Q is parsed and mapped into the internal query representation, $R(Q)$.

Step 2 (qualitative selection): From $R(Q)$, Π_{sd}, and Π_{dom}, the qualitative part of Π_{sel} is used to single out different query options by respecting qualitative aspects only, where explicit preferences, $<_u$, and implicit priorities must be taken into account. To this end, a priority relation $<$ is computed on rules, which is then used in a prioritized logic program $(\Pi_Q, <)$. Candidate solutions are computed as preferred answer sets of $(\Pi_Q, <)$.

Step 3 (optimization): Among the candidates of Step 2, the one is chosen which is best under the quantitative aspects of Π_{sel}, and the selected source is output.

3 Preliminaries

We briefly recall the basic concepts from answer-set programming. An *extended logic program* (ELP) [31] is a finite set of rules r in a function-free first order language of form

$$L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n,$$

where each L_i, $0 \leq i \leq n$, is a *literal*, i.e., an atom A or a negated atom $\neg A$; "*not*" denotes negation as failure. Intuitively, rule r means that we can conclude L_0 if (i) $L_1, \ldots, L_m$ are known and (ii) $L_{m+1}, \ldots, L_n$ are *not* known. We call the literal L_0 the *head* of r (denoted $H(r)$) and the set $\{L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\}$ the *body* of r (denoted $B(r)$). Furthermore, we define $B^+(r) = \{L_1, \ldots, L_m\}$ and $B^-(r) = \{L_{m+1}, \ldots, L_n\}$. If $B(r) = \emptyset$, then r is called a *fact*. To ease notation, for any program Π and any set S of literals, we write $\Pi \cup S$ to refer to the program $\Pi \cup \{L \leftarrow |\ L \in S\}$. Finally, for literal L, we write $\neg L$ to denote its complementary literal, i.e., $\neg L = A$ if $L = \neg A$, and $\neg L = \neg A$ if $L = A$, for any atom A.

The semantics of an ELP Π is given in terms of the semantics of its ground instantiation, $ground(\Pi)$, over the Herbrand universe. In what follows, assume that all objects (literals, rules, etc.) are ground.

An *interpretation*, I, is a consistent set of (ground) literals, i.e., I does not contain a complementary pair A, $\neg A$ of literals. A literal L is *true* in I (in symbols, $I \models L$) if $L \in I$ and *false* otherwise. The body $B(r)$ of a rule r is true in I (denoted $I \models B(r)$) iff (i) each $L \in B^+(r)$ is true in I and (ii) each $L \in B^-(r)$ is false in I. Rule r is true in I ($I \models r$) iff $H(r)$ is true in I whenever $B(r)$ is true in I. I is a *model* of program Π, denoted $I \models \Pi$, if $I \models r$ for all $r \in \Pi$.

The *reduct*, Π^S, of a program Π *relative to* a set S of literals is defined by

$$\Pi^S = \{H(r) \leftarrow B^+(r) \mid r \in \Pi \text{ and } B^-(r) \cap S = \emptyset\}.$$

An interpretation I is an *answer set* of a program Π, if I is a minimal model of Π^I with respect to set inclusion. Observe that any answer set of Π is *a fortiori* a model of Π. The set of all *generating rules* of an answer set S from Π is given by $GR(\Pi, S) = \{r \in \Pi \mid S \models B(r)\}$.

Example 2 Let $\Pi = \{\, s \leftarrow not\, t;\ n \leftarrow ;\ t \leftarrow n, not\, s;\ w \leftarrow t\,\}$. For the interpretation $S = \{n, t, w\}$, we have $\Pi^S = \{n \leftarrow ;\ t \leftarrow n;\ w \leftarrow t\}$. Clearly, S is a minimal model of Π^S, and thus S is an answer set of Π. Note that $S' = \{s, n\}$ is another answer set of Π.

A (possibly non-ground) program Π is *locally stratified* [54], iff there exists a mapping λ assigning each literal occurring in $ground(\Pi)$ a natural number, such that for rule $r \in ground(\Pi)$ it holds that (i) $\lambda(H(r)) \geq \max_{L \in B^+(r)} \lambda(L)$ and (ii) $\lambda(H(r)) > \max_{L \in B^-(r)} \lambda(L)$; note that Π is (globally) *stratified* (cf. [2]), if moreover $\lambda(L) = \lambda(L')$ holds for all positive (resp., negative) literals L and L' with the same predicate. It is well-known that if a program is locally stratified, then it has at most one answer set.

A refinement of the answer-set semantics is the admission of preferences among the rules of a given ELP, yielding so-called *prioritized logic programs*. Several approaches along this way have been introduced in the literature, like, e.g., [8, 39]; here, we use a preference approach based on a method due to Delgrande, Schaub, and Tompits [16], defined as follows.

Let Π be an ELP and $<$ a strict partial order between elements of Π (i.e., $<$ is an irreflexive and transitive relation). Informally, for rules $r, s \in \Pi$, the relation $r < s$ expresses that s has preference over r. Define the relation $<_{\mathcal{G}}$ over the ground instantiation $ground(\Pi)$ of Π by setting $\hat{r} <_{\mathcal{G}} \hat{s}$ iff $r < s$, for $\hat{r}, \hat{s} \in ground(\Pi)$. Then, the pair $(\Pi, <)$ is called a *prioritized extended logic program*, or simply a *prioritized logic program*, if the relation $<_{\mathcal{G}}$ is a strict partial order.

The semantics of prioritized programs is as follows. Let $(\Pi, <)$ be a prioritized logic program where Π is ground, and let S be an answer set of Π. We call S a *preferred answer set of* $(\Pi, <)$ iff there exists an enumeration $\langle r_i \rangle_{i \in I}$ of $GR(\Pi, S)$ such that, for every $i, j \in I$, we have that:

(P_1) $B^+(r_i) \subseteq \{H(r_k) \mid k < i\}$;

(P_2) if $r_i < r_j$, then $j < i$; and

(P_3) if $r_i < r'$ and $r' \in \Pi \setminus GR(\Pi, S)$, then $B^+(r') \not\subseteq S$ or $B^-(r') \cap \{H(r_k) \mid k < i\} \neq \emptyset$.

Conditions (P_1) – (P_3) realize a strongly "prescriptive" interpretation of preference, in the sense that, whenever $r < s$ holds, it is ensured that s is known to be applied or blocked ahead of r (with respect to the order of rule application). More specifically, (P_2) guarantees that all generating rules are applied according to the given order, whilst (P_3) assures that any preferred yet inapplicable rule is either blocked due to the non-derivability of its prerequisites or because it is defeated by higher ranked or unrelated rules. As shown in [16], the selection of preferred answer sets can be encoded by means of a suitable translation from prioritized logic programs into standard ELPs.

Preferred answer sets of a prioritized program $(\Pi, <)$ where Π is non-ground are given by the preferred answer sets of the prioritized program $(ground(\Pi), <_{\mathcal{G}})$, where $<_{\mathcal{G}}$ is as above. Note that the concept of prioritization realizes a *filtering* of the answer sets of a given program Π, as every preferred answer set of $(\Pi, <)$ is an answer set of Π, but not vice versa.

Besides imposing *qualitative selection criteria*, like assigning preferences between different rules, another refinement of the answer-set semantics are *weak constraints* [10, 43], representing a *quantitative filtering of answer sets*. Formally, a weak constraint is an expression of form

$$\Leftarrow\ L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\ [w : l], \tag{1}$$

where each L_i, $1 \leq i \leq n$, is a literal and $w, l \geq 1$ are natural numbers. The number w is the *weight* and l is the *priority level* of the weak constraint (1). Informally, weak constraints work as follows: From the answer sets of the weak-constraint free part of a program Π, prune first those where the sum of weights of violated constraints in the highest priority level is not minimal, then those where the sum of weights of violated constraints in the next lower level is not minimal, and so on. We refer the reader to [43] for the precise formal definition.[2]

4 Query Representation and Abstraction

An important aspect of our approach is a meaningful description of a given query Q. For our purposes, we need a suitable representation of the constituents of Q in terms of predicates and objects. Simply mapping Q to logical facts reflecting its syntactic structure, as given by the query grammar, will not serve our purposes. Rather, we need an *abstract* description which provides "interesting" information about Q, such as occurrence of an attribute or a value in Q, related to the scope of appearance. For example, the value "Hitchcock" occurs in the query of Example 1 in a selection condition, which is performed on the attribute $LastName$ reached by the access path $Movie/Director/Personalia$.

In our query abstraction method, we adopt a general view in which a query has a *construct part*, a *where part*, and a *source part*, corresponding to the SQL *select*, *where*, and *from parts*, respectively. We use a high-level description of Q independent of a fixed query language. It is extracted from a low-level query representation, $R(Q)$, which is generated by a parser and contains structural details, using a logic program Π_{qd} and relevant ontology knowledge Π_{ont}. In detail, the high-level description is given by the (unique) answer set of $\Pi_{ont} \cup \Pi_{qd} \cup R(Q)$.

[2] The part "$[w : l]$" is convenient syntactic sugar for the original definition in [10], which merely provided a partitioning of the weak constraints in priority levels.

4.1 Low-level representation

The representation $R(Q)$ is the first level of abstraction from the "physical" (syntactic) query expression Q. The query and its syntactic subqueries are named by constants (e.g., $q_1, q_2, \ldots$). References to items (attributes or concepts) which are selected or compared to values or other items are given by maximal access paths in Q, which we name by unique constants and refer to as *item reference paths* (*IRPs*).

Example 3 In Example 1, the query is named q_1, and has a subquery q_2 in the construct `<MovieList>` part. There are three IRPs, namely "$Movie/Title$", "$Movie/Director/Personalia/FirstName$", and "$Movie/Director/Personalia/LastName$", which we name o_1, o_2, and o_3, respectively.

The program $R(Q)$ is expressed in terms of the following predicates:

- $sub_query(Q', Q)$: Q' is a structural subquery of query Q (possibly itself a subquery);
- $query_cand(Q)$: identifies the overall query;
- $source(S, Q)$: query Q accesses source S;
- $db_name(S)$: source S is a database;
- $whereRef(O, T, P, Q)$: an IRP O accesses an item under element T and remainder path P in the where part of query Q;
- $subpath(O, T_1, P_1, T_2, P_2)$: the path T_2/P_2 is a direct subpath of T_1/T_2 in the IRP O;
- $whereRefCmp(O_1, R, O_2)$: the items of IRPs O_1 and O_2 are compared using operator R;
- $whereCmp(O, R, V)$: the item of IRP O is compared to value V using operator R;
- $consRef(O, T, P)$: the item of IRP O is constructed under element T and remaining path P in the (answer-) construction part of query Q;

$R(Q)$ must respect that query languages may allow for nested queries. However, in a query expression, an outmost query as the "root" of nesting should be identifiable, as well as structural (syntactic) subqueries of it. They are described using $query_cand$ and sub_query, respectively.

Along an IRP, item references relative to a position are captured by the $whereRef$ predicate, and suffix inclusions for this IRP are stored as $subpath$ facts. The predicates $whereRefCmp$ and $whereCmp$ mirror the comparison of two items and the comparison of an item with a value, respectively. Items that occur in the construction part of a query are also identified by an IRP and stored using $consRef$.

Example 4 The low-level representation $R(Q)$ of the query in Example 1, contains

$$sub_query(q_2, q_1); \quad query_cand(q_1); \quad source(\text{“}MovieDB\text{”}, q_2); \quad db_name(\text{“}MovieDB\text{”}),$$

and, e.g., for the third IRP, o_3, which accesses "*LastName*", the facts:

$whereRef\,(o_3, \text{“}LastName\text{”}, \text{“ ”}, q_2)$;
$whereRef\,(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2)$;
$whereRef\,(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_2)$;
$whereRef\,(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_2)$;
$whereRef\,(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_2)$;
$subpath(o_3, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}LastName\text{”})$;
$subpath(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”})$;
$subpath(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”})$;
$subpath(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, \text{“}MovieDB\text{”},$
$\text{“}Movie/Director/Personalia/LastName\text{”})$;
$whereCmp(o_3, equal, \text{“}Hitchcock\text{”})$.

The complete low-level representation $R(Q)$ of the query is given in Appendix B.

4.2 High-level predicates

The high-level description enriches and abstracts from $R(Q)$, and brings in ontological knowledge. The following notion is convenient.

Definition 1 *A pair (C, P), where C is a concept from the ontology and P is a path, is a* context-reference pair (*CRP*) *of a query Q, if Q contains an IRP with suffix “C/P”.*

In Example 1, $(\text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”})$ and $(\text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”})$ are some CRPs of the query q_2.

The following high-level description predicates are defined:

- $query(Q)$: identifies an “independent” (sub-)query Q (i.e., executable on some source), which is, moreover, not embraced by a sourceless query Q' merely restructuring the result of Q;
- $access(O, C, P, Q)$: states that (C, P) is a CRP for Q via IRP O in the where-part of Q;
- $occurs(O, V)$: the value V is associated with an IRP O in the overall query;
- $selects(O, R, V)$: like $occurs$, but details the association with a comparison operator R;
- $constructs(O, I, P)$: states that the item of IRP O, by use of a variable, also appears in the construct-part of the global query, as an item I under path P (which may be different from the path in the where-part);
- $joins(O_1, O_2, R)$: records (theta-)joins of (or within) queries between IRPs O_1 and O_2 under comparison operator R.

Example 5 For the query in Example 1, we have $query(q_1)$, but not $query(q_2)$ since the embracing query q_1 has no source and merely structures the result of q_2. The following $access$ facts result from o_1 and o_3:

$access(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_1)$;
$access(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_1)$;
$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1)$;
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1)$;
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1)$;
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$.

Here, $MovieDB$, $Movie$, $Director$, and $Person$ are concepts given by the ontology, and $Personalia$ is known to be a synonym of $Person$ (see Section 4.3 for further discussion).

The fact $occurs(o_3, \text{“}Hitchcock\text{”})$ states that value $Hitchcock$ is associated with o_3. This is detailed by $selects(o_3, equal, \text{“}Hitchcock\text{”})$, where $equal$ represents equality. For the $constructs$ predicate, the fact $constructs(o_1, \text{“}Movie\text{”}, \text{“ ”})$ is included. There are no $joins$ facts since the query has no join. The complete high-level description is given in Appendix C.

4.3 Query abstraction program

The high-level description facts are obtained by applying the query abstraction program, Π_{qd}, using the facts of $R(Q)$ and some ontological facts Π_{ont} as input. Π_{qd} is given by a set of specific rules, which we present in the rest of this section. They compute the predicates from Section 4.2, where auxiliary predicates are used. They furthermore enlarge the low-level predicate $subpath$, by taking it, as well as Π_{ont}, into account as follows:[3]

$$
\begin{array}{lrcl}
qd_1: & subpath(O,T_1,P_1,T_3,P_3) & \leftarrow & subpath(O,T_1,P_1,T_2,P_2), subpath(O,T_2,P_2,T_3,P_3);\\
qd_2: & subpath(O,T,P_1,T_2,P_2) & \leftarrow & subpath(O,L,P_1,T_2,P_2), synonym(L,T);\\
qd_3: & subpath(O,T_1,P_1,T,P_2) & \leftarrow & subpath(O,T_1,P_1,L,P_2), synonym(L,T).
\end{array}
$$

Rule qd_1 expresses transitivity for elements occurring in paths, and qd_2 and qd_3 deal with synonyms, which is imported ontological knowledge; $synonym$ applies to all pairs of synonymous element names (e.g., names of IDREF attributes).

The following two rules define useful projections of low-level predicates:

$$
\begin{array}{lrcl}
qd_4: & has_source(Q) & \leftarrow & source(_,Q);\\
qd_5: & is_sub_query(Q) & \leftarrow & sub_query(Q,_).
\end{array}
$$

Using them, an auxiliary predicate $iquery_cand$ is defined for candidates which may satisfy the $query$ predicate; these are the overall query and subqueries having a database or a document as its source:

$$
\begin{array}{lrcl}
qd_6: & iquery_cand(Q) & \leftarrow & query_cand(Q);\\
qd_7: & iquery_cand(Q) & \leftarrow & is_sub_query(Q), source(X,Q), db_name(X).
\end{array}
$$

The high-level predicates are now defined as follows. Independent, separate queries are specified by respecting the nesting structure:

$$
\begin{array}{lrcl}
qd_8: & top_query(S,Q) & \leftarrow & sub_query(S,X), not\ iquery_cand(S), top_query(X,Q);\\
qd_9: & top_query(S,Q) & \leftarrow & sub_query(S,X), iquery_cand(S), top_query(X,Q), not\ has_source(X);\\
qd_{10}: & top_query(Q,Q) & \leftarrow & iquery_cand(Q), not\ is_sub_query(Q);\\
qd_{11}: & top_query(Q,Q) & \leftarrow & iquery_cand(Q), sub_query(Q,S), source(X,S);\\
qd_{12}: & query(Q) & \leftarrow & top_query(Q,Q).
\end{array}
$$

A query is considered to be independent if it is the topmost independent query of itself (qd_{12}). This is the case if the query is a candidate for a separate query and it is either the outmost query (qd_{10}) or a direct structural subquery of a query to a source (qd_{11}). Intuitively, qd_{11} says that a candidate query nested within another query is viewed as a separate query only if the nesting was not for mere syntactic reasons. In this

[3] In a clean separation of $R(Q)$ and the high-level description, a fresh predicate would be in order here. However, it is convenient and economic to reuse the predicate $subpath$, as it is only enlarged.

case, and if a nested query is not a candidate for a separate query, its topmost independent query is the one of the embracing query (qd_9 resp. qd_8).

The next rules define the remaining high-level description predicates.

The auxiliary predicate $has_constructs$ guarantees that at least one $constructs$ fact is generated for each context reference constructed in the query answer.

$$
\begin{array}{lrcl}
qd_{13}: & access(O,T,P,Q) & \leftarrow & whereRef(O,T,P,S), class0(T), top_query(S,Q);\\
qd_{14}: & access(O,T,P,Q) & \leftarrow & whereRef(O,L,P,S), synonym(L,T), class0(T), top_query(S,Q);\\
qd_{15}: & constructs(O,T,P) & \leftarrow & consRef(O,T,P), class0(T);\\
qd_{16}: & constructs(O,T,P) & \leftarrow & consRef(O,L,P), synonym(L,T), class0(T);\\
qd_{17}: & has_constructs(O) & \leftarrow & consRef(O,T,P), class0(T);\\
qd_{18}: & has_constructs(O) & \leftarrow & consRef(O,L,P), synonym(L,T), class0(T);\\
qd_{19}: & constructs(O,\text{“ ”},\text{“ ”}) & \leftarrow & consRef(O,_,_), not\ has_constructs(O);\\
qd_{20}: & occurs(O,V) & \leftarrow & whereCmp(O,C,V);\\
qd_{21}: & selects(O,C,V) & \leftarrow & whereCmp(O,C,V);\\
qd_{22}: & joins(O_1,O_2,C) & \leftarrow & whereRefCmp(O_1,C,O_2).
\end{array}
$$

Note that some rules reference the ontology predicate $class0$. A fact $class0(e)$ should exist in (or being entailed by) the domain ontology for all elements e that are considered to be concepts.

When queries are joined over CRPs, then some of the occurrence, selection, and construction information of one CRP is also valid for the other. Hence, we can build a form of a closure over joined CRPs, which is expressed by the following rules:

$$
\begin{array}{lrcl}
qd_{23}: & constructs(O_1,T,P) & \leftarrow & joins(O_1,O_2,equal), constructs(O_2,T,P);\\
qd_{24}: & constructs(O_2,T,P) & \leftarrow & joins(O_1,O_2,equal), constructs(O_1,T,P);\\
qd_{25}: & occurs(O_1,V) & \leftarrow & joins(O_1,O_2,C), occurs(O_2,V);\\
qd_{26}: & occurs(O_2,V) & \leftarrow & joins(O_1,O_2,C), occurs(O_1,V);\\
qd_{27}: & selects(O_1,C,V) & \leftarrow & joins(O_1,O_2,equal), selects(O_2,C,V);\\
qd_{28}: & selects(O_2,C,V) & \leftarrow & joins(O_1,O_2,equal), selects(O_1,C,V);\\
qd_{29}: & selects(O_1,notequal,V) & \leftarrow & joins(O_1,O_2,notequal), selects(O_2,equal,V);\\
qd_{30}: & selects(O_2,notequal,V) & \leftarrow & joins(O_1,O_2,notequal), selects(O_1,equal,V).
\end{array}
$$

We remark that, as easily seen, the rules of Π_{qd} form a (locally) stratified logic program, and thus $\Pi_{ont} \cup \Pi_{qd} \cup R(Q)$ has a unique answer set.

Example 6 Let us consider how the high-level fact $access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$ is derived in Π_{qd}, given $R(Q)$ of the query in Example 1.

Since $query_cand(q_1)$ is in $R(Q)$, we obtain by qd_6 $iquery_cand(q_1)$. Since $is_sub_query(q_1)$ is not derivable, by qd_{10} we can derive $top_query(q_1, q_1)$ (i.e., that q_1 is independent). Next, we can derive $is_sub_query(q_2)$ in view of qd_5 and thus $iquery_cand(q_2)$ by qd_7, given that $R(Q)$ includes

$$sub_query(q_2, q_1),\ source(\text{“}MovieDB\text{”}, q_2), \text{ and } db_name(\text{“}MovieDB\text{”}).$$

Since q_1 has no source (i.e., $has_source(q_1)$ is not derivable), we can thus derive $top_query(q_2, q_1)$ from qd_9. The fact $access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$ is now derived by means of qd_{14}, using

$$whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2)$$

from $R(Q)$, as well as $synonym(\text{“}Personalia\text{”}, \text{“}Person\text{”})$ and $class0(\text{“}Person\text{”})$ from the ontology, and the derived fact $top_query(q_2, q_1)$. Note that $access(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_1)$ is not derivable, as $class0(\text{“}Personalia\text{”}) \notin \Pi_{ont}$.

While our method works well for experimental queries that we considered (cf. also Section 9), we can not state that it is complete in some sense, and refinements will be desired. Due to syntactical dependencies and its generalizing nature, "intelligent" declarative query abstraction is, per se, an interesting area for further research, as the following example illustrates.

Example 7 Consider the more complex XML-QL query asking for movies directed by Josef Sternberg in which Marlene Dietrich acted:

```
FUNCTION SternbergDietrichMovies($MovieDB:"Movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE <MovieDB> <Movie>
        <Title> $t </Title>
           <Director> <Personalia>
              <FirstName> "Josef" </FirstName>
              <LastName> "Sternberg" </LastName>
           </Personalia> </Director>
           <Actor Personalia=$pid>
           </Actor>
        </Movie>
        <Person ID=$pid>
             <FirstName> "Marlene" </FirstName>
             <LastName> "Dietrich" </LastName>
        </Person> </MovieDB>
     IN source($MovieDB)
     CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

This query, like the one of Example 1, complies with the XML-DTD in Appendix A. However the first- and last name information of director Josef Sternberg is accessed syntactically different than the respective information of actress Marlene Dietrich. The context references concerning Sternberg are implicit joins over `Personalia` of type IDREF, while the context references concerning Dietrich make this join explicit.[4] This syntactical difference is reflected in the high-level description of the query, which is provided in Appendix C. Consider, e.g., the following facts and compare the access predicates for last name Sternberg with those for Dietrich:

$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1).$
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1).$
$access(o_4, \text{“}MovieDB\text{”}, \text{“}Movie/Actor/Personalia\text{”}, q_1).$
$access(o_5, \text{“}MovieDB\text{”}, \text{“}Person/ID\text{”}, q_1).$
$access(o_7, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}, q_1).$
$access(o_7, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1).$
$joins(o_4, o_5, equal).$

[4]Currently, XML-QL (version 0.9) does not support implicit joins over IDREF-references. They must be written as explicit joins over the corresponding attribute (e.g., as for the context references concerning Marlene Dietrich).

Although the explicit join is reflected in the high-level description, its connection with the name references has been lost.

These differences cannot be eliminated without the underlying XML-DTD. Thus, to keep the query abstraction process generic, syntactic issues like these must be tackled by the query parser, i.e., the low-level representation. Note, however, that these representational differences have no influence on the site-selection process and experiments in Sections 8 and 9, respectively.

5 Source Description

Besides query information and domain knowledge, source selection needs a suitable description of the information sources to select from. This is provided by means of meta-knowledge represented in the source-description part of the knowledge base, given in the form of a (simple) logic program, which is assumed to have a unique answer set. Different predicates can be used for this purpose, depending on the specific application. In the following, we introduce, in an exemplary way, a basic suite of predefined *source-description predicates*, which cover several aspects of an information source.

Thematic aspects:

$accurate(S, T, V)$: source S, topic T, value V;
$covers(S, T, V)$: source S, topic T, value V;
$specialized(S, T)$: source S, topic T;
$relevant(S, T)$: source S, topic T.

The first two predicates express the accuracy and coverage of a source for a topic, using values from $\{low, med, high\}$. The others are for stating that a source is specialized or relevant for a particular topic, respectively.

Cost aspects:

$avg_download_time(S, V)$: source S, value V;
$avg_down_time(S, V)$: source S, value V.
$charge(S, V)$: source S, value V;

Costs for accessing an information source can be expressed by these predicates, again using values low, med, $high$, and, for $charge$, also no. While $charge$ is used for direct costs, $avg_download_time$ and avg_down_time are indirect costs (affecting network traffic).

Technical aspects:

$source_type(S, T_1, T_2)$: source S, organizational type T_1, query type T_2;
$source_language(S, L)$: source S, language L;
$data_format(S, F)$: source S, format F;
$update_frequency(S, V)$: source S, value V;
$last_update(S, D)$: source S, date F;
$reliable(S, V)$: source$S$, value V;
$source(S)$: source S;
$up(S)$: source S.

Different types of sources are distinguished by their organization (commercial or public) and by the type of data access provided (queryable, downloadable, or both). Besides source language and data format (XML, relational, HTML, text, or other), the frequency of data update (low, medium, or high), the date of the last update, or the reliability of a source (low, medium, or high) may be criteria for source selection. Finally, $source$ and up are used to identify sources and to express that a source is currently accessible, respectively.

As pointed out above, the above predicates are just a rudiment of a vocabulary for source description, and we are far from claiming that they capture all aspects and each aspect in sufficient detail resp. granularity (e.g., the three-valued scale used). However, the user resp. administrator has the possibility to introduce further predicates and define them in the source-description program Π_{sd}. Note that Π_{sd} can profitably take advantage of default rules to handle incomplete information, e.g., that a source is accessible by default, or that the language of text items is English.

As insinuated in the previous paragraph, we assume here that detailed source descriptions are edited by an administrator of an information system in whose application area the selection process takes place. This does not preclude that a preliminary or partial description is created automatically, addressing aspects such as source language, type, data format, etc., nor that the information system is open for new sources entering it, advertising their description to a source registration. However, a number of aspects for selection, such as coverage, specialization, or relevance, might be difficult to assess automatically and require experience gained from interaction with a source like in real-life scenarios (think of, e.g., different travel agencies offering flights). Here, the administrator might bring in such knowledge initially, and the description might be updated according to new information obtained, e.g., by performance monitoring and user feedback. For updating an employed description, approaches such as [1, 22] may be applied. In general, this is a complex and interesting issue, but it is beyond the scope of this paper.

We remark furthermore that the proviso that Π_{sd} possesses a unique answer set can be ensured, e.g., by requiring (local) stratification of Π_{sd}, or by the condition that its well-founded model is total. In principle, multiple answer sets of Π_{sd} could be admitted as well, which give rise to different scenarios that can be handled in different ways; e.g., adhering to a credulous resp. skeptical principle, according to which the different scenarios are considered *en par* resp. such that only selections in all scenarios are retained, or to a preference-based approach which discriminates between the different scenarios. However, we do not further elaborate on this issue.

6 Source Selection

In this section, we introduce source-selection programs, which are central to our architecture. Basically, a source-selection program Π_{sel} is a prioritized logic program having four parts, namely (i) a core unit Π^c_{sel}, containing the actual source-selection rules, (ii) a set Π^{aux}_{sel} of auxiliary rules, (iii) an order relation $<_u$ defined over members of Π^c_{sel}, and (iv) an optimization part Π^o_{sel}, containing weak constraints.

6.1 Syntax

Definition 2 *The alphabet, $\mathcal{A}_{sel}$, of a source-selection program Π_{sel} in a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ consists of the following pairwise disjoint categories:*

- *the function-free alphabets $\mathcal{A}_{qd}$, $\mathcal{A}_{sd}$, and $\mathcal{A}_{dom}$ of the query description Π_{qd}, the source description Π_{sd}, and the domain theory Π_{dom}, respectively;*
- *the predicate*

  $$query_source(S, Q),$$

  expressing that source S is selected for evaluating query Q;
- *the predicates*

$default_class(O, C, Q)$ *and* $default_path(O, P, Q)$*,*

which are projections of $access(O, C, P, Q)$*; and*

- *a set* $\mathcal{A}_{aux}$ *of auxiliary predicates;*

Informally, $default_class(O, C, Q)$ and $default_path(O, P, Q)$ serve to specify a default status for selection rules depending on context-reference pairs matched in the query Q. For example, $default_class(O,$ "$Person$", $Q)$ in the body of rule r says that r is eligible, if the concept $Person$ occurs in the access path O and there is no other rule s that refers to some CRP (C', P') matched in Q. These defaults are semantically realized using a suitable rule ordering.

Rules are constructed as in Section 3. We use upper-case letters for variables and lower-case letters for constants. We write $r(X_1, \ldots, X_n)$ to indicate that rule r has variables $X_1, \ldots, X_n$. The set of all literals over atoms in $\mathcal{A}_\ell$, where $\ell \in \{qd, sd, dom, aux, sel\}$, is denoted by Lit_ℓ.

Definition 3 *A* source-selection program over $\mathcal{A}_{sel}$ *is a tuple* $(\Pi_{sel}, <_u)$*, where*

(*i*) Π_{sel} *is a collection of rules over* $\mathcal{A}_{sel}$ *consisting of the following parts:*

(*a*) *the* core unit Π^c_{sel}*, containing rules of form*

$$query_source(S, Q) \quad \leftarrow \quad L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n;$$

(*b*) *a set* Π^{aux}_{sel} *of* auxiliary rules *of form*

$$L_0 \quad \leftarrow \quad L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n;$$

(*c*) *an* optimization part Π^o_{sel} *containing weak constraints of form*

$$\Leftarrow\ L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n\ [w : l],$$

where L_0 *is either a literal from* Lit_{aux} *or is of form* $\neg query_source(\cdot, \cdot)$*, and* $L_i \in Lit_{sel}$ *for* $1 \leq i \leq n$*, and* $w, l \geq 1$ *are integers.*

(*ii*) *The relation* $<_u$ *is a strict partial order between rules in* Π^c_{sel}*; the elements of* $<_u$ *are called* user-defined preferences*. If* $r <_u s$*, then* s *is said to have* preference over r.

The rules in the core unit Π^c_{sel} serve for selecting a source, based on information from the domain description Π_{dom}, the source description Π_{sd}, the query description $R(Q) \cup \Pi_{qd}$, and possibly from auxiliary rules. The latter may be used, e.g., for evaluating complex conditions. In terms of $<_u$, preference of source selection can be expressed. As well, the weak constraints in Π^o_{sel} are used to filter answer sets under quantitative conditions.

Example 8 Consider a simple source-selection program, $(\Pi_{sel}, <_u)$, for our movie domain:

```
% source selection rules
```

r_1: $query_source(s_2, Q) \leftarrow default_class(O,$ "$Person$", $Q)$;

r_2: $query_source(s_1, Q) \leftarrow selects(O, equal,$ "$Hitchcock$"$),$
$access(O,$ "$Director$", "$Personalia/LastName$", $Q)$;

r_3: $query_source(S, Q) \leftarrow default_path(O,$ "$LastName$", $Q)$, $default_class(O, T, Q)$,

$accurate(S, T, high)$;

% auxiliary rules

r_4: $high_acc(T, Q) \leftarrow access(O, T, P, Q), accurate(S, T, high)$;

r_5: $high_cov(T, Q) \leftarrow access(O, T, P, Q), covers(S, T, high)$;

% optimization constraints

c_1: $\Leftarrow query_source(S, Q), high_acc(T, Q), not\ accurate(S, T, high)\ [10:1]$;

c_2: $\Leftarrow query_source(S, Q), high_cov(T, Q), not\ covers(S, T, high)\ [5:1]$;

% user preferences

$r_1(Q, _) <_u r_3(Q, _, _, _)$.

Intuitively, r_1 advises to choose source s_2 if the query involves persons and no more specific rule is eligible. Rule r_2 says to choose source s_1 if the query contains an explicit select on the movie director Hitchcock. Rule r_3 demands to choose a source if, on some query access path, "*LastName*" is accessed under some concept T (with arbitrary intermediate access path), and the source is highly accurate for T. Rules r_4 and r_5 define auxiliary predicates which hold on concepts T appearing in the query such that some source with high accuracy and coverage, respectively, for T exists. The weak constraints c_1 and c_2 state penalties for choosing a source that does not have high accuracy (weight 10) resp. coverage (weight 5) for a concept in the query while such a source exists. Finally, $r_1(Q, _) <_u r_3(Q, _, _, _)$ expresses preference of instances of r_3 over r_1 on the same query.

6.2 Semantics

The semantics of a source-selection program $(\Pi_{sel}, <_u)$ in a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ on a query Q is given by a *selection answer set* of $(\Pi_{sel}, <_u)$, which is defined as a preferred answer set of a prioritized ELP $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$ associated with $\mathcal{S}$ and Q. The program Π_Q contains ground instances of rules and constraints in Π_{sel}, and further rules ensuring that a single source is selected per query and defining the default context predicates. The order relation $<$ is formed from the user preferences $<_u$ and the implicit priorities derived from context references in the core unit and from auxiliary rules. In that, we must suitably combine preference information and handle conflicts, which we do by a cautious conflict elimination policy.

We commence the formal details with the following notation: For any rule r, its *defaultization*, r^Δ, is given by $H(r) \leftarrow B(r), not\ \neg H(r)$. We assume that user-defined preferences between rules carry over to their defaultizations.

Definition 4 *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ be a selection base and Q a query. Then, the program Π_Q contains all ground instances of the rules and constraints in $\Pi^{aux}_{sel} \cup \Pi^{o}_{sel}$, as well as all ground instances of the following rules:*

(*i*) *the defaultization r^Δ of r, for each $r \in \Pi^{c}_{sel}$;*

(*ii*) *the* structural rule

$$\neg query_source(S, Q) \leftarrow query_source(S', Q), S \neq S'; \quad (2)$$

(*iii*) *the* default context rules

$$default_class(O, C, Q) \leftarrow access(O, C, _, Q); \quad (3)$$
$$default_path(O, P, Q) \leftarrow access(O, _, P, Q). \quad (4)$$

Intuitively, the defaultization makes the selection rules in Π_{sel} defeasible with respect to the predicate $query_source$; the structural rule enforces that only one source is selected; and the default context rules define the two default predicates. Since our language has no function symbols, Π_Q is finite, and its size depends on the constants appearing in Π_{qd}, $R(Q)$, Π_{sd}, and Π_{dom}.

For $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ and query Q, we call any answer set of $\Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}$ a *selection input of $\mathcal{S}$ for Q*. The set of all selection inputs of $\mathcal{S}$ for Q is denoted by $Sel(\mathcal{S}, Q)$. For $I \in Sel(\mathcal{S}, Q)$, we define

$$I_{def} = I \cup \{ default_class(o, c, q),\ default_path(o, p, q) \mid access(o, c, p, q) \in I \}.$$

Notice that, in general, a selection base may admit multiple selection inputs for a query Q. However, in many cases, there may exist only a single selection input—in particular, if the source description Π_{sd} and the domain knowledge Π_{dom} have unique answer sets. In our framework, this is ensured if, e.g., these components are represented by locally stratified programs.

Definition 5 *A rule $r \in \Pi_Q$ is* relevant for Q *iff there is some $I \in Sel(\mathcal{S}, Q)$ such that $B^\dagger(r)$ is true in I_{def}, where $B^\dagger(r)$ results from $B(r)$ by deleting each element which does not contain a predicate symbol from $\mathcal{A}_{qd} \cup \mathcal{A}_{sd} \cup \mathcal{A}_{dom} \cup \{ default_class, default_path \}$.*

In the sequel, we denote for any binary relation R its transitive closure by R^*.

We continue with the construction of the preference relation $<$, used for interpreting a source-selection program $(\Pi_{sel}, <_u)$ relative to a selection base $\mathcal{S}$ and a query Q, in terms of an associated prioritized logic program $(\Pi_Q, <)$.

Informally, the specification of $<$ depends on the following auxiliary relations:

- the preference relation $\preceq_c$, taking care of implicit context priorities;
- the intermediate relation $\trianglelefteq$, representing a direct combination of user-defined preferences with context preferences; and
- the preference relation $<'$, removing possible conflicts within the joined relation $\trianglelefteq$ and ensuring transitivity of the resultant order $<$.

More specifically, the relation $\preceq_c$ is the first step towards $<$, transforming structural context information into explicit preferences, in virtue of the following specificity conditions:

- relative to the same IRP and query, rules that mention default concepts or a CRP matching the query are considered more specific than rules mentioning default paths;
- default contexts for concepts are assumed to be more specific than default contexts for attributes; and
- with respect to the same IRP, rules with a larger CRP (C, P) are considered more specific than rules with a shorter CRP (C', P') (i.e., P' is a subpath of P).

The second step in the construction of $<$ is the relation $\trianglelefteq$, which is just the union of the user preferences $<_u$ and the context priorities $\preceq_c$. In general, this will not be a strict partial order. To enforce irreflexivity, we remove all priorities $n_r \trianglelefteq n_s$ which lie on a cycle $n_r \trianglelefteq n_s \trianglelefteq n_{s_1} \trianglelefteq \cdots \trianglelefteq n_{s_n} \trianglelefteq n_r$, resulting in $<'$. Finally, taking the transitive closure of $<'$ yields $<$. The formal definition of relation $<$ is as follows.

Definition 6 *Let $\mathcal{S}$ be a selection base, Q a query, and Π_Q as in Definition 4. We introduce the following binary relations over the rules in Π_Q: For $r, s \in \Pi_Q$,*

1. *$r \preceq_c s$ iff r, s are relevant for Q, $r \neq s$, and one of (O_1)–(O_3) holds:*

   (O_1) *$default_path(o_1, p_1, q) \in B(r)$, and we have either $default_class(o_2, t_2, q) \in B(s)$ or $access(o_2, t_2, p_2, q) \in B(s)$;*

   (O_2) *$default_class(o_1, t_1, q) \in B(r)$ and $access(o_2, t_2, p_2, q) \in B(s)$;*

   (O_3) *$access(o, t_1, p_1, q) \in B(r)$, $access(o, t_2, p_2, q) \in B(s)$, and t_1/p_1 is a subpath of t_2/p_2;*

2. *$r \trianglelefteq s$ iff $r <_u s$ and r, s are relevant for Q, or $r \preceq_c s$;*

3. *$r <' s$ iff $r \trianglelefteq s$ but not $s \trianglelefteq^* r$.*

Then, the relation $<$ is given as the transitive closure of relation $<'$.

Example 9 Reconsider $(\Pi_{sel}, <_u)$ from Example 8. Suppose the domain ontology contains the concepts "*MovieDB*", "*Actor*", "*Movie*", "*Director*", and "*Person*", and that "*Personalia*" and "*Person*" are synonymous. Assume further that the query Q of Example 1 has a unique selection input I, containing the following source-description facts:

$accurate(s_1, \text{“}Director\text{”}, high)$;
$covers(s_2, \text{“}Person\text{”}, high)$;
$reliable(s_3, low)$,

plus the following facts resulting from the query description and the default context rules:

$default_path(o_3, \text{“}LastName\text{”}, q_1)$;
$default_class(o_3, \text{“}Person\text{”}, q_1)$;
$default_class(o_3, \text{“}Director\text{”}, q_1)$;
$selects(o_3, equal, \text{“}Hitchcock\text{”})$;
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$;
$default_class(o_2, \text{“}Person\text{”}, q_1)$;
$access(o_2, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1)$;
$access(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_1)$;
$access(o_2, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1)$;
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1)$.

These elements are exactly those contributing to relevant instances of Π_Q. The relevant instances of r_1, r_2, and r_3 are given by the ground rules $r_1(q_1, o_2)$, $r_2(q_1, o_3)$, $r_2(q_1, o_3)$, and $r_3(q_1, s_1, o_3, \text{“}D\text{”})$ (for brevity, we write here "D" for "$Director$"). Intuitively, we expect $r_2(q_1, o_3)$ to have highest priority among these rule instances, since the bodies of the instances of r_1 and r_3 contain default predicates while r_2 references a specific context. Actually, the order relation $<$ includes for the relevant instances of r_1, r_2, and r_3 the following pairs:

$$r_1(q_1, o_2) < r_2(q_1, o_3), \quad r_1(q_1, o_3) < r_2(q_1, o_3), \quad \text{and} \quad r_3(q_1, s_1, o_3, \text{“}D\text{”}) < r_2(q_1, o_3).$$

Note that both r_4 and r_5 have two relevant instances. However, they do not influence the above rule ordering. Informally, they are either unrelated to or "ranked between" $r_2(q_1, o_3)$ and the relevant instances of r_1 and r_3 (since the $access$ predicates of r_4 and r_5 refer to the same context as the context referenced in the body of r_2, or to a subpath of such a context). Hence, the relevant instance of r_2 has highest priority.

As for r_1 and r_3, the auxiliary relation $\trianglelefteq$ contains two further structural priorities, namely

$$r_3(q_1, s_1, o_3, \text{“}D\text{”}) \preceq_c r_1(q_1, o_2) \quad \text{and} \quad r_3(q_1, s_1, o_3, \text{“}D\text{”}) \preceq_c r_1(q_1, o_3).$$

They are in conflict with the user preferences

$$r_1(q_1, o_2) <_u r_3(q_1, s_1, o_3, \text{“}D\text{”}) \quad \text{and} \quad r_1(q_1, o_3) <_u r_3(q_1, s_1, o_3, \text{“}D\text{”}),$$

respectively. This is resolved in the resultant relation $<$ by removing these preferences.

Note that in Definition 6, the final order $<$ enforces a cautious conflict resolution strategy, in the sense that it remains "agnostic" with respect to priority information causing conflicts. Alternative definitions of $<'$, such as removal of a minimal cutset eliminating all cycles in $\trianglelefteq$, may be considered as well; however, this may lead to a nondeterministic choice since, in general, multiple such cutsets exist. Different choices lead to different orders $<$, which may lead to different results of the source-selection program. Thus, unless a well-defined *particular* minimal cutset is singled out, by virtue of preference conflicts, the result of the source-selection process might not be deterministic. Furthermore, an extended logic program component computing a final order based on minimal cutsets is more involved than a component computing the relations in Definition 6.

Combining Definitions 4 and 6, we obtain the translation $\mathcal{E}(\cdot, \cdot)$ as follows:

Definition 7 *Let $\mathcal{S}$ be a selection base and Q a query. Then, the evaluation $\mathcal{E}(\mathcal{S}, Q)$ of $\mathcal{S}$ with respect to Q is given by the prioritized logic program $(\Pi_Q, <)$, where Π_Q and $<$ are as in Definitions 4 and 6, respectively.*

Selection answer sets of source-selection programs are then obtained as follows:

Definition 8 *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{sd}, \Pi_{dom}, \Pi_{sel}, <_u \rangle$ be a selection base, and let $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$ for query Q. Then, $S \subseteq Lit_{sel}$ is a* selection answer set of $(\Pi_{sel}, <_u)$ for Q with respect to $\mathcal{S}$ *iff S is a preferred answer set of the prioritized logic program $(\Pi_Q \cup I, <)$ for some $I \in Sel(\mathcal{S}, Q)$. A source s is* selected for Q *iff $query_source(s, q)$ belongs to some selection answer set of $(\Pi_{sel}, <_u)$ for Q (with respect to $\mathcal{S}$), where the constant q represents Q.*

Example 10 In our running example, $(\Pi_{sel}, <_u)$ has a unique selection answer set S with respect to $\mathcal{S}$ for the query Q (represented by q_1) from Example 8. It contains $query_source(s_1, q_1)$, which is derived from the core rule $r_2(q_1, o_3)$, having the highest priority among the applicable rules leading to a single preferred answer set for the weak-constraint free part of Π_{sel}. If we replace, e.g., r_1 by the rule

$$query_source(s_2, Q) \leftarrow access(O, \text{“}Person\text{”}, P, Q)$$

and adapt the corresponding user preference to $r_1(Q, _, _) <_u r_3(Q, _, _, _)$, then the weak-constraint free part of Π_{sel} has two preferred answer sets: one, S_1, is identical to S (where applying $r_2(q_1, o_3)$ is preferred to applying $r_1(q_1, o_3)$, given that $r_1(q_1, o_3) < r_2(q_1, o_3)$); in the other answer set, S_2, the rule $r_1(q_1, o_2)$ is applied and $query_source(s_2, q_1)$ is derived. Informally, the replacement removes the preference of

$r_2(q_1, o_3)$ over $r_1(q_1, o_2)$, since their $access$ predicates refer to different contexts ("$.../FirstName$" and "$.../LastName$", respectively). Consequently, $r_1(q_1, o_2)$ has maximal preference like $r_2(q_1, o_3)$.

Given that S_1 has weight 5, caused by violation of $c_2(s_1, q_1, \text{"}Person\text{"})$, but S_2 has weight 10, caused by violation of $c_1(s_2, q_1, \text{"}Director\text{"})$, S_1 is the selection answer set of $(\Pi_{sel}, <_u)$ for Q.

7 Properties of Source-Selection Programs

In this section, we discuss some properties of our framework. To this end, we introduce the following concept: Given logic programs Π_1 and Π_2, we say that Π_1 is *independent* of Π_2, if each predicate symbol occurring in some rule head of Π_2 does not occur in Π_1. Intuitively, if Π_1 is independent of Π_2, then Π_1 may serve as an "input" for Π_2. This idea is made precise by the following proposition, which is an immediate consequence of results in [24, 47]:

Proposition 1 *Let Π_1 and Π_2 be two extended logic programs, possibly containing weak constraints, and let S be a set of ground literals. If Π_1 is independent of Π_2, then S is an answer set of $\Pi_1 \cup \Pi_2$ iff there is some answer set I of Π_1 such that S is an answer set of $\Pi_2 \cup I$.*

Now, taking the specific structure of our source-selection architecture into account, we obtain the following characterization, which links the evaluation method of source-selection programs to the usual semantics of prioritized logic programs.

Theorem 1 *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ be a selection base, let Q be a query, and let $\mathcal{E}(\mathcal{S},Q) = (\Pi_Q, <)$. Then, S is a selection answer set of $(\Pi_{sel}, <_u)$ for Q with respect to $\mathcal{S}$ iff S is a preferred answer set of $(\Pi_Q \cup \Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}, <)$.*

Proof. Let Π_0 denote the program $\Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom}$. Recall that S is a selection answer set of $(\Pi_{sel}, <_u)$ for Q with respect to $\mathcal{S}$ iff S is a preferred answer set of $(\Pi_Q \cup I, <)$, for some answer set I of Π_0. Since the predicate symbols occurring in the heads of rules in a source-selection program do not occur in rules from the query description, the source description, or the domain theory, we obviously have that Π_0 is independent of Π_Q. Moreover, it holds that S is a preferred answer set of $(\Pi_Q \cup I, <)$ only if S is an answer set of $\Pi_Q \cup I$. Hence, applying Proposition 1, we have that S is an answer set of $\Pi_Q \cup I$ for some answer set I of Π_0 iff S is an answer set of $\Pi_Q \cup \Pi_0$. From this, the assertion of the theorem is an immediate consequence. □

Strengthening Theorem 1, the construction of $\mathcal{E}(\mathcal{S}, Q)$ can itself be realized in terms of a *single* logic program, by describing preference relations *directly at the object level*. For this purpose, we introduce a set N of constants serving as names for rules, and a new binary predicate $pref(\cdot, \cdot)$, defined over N, expressing preference between rules. The extended alphabet $\mathcal{A}_{sel} \cup \{pref(\cdot, \cdot)\} \cup N$ is denoted by $\bar{\mathcal{A}}_{sel}$. We furthermore assume an injective function $n(\cdot)$ which assigns to each rule $r \in \Pi_Q$ a name $n(r) \in N$. To ease notation, we also write n_r instead of $n(r)$. Finally, Lit_{pref} denotes the set of literals having predicate $pref$. Note that $Lit_{pref} \cap Lit_{sel} = \emptyset$.

Theorem 2 *Let $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$ be a selection base, let Q be a query, and let $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$. Furthermore, let $\Pi_{\mathcal{S}} = \Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd} \cup \Pi_Q$. Then, there exists a logic program Π_{obj} over an alphabet $\hat{\mathcal{A}} \supseteq \bar{\mathcal{A}}_{sel}$ such that every answer set S of $\Pi_{\mathcal{S}} \cup \Pi_{obj}$ satisfies the following conditions:*

1. *$S \cap Lit_{pref}$ represents $<$, i.e., $pref(n_r, n_{r'}) \in S$ iff $r < r'$; and*

2. *$S \cap Lit_{sel}$ is a selection answer set of $(\Pi_{sel}, <_u)$ for Q with respect to $\mathcal{S}$ iff S is a preferred answer set of the prioritized program $(\Pi_{\mathcal{S}} \cup \Pi_{obj}, <)$.*

Proof. We describe Π_{obj} here, but omit the detailed argument that it works properly. Informally, Π_{obj} has two parts: the first, Π_{rel}, is derived from $\Pi_{\mathcal{S}}$ and takes care of computing the relevant rules for Q, by utilizing weak constraints; the second part, Π_{pref}, is a locally stratified logic program and determines the relations in Definition 6. We start with the construction of Π_{rel}.

For each predicate $p \in \mathcal{A}_{sel}$ and each $r \in \Pi_Q$, we introduce a new predicate $\overline{p}_r$ of the same arity as p. In addition, we introduce a new atom rel_r, informally expressing that rule r is relevant. If α is either a literal, a set of literals, a rule, or a program, then by $\lceil \alpha \rceil_r$ we denote the result of uniformly replacing each atom $p(x_1, , \ldots, x_n)$ occurring in α by $\overline{p}_r(x_1, \ldots, x_n)$.

For each $r \in \Pi_Q$, we define a program Π_r containing the following items:

(i) each rule in $\lceil \Pi_{qd} \cup R(Q) \cup \Pi_{sd} \cup \Pi_{dom} \rceil_r$;

(ii) the rule $rel_r \leftarrow \lceil B^{\dagger}(r) \rceil_r$; and

(iii) the extended default context rules

$$\overline{default_class}_r(O, C, Q) \leftarrow \overline{access}_r(O, C, _, Q);$$
$$\overline{default_path}_r(O, P, Q) \leftarrow \overline{access}_r(O, _, P, Q).$$

As easily checked, r is relevant for Q iff Π_r has some answer set containing rel_r. Now, Π_{rel} is defined as the collection of each of the programs Π_r, together with weak constraints of form

$$\Leftarrow \ not\, rel_r\ [1 : m + 1], \tag{5}$$

for every $r \in \Pi_Q$, where m is the maximal priority level of the weak constraints occurring in Π^o_{sel}. Since, for any $r, s \in \Pi_Q$ with $r \neq s$, the programs Π_r and Π_s are defined over disjoint alphabets, and given the inclusion of the weak constraints (5) in Π_{rel}, we obtain that Π_{rel} satisfies the following property:

($*$) for every answer set S of Π_{rel} and every $r \in \Pi_Q$, r is relevant for Q iff $rel_r \in S$.

These answer sets are used as inputs for the program Π_{pref}, which is defined next.

Let $pr(n, m)$ and $pr'(n, m)$ be new binary predicates, where n, m are names. Then,

$$\begin{aligned}
\Pi_{pref} \quad = \quad & \{\ pr(n_r, n_s) \leftarrow rel_r, rel_s \mid r, s \in \Pi_Q \text{ and either } r <_u s \text{ or} \\
& \qquad r, s \text{ satisfy Condition } (O_1) \text{ or } (O_2) \text{ of Definition 6}\ \} \\
\cup \quad & \{\ pr(n_r, n_s) \leftarrow rel_r, rel_s, subpath(o, t_1, p_1, t_2, p_2) \mid r, s \in \Pi_Q, \\
& \qquad access(o, t_1, p_1, q) \in B(r), access(o, t_2, p_2, q) \in B(s)\ \} \\
\cup \quad & \{\ pr'(N_1, N_2) \leftarrow pr(N_1, N_2); \\
& \quad pr'(N_1, N_3) \leftarrow pr'(N_1, N_2), pr(N_2, N_3); \\
& \quad pref(N_1, N_2) \leftarrow pr(N_1, N_2), not\, pr'(N_2, N_1); \\
& \quad pref(N_1, N_3) \leftarrow pref(N_1, N_2), pref(N_2, N_3)\ \}.
\end{aligned}$$

Obviously, Π_{pref} is a locally stratified program. Moreover, in view of Condition ($*$), and since Π_{rel} is independent of Π_{pref} and $\Pi_{\mathcal{S}}$ is independent of Π_{obj}, for every answer set S of $\Pi_{\mathcal{S}} \cup \Pi_{obj} = \Pi_{\mathcal{S}} \cup \Pi_{rel} \cup$

Π_{pref}, we have that (i) $pr(n_r, n_s) \in S$ iff $r \trianglelefteq s$; (ii) $pr'(n_r, n_s) \in S$ iff $r \trianglelefteq^* s$; and (iii) $pref(n_r, n_s) \in S$ iff $r < s$. This shows Condition 1 of the theorem.

As for Condition 2, consider some answer set S of $\Pi_{\mathcal{S}} \cup \Pi_{obj}$. Since $\Pi_{\mathcal{S}}$ is independent of Π_{obj}, S is of form $I \cup J$, where I is an answer set of $\Pi_{\mathcal{S}}$ and J is a set of ground literals disjoint from Lit_{sel}. Hence, $S \cap Lit_{sel} = I$. According to Theorem 1, I is a selection answer set of $(\Pi_{sel}, <_u)$ for Q with respect to $\mathcal{S}$ iff I is a preferred answer set of $(\Pi_{\mathcal{S}}, <)$. But it is easily seen that the latter holds just in case $I \cup J$ is a preferred answer set of $(\Pi_{\mathcal{S}} \cup \Pi_{obj}, <)$. This proves the result. □

We remark that Π_{rel} can be simplified by taking independence of subprograms of $\Pi_{\mathcal{S}}$ and possible uniqueness of answer sets for them into account. For example, if Π_{sd} has a unique answer set, then we may use in each program Π_r simply $\lceil \alpha \rceil_r = \alpha$ for each literal over $\mathcal{A}_{sd}$. In particular, if the program $\Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd}$ has a unique answer set (e.g., if this program is locally stratified), then we may simply take as Π_{rel} the program $\Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd}$ together with all rules $rel_r \leftarrow B^\dagger(r)$, for $r \in \Pi_Q$.

As for the computational complexity of source selection programs, we can derive the following result as a consequence of Theorem 2.

Theorem 3 *Given a query Q and the grounding $\Pi'_{\mathcal{S}}$ of $\Pi_{\mathcal{S}} = \Pi_{qd} \cup R(Q) \cup \Pi_{dom} \cup \Pi_{sd} \cup \Pi_Q$ for a selection base $\mathcal{S} = \langle \Pi_{qd}, \Pi_{dom}, \Pi_{sd}, \Pi_{sel}, <_u \rangle$, where $\mathcal{E}(\mathcal{S}, Q) = (\Pi_Q, <)$, deciding whether $(\Pi_{sel}, <_u)$ has some selection answer set for Q with respect to $\mathcal{S}$ is* NP*-complete, and computing any such selection answer set is complete for* $\mathrm{FP}^{\mathrm{NP}}$*, i.e., computable in polynomial time with an* NP *oracle.*

Proof. Obviously, the groundings Π'_{rel} and Π'_{pref} of the programs Π_{rel} and Π_{pref}, respectively, in the proof of Theorem 2 are constructible in polynomial time from $\Pi'_{\mathcal{S}}$ and Q, and so is the ground program $\Pi' = \Pi'_{\mathcal{S}} \cup \Pi'_{rel} \cup \Pi'_{pref}$. Furthermore, the preferred answer sets of $(\Pi', <)$ correspond to the selection answer sets of $(\Pi_{sel}, <_u)$. Since deciding whether a prioritized logic program (with no weak constraints) has a preferred answer set is NP-complete (cf. [16]), it follows that deciding whether $(\Pi_{sel}, <_u)$ has a selection answer set for Q with respect to $\mathcal{S}$ is in NP. Note that the presence of weak constraints has no influence on the worst-case complexity of deciding the existence of (preferred) answer sets. Moreover, NP-hardness is immediate since the auxiliary rules can form any standard logic program.

From any answer set S of Π', an answer set of $(\Pi_{sel}, <_u)$ is easily computed. Computing such an S is feasible in polynomial time with an NP oracle, sketched as follows. First compute the minimum vector of weak constraint violations, v^*, (i.e., the sum of weights of violated constraints at each level) using the oracle, performing binary search at each level asking whether a violation limit can be obeyed. Then, build atom by atom an answer set S whose violation cost matches v^* using the NP oracle. Overall, this is possible in polynomial time with an NP oracle, hence the problem is in $\mathrm{FP}^{\mathrm{NP}}$.

The hardness for $\mathrm{FP}^{\mathrm{NP}}$ follows from a reduction in [10], which shows how the lexicographic maximum truth assignment to a SAT instance, whose computation is well-known to be complete for $\mathrm{FP}^{\mathrm{NP}}$ [42], can be encoded in the answer set of an ELP with weak constraints. □

Theorem 3 implies that under *data complexity* of source-selection programs, i.e., the selection base $\mathcal{S}$ is fixed while the query Q (represented by the facts $R(Q)$) may vary, the problems in Theorem 3 are in NP resp. $\mathrm{FP}^{\mathrm{NP}}$, since the grounding $\Pi'_{\mathcal{S}}$ of $\Pi_{\mathcal{S}}$ is polynomial in the size of $\mathcal{S}$ and Q in this case. If, moreover, the size of Q is small and bounded by a constant, then the problems are solvable in *polynomial time*, since then the number of rules in $\Pi'_{\mathcal{S}}$ is bounded by some constant as well.

The program $\Pi_{\mathcal{S}} \cup \Pi_{obj}$ in Theorem 2 represents, via preferred answer sets for a dynamic rule preference given by the atoms over $pref$, the selection answer sets of $(\Pi_{sel}, <_u)$ for Q. It can be easily adapted to a

fixed program $\Pi'_{\mathcal{S}}$ such that for any query Q, the dynamic preferred answer sets of $\Pi'_{\mathcal{S}} \cup R(Q)$ represent the selection answer sets of $(\Pi_{sel}, <_u)$ for Q (cf. [16] for more details on dynamic preferences).

One of the desiderata of our approach is that each answer set selects a unique source, for any query Q. The following result states that this property is indeed fulfilled.

Theorem 4 *Let S be a selection answer set of $(\Pi_{sel}, <_u)$ for query Q with respect to $\mathcal{S}$. Then, for any constant q it holds that $|\{s \mid query_source(s, q) \in S\}| \leq 1$.*

Proof. The presence of the structural rule (2) in the evaluation program Π_Q enforces that, whenever S contains two ground atoms of form $query_source(s, q)$ and $query_source(s', q)$, S must be inconsistent, and thus S violates the consistency criterion of answer sets. □

Lastly, the following result concerns the order of application of source-selection rules, stating that source selection is blocked in terms of priorities as desired.

Theorem 5 *Let S be a selection answer set of $(\Pi_{sel}, <_u)$ for query Q with respect to $\mathcal{S}$, and let $r^\Delta \in \Pi_Q$ be the defaultization of some rule r belonging to the grounding of Π^c_{sel} for Q with respect to $\mathcal{S}$.*

Suppose that $B(r)$ is true in S but $H(r) \notin S$. Then, there is some $s \in \Pi_Q$ such that

1. *either s belongs to the grounding of Π^{aux}_{sel} for Q with respect to $\mathcal{S}$ and $H(s) = \neg H(r)$, or s is the defaultization of a rule from the grounding of Π^c_{sel} for Q with respect to $\mathcal{S}$;*
2. *$B(s)$ and $H(s)$ are true in S; and*
3. *either r^Δ and s are incompatible with respect to $<$, or else $r^\Delta < s$ holds.*

Proof. Given that S is a selection answer set of $(\Pi_{sel}, <_u)$ for query Q with respect to $\mathcal{S}$, we have that S is a preferred answer set of the prioritized logic program $(\Pi_Q \cup I, <)$, where I is some selection input of $\mathcal{S}$ for query Q, and, *a fortiori*, that S is an answer set of $\Pi_Q \cup I$. From the latter and $H(r) \notin S$, it follows that $r^\Delta = H(r) \leftarrow B(r), not\ \neg H(r)$ is not a member of $GR(\Pi, S)$. Hence, in view of the assumption that $B(r)$ is true in S, we get that $\neg H(r) \in S$ must hold.

Since S is a preferred answer set of $(\Pi_Q \cup I, <)$, there is some enumeration $\langle r_i \rangle_{j \in J}$ of $GR(\Pi_Q \cup I, S)$ such that Conditions (P_1) – (P_3) hold (cf. Section 3). We take $s = r_\ell$, where ℓ is as follows. Given that $\neg H(r) \in S$, there is some smallest index $j_0 \in J$ such that $r_{j_0} \in GR(\Pi_Q \cup I, S)$ and $H(r_{j_0}) = \neg H(r)$. If r_{j_0} belongs to the grounding of Π^{aux}_{sel} for Q with respect to $\mathcal{S}$, then $\ell = j_0$. Otherwise, by the syntactic form of a source-selection program, r_{j_0} must be an instance of the structural rule (2). By (P_1) – (P_3), the defaultization r' of a rule from the grounding of Π^c_{sel} for Q with respect to $\mathcal{S}$ must exist such that $r' = r_{j'_0} \in GR(\Pi_Q \cup I, S)$ and $j'_0 < j_0$. In this case, $\ell = j'_0$.

We show that r_ℓ satisfies Conditions 1–3. Clearly, Condition 1 is satisfied. Furthermore, Condition 2 is an immediate consequence of the fact that $r_\ell \in GR(\Pi_Q \cup I, S)$. It remains to show that Condition 3 holds.

Towards a contradiction, assume that $r_\ell < r^\Delta$. Since $r_\ell \in GR(\Pi, S)$ and $r^\Delta \notin GR(\Pi, S)$, from Condition (P_3) we get that $B^-(r^\Delta) \cap \{H(r_k) \mid k < \ell\} \neq \emptyset$, as $B^+(r) \subseteq S$ and $B^+(r^\Delta) = B^+(r)$. Now, obviously $\{H(r_k) \mid k < \ell\} \subseteq S$. Moreover, since $B^-(r) \cap S = \emptyset$ and $B^-(r^\Delta) = B^-(r) \cup \{\neg H(r)\}$, we obtain that $\neg H(r) \in \{H(r_k) \mid k < \ell\}$. Hence, there must be some $k_0 < \ell$ such that $r_\ell \in GR(\Pi, S)$ and $H(r_\ell) = \neg H(r)$. But this contradicts the condition that j_0 $(\geq \ell)$ is the smallest index j such that $r_j \in GR(\Pi, S)$ and $H(r_j) = \neg H(r)$. Hence, we either have that r_ℓ and r^Δ are incompatible with respect to $<$, or $r^\Delta < r_\ell$ must hold. □

8 Implementation and Application

We have implemented our source-selection approach on top of the DLV system [21, 43] and its front end plp [15] for prioritized logic programs. The evaluation of source-selection programs proceeds in three steps: First, the set of all selection inputs for a query Q is computed from $R(Q)$, Π_{qd}, Π_{sd}, and Π_{dom}, using DLV (cf. Theorem 2). Second, a call to DLV calculates the priority relation, $<$, from the set of selection inputs and Π_{sel}, and third, the answer sets of $(\Pi_Q, <)$ are determined by employing plp and DLV.

The entire process is implemented as an Eclipse Prolog program and is independent of the actual query language. For XML-QL queries, however, a query parser, written in C++, for generating the low-level representation $R(Q)$ of a query Q has been developed. A query parser for SQL queries is also available. Furthermore, we have "agentized" the source-selection system in the *IMPACT* agent platform [59], such that source-selection agents can be built which may also issue the execution of XML-QL queries on XML-data sources.

8.1 Application: Movie databases

As an application domain, we considered the area of movie databases, and we have built an experimental environment for source selection in this domain, using the prototype implementation of the source selection described above.

8.1.1 Movie sources

We used the *Internet Movie Database* (*IMDb*) [40] as the main source for raw data, as well as the *EachMovie Database* provided by Compaq Computer Corporation [20], to generate a suite of XML movie databases, described below. To this end, (parts of) the large databases were wrapped offline to XML, using a DTD (provided in Appendix A) which we modeled from a set of relevant movie concepts captured by the *Open Directory Project* [52].

RandomMovies (RM): Contains numerous movies randomly wrapped from the IMDb. Besides title and language information (always value "English"), each item comprises, where available, genre, release date, running time, reviews, the two main actors, directors, screenwriters, as well as the corresponding soundtrack (some with composer); persons are recorded by name.

RandomPersons (RP): Contains data of numerous actors, directors, screenwriters, and some composers, randomly wrapped from the IMDb. Besides names, person data comprises birth date, country, and a biography, and may of course again be incomplete.

EachMovie (EM): Wrapped from Compaq's EachMovie database, this source stores English movies plus ratings. For most entries, it has genre information, and for half of them a release date (after 1995). It has no information about actors, directors, soundtracks, etc., however.

Hitchcock (HC): Wrapped from the IMDb, this source stores all movies directed by Alfred Hitchcock, in the format of RandomMovies but with all involved actors listed. For each person, it also has information (if available) about birth date, country, and biography.

KellyGrant (KG): Similar to HC, this source stores all movies in which either Grace Kelly or Cary Grant were actors, as well as all persons involved.

Horror60 (H60): A collection of horror movies from the 1960's, found in the IMDb. Movie and person data are as before, but almost no soundtracks and composers are stored.

For illustration, we give the description of one of the sources (a detailed account of the above databases, using the predicates of Section 5, is given in Appendix E).

Example 11 The description Π_{sd} of source KellyGrant contains the following facts:

$$
\begin{array}{l}
source(s_KellyGrant);\\
up(s_KellyGrant);\\
data_format(s_KellyGrant, xml);\\
update_frequency(s_KellyGrant, low);\\
specialized(s_KellyGrant, \text{“}Kelly\text{”});\\
specialized(s_KellyGrant, \text{“}Grant\text{”});\\
covers(s_KellyGrant, \text{“}Movie\text{”}, low);\\
covers(s_KellyGrant, c, high), \quad \text{for } c \in \{\text{“}ReleaseDate\text{”}, \text{“}Person\text{”}, \text{“}BirthDate\text{”}, \text{“}Actor\text{”}\};\\
covers(s_KellyGrant, fifties, high);\\
covers(s_KellyGrant, sixties, high);\\
\neg relevant(s_KellyGrant, p), \quad \text{for } p \in \{seventies, eighties, ninties, twothousands\}.
\end{array}
$$

Informally, Π_{sd} expresses that KellyGrant is an XML source which is (currently) up and rarely updated. It is specialized on topics *Kelly* and *Grant*, and has high coverage about persons, especially actors, and their birth dates, while low coverage about movies in general. However, it highly covers the release dates of the stored movies, most of which are from the fifties and sixties. Further information about KellyGrant is derived from default rules like the ones given below, stating that English is the default language for all sources:

$$
\begin{array}{rcl}
source_language(S, \text{“}English\text{”}) & \leftarrow & source(S), not\ \neg source_language(S, \text{“}English\text{”});\\
\neg source_language(S, \text{“}English\text{”}) & \leftarrow & source_language(S, L), \neq (L, \text{“}English\text{”}).
\end{array}
$$

8.1.2 Domain knowledge

The domain knowledge, Π_{dom}, consists of an *ontology* part and *background knowledge*. The former contains the relevant concepts of the application domain and their interrelationships. The ontological knowledge might be generated semi-automatically, e.g., from an underlying XML DTD, or it may stem from an external ontological reasoning engine, which is interfaced via the current part of the domain knowledge. Domain models inferred from large-scale ontologies could be automatically accessed by means of a (syntactic) translation of their formal representation into logic programming facts. For example, OIL-based ontologies [26] could be accessed via the FaCT system [25], and ontologies built on Ontolingua [27] could be interfaced by translating corresponding models represented and communicated by KIF statements. Similarly, one could also interface OpenCyc's (`http://www.opencyc.org`) free version of CYC via a translation of its formal representation language CycL. The concrete technical realization of such interfaces is an interesting topic for further research.

For our experiments, we used a simple ontology by representing the concepts of the XML DTD as classes with appropriate attributes in a hierarchy using an *is-a* relation, together with plain ontological reasoning.

Example 12 The concept `Movie` is captured by the fact $class(\text{“}Movie\text{”})$, and its attributes by the facts $has_att(\text{“}Movie\text{”}, att)$, for $att \in \{title, alternativeTitles, genre, releaseDate, runningTime, language, review\}$. Some rules for ontological reasoning are the following standard inheritance rules:

$$\begin{aligned} instance(I, C_2) &\leftarrow instance(I, C_1), is_a(C_1, C_2); \\ has_att(C_1, A) &\leftarrow is_a(C_1, C_2), has_att(C_2, A). \end{aligned}$$

The ontology part of Π_{dom} also serves to define concepts and possible synonyms for the query abstraction in Section 4. Using the XML DTD, we may extract the concepts given by the facts $class0(O)$, for $O \in \{\text{“}MovieDB\text{”}, \text{“}Movie\text{”}, \text{“}Director\text{”}, \text{“}Actor\text{”}, \text{“}Composer\text{”}, \text{“}Person\text{”}, \text{“}Review\text{”}, \text{“}Soundtrack\text{”}, \text{“}Screenwriter\text{”}\}$, and $synonym(\text{“}Personalia\text{”}, \text{“}Person\text{”})$.

The background part of Π_{dom} serves to formalize “common sense” knowledge of the application domain, which is an important source of information for the selection process. This part is usually quite extensive. On the one hand, it has rules capturing typical relationships between ontological concepts and, on the other hand, it comprises “well-known” instances of these concepts. Due to space reasons, we only show a few rules of Π_{dom} here. Note however, that it also implements temporal reasoning over decades, by relating every year since 1920 to its decade.

Example 13 Some (typical) rules from the background knowledge are:

$$\begin{aligned} s_1\colon\ & involved(P, M) \leftarrow directed(P, M); \\ s_2\colon\ & instance(P, \text{“}Director\text{”}) \leftarrow directed(P, M); \\ s_3\colon\ & life_period(P, B, E) \leftarrow instance(P, \text{“}Person\text{”}), has_att(P, birthDate, B1), \\ & \qquad not\ dead(P), calender_year(B1, B), current_year(E); \\ s_4\colon\ & possible_genre(M, G) \leftarrow involved(P, M), default_genre(P, G), not\ defined_genre(M). \end{aligned}$$

Intuitively, rules like s_1 and s_2 infer, from roles *acted*, *directed*, etc., between a person and a movie, that the corresponding person is an actor, director, etc., and that he or she is involved in the movie. Rule s_3 assigns a life period to a person from her birth date, while rule s_4 infers a possible genre for a movie, if an involved person and his or her default genre are known.

The following facts are representations of specific movie-historic incidents (actually, for Grace Kelly and the movie “Arsenic and Old Lace”):

$$\begin{aligned} & instance(perKelly, \text{“}Actor\text{”}); \\ & has_att(perKelly, name, nameKelly); \\ & has_att(perKelly, birthDate, 1929); \\ & has_att(perKelly, dateOfDeath, 1982); \\ & prod_period(perKelly, 1945, 1960); \\ & instance(nameKelly, name); \\ & has_att(nameKelly, firstName, \text{“}Grace\text{”}); \\ & has_att(nameKelly, firstName, \text{“}Patricia\text{”}); \\ & has_att(nameKelly, lastName, \text{“}Kelly\text{”}); \\ & instance(m12, \text{“}Movie\text{”}); \\ & has_att(m12, title, \text{“}Arsenic\ and\ Old\ Lace\text{”}); \\ & has_att(m12, releaseDate, 1944); \\ & acted(perGrant, m12). \end{aligned}$$

8.1.3 Source-selection program

The experimental movie source-selection program fills several pages, and is too complex to be listed and discussed here in detail (see Appendix F). We only give an informal description and highlight the most important aspects.

Among all source-selection rules, default rules have lowest priority, and are used only in the core. They make default suggestions for query sources in case no other core source-selection rule is eligible. Some examples are the following default rules:

$$
\begin{array}{lrcl}
r_1\colon & query_site(S,Q) & \leftarrow & default_path(O,P,Q),\ occurs(O,V),\\
 & & & specialized(S,P);\\
r_{20}\colon & query_site(s_RandomMovies,Q) & \leftarrow & default_class(O,\text{“}Movie\text{”},Q);\\
r_{21}\colon & query_site(s_RandomPersons,Q) & \leftarrow & default_class(O,\text{“}Person\text{”},Q).
\end{array}
$$

The first rule is generic, whilst the others are specific. Informally, r_1 advises to query source S if it is specialized for P, where P is some path of an access in the query that is compared to some value. For example, suppose P is instantiated with *LastName*; if some source is specialized for last names, then it is chosen unless a source-selection rule with higher priority is applicable. Similarly, the specific rules suggest to select RandomMovies or RandomPersons if the query entails an access under object *“Movie”* or *“Person”*, respectively.

Non-default core source-selection rules also appear in either generic or specific form:

$$
\begin{array}{lrcl}
r_{40}\colon & query_site(S,Q) & \leftarrow & source(S),\ query(Q),\ high_coverage(S,Q);\\
r_{41}\colon & query_site(S,Q) & \leftarrow & source(S),\ query(Q),\ special(S,Q);\\
r_{61}\colon & query_site(s_Hitchcock,Q) & \leftarrow & access(O_1,\text{“}Person\text{”},\text{“}LastName\text{”},Q),\\
 & & & selects(O_1,equal,\text{“}Hitchcock\text{”}),\\
 & & & access(O_2,\text{“}Person\text{”},\text{“}FirstName\text{”},Q),\\
 & & & selects(O_2,equal,\text{“}Alfred\text{”}).
\end{array}
$$

The generic rules r_{40} and r_{41} suggest to query any source that highly covers the query or is special for it, respectively. The specific rule r_{61} advises to query the Hitchcock source if a query selects a person named Alfred Hitchcock. Note that $high_coverage$ and $special$ are *auxiliary* predicates, defined by auxiliary rules (see below).

Since no $access$ predicate and no default predicates occur in r_{40} and r_{41}, there is no (direct) structural precedence between them and rule r_{61} as well as r_1, r_{20}, and r_{21}. The following user preferences explicitly establish preferences among them:

$$
\begin{array}{lll}
r_1(_,Q,_,_,_) <_u r_{40}(_,Q); & r_1(_,Q,_,_,_) <_u r_{41}(_,Q); & r_{20}(Q,_) <_u r_{41}(_,Q);\\
r_{21}(Q,_) <_u r_{41}(_,Q); & r_{40}(_,Q) <_u r_{41}(_,Q); & r_{40}(_,Q) <_u r_{61}(Q,_,_);\\
r_{41}(_,Q) <_u r_{61}(Q,_,_). & &
\end{array}
$$

Auxiliary rules are used to define auxiliary predicate as well as to filter irrelevant sources:

$$
\begin{array}{lrcl}
a_1\colon & special(S,Q) & \leftarrow & special_topic(S,Q,T);\\
a_2\colon & special_topic(S,Q,T) & \leftarrow & inferred_topic(Q,T),\ specialized(S,T);\\
a_3\colon & inferred_topic(S,Q,T) & \leftarrow & matchingMovie(Q,M),\ involved(P,M),\\
 & & & has_att(P,name,N),\ has_att(N,lastName,T);
\end{array}
$$

a_4: $\neg query_site(S, Q) \leftarrow irrelevant(S, Q);$

r_{92}: $irrelevant(S, Q) \leftarrow access(O, \text{“}MovieDB\text{”}, \text{“}Movie/ReleaseDate/Date\text{”}, Q),$
$selects(O, equal, V),\ calender_year(V, Y),\ decade(Y, D),$
$\neg relevant(S, D).$

Informally, rule a_1 states that a source is special for a query, if a topic associated with the query exists for which it is special. Rule a_2 suggests that one way to associate a topic to a query is to infer a topic, e.g., by rule a_3. Hence, if the query accesses a movie that is known and T is the last name of a person involved in it, then a source is concluded to be special for that query if it is specialized for T. Rule a_4 states that a source must not be queried if it is irrelevant for a query; in view of rule r_{92}, this is the case if the source is not relevant for the decade of a movie's release date.

Finally, the quantitative part of the source-selection program has weak constraints like:

w_1: $\Leftarrow query_site(S, Q),\ default_class(O, T, Q),$
$constructs(O, C, P),\ covers(S_1, T, high),\ not\ covers(S, T, high)$ [3:1];

w_2: $\Leftarrow query_site(S, Q),\ high_covered_topic(S_1, Q, T),$
$decade_name(T),\ not\ high_covered_topic(S, Q, T)$ [1:1].

Intuitively, w_1 assigns a penalty of 3 per concept T that is asked (resp., constructed) by the query, to any answer set which selects a source that does not highly cover T while a source highly covering T exists. Similarly, w_2 assigns a penalty of 1 per decade that is associated to the query, to any answer set which selects a source which does not highly cover this decade while some other highly covering source exists.

9 Experiments

For testing the approach on the movie application, we used a number of natural user queries listed below. Their formulation in XML-QL is straightforward; for Q_1 and Q_2 it appears in Example 1 and 7, respectively, while for the others it is given in Appendix G.

Q_1 : *Which movies were directed by Alfred Hitchcock?*

Q_2 : *In which movies directed by Josef von Sternberg acted Marlene Dietrich?*

Q_3 : *In which year the movie 'Arsenic and Old Lace' has been released?*

Q_4 : *In which movies directed by Alfred Hitchcock acted Marlene Dietrich?*

Q_5 : *In which film noirs did Marilyn Monroe act?*

Q_6 : *Which movies did Laurel and Hardy act in 1940?*

Q_7 : *Which movies acted by Frank Sinatra have a soundtrack by Elmer Bernstein?*

Q_8 : *When was James Dean born?*

Q_9 : *In which movies did Jane Russell and Marilyn Monroe act together?*

Q_{10} : *In which movies did Cary Grant and Marilyn Monroe act together?*

Q_{11} : *In which comedies did John Wayne act?*

Table 1: Experimental Results for Original Selection Base

Query	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
Candidates	HC	RM, HC, H60, KG	KG	HC	RM, HC, KG	RM, HC, KG	RM, HC, H60, KG	RP	RM, HC, H60, KG
Best	HC	RM	KG	HC	RM	RM	RM	RP	RM

Query	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}
Candidates	KG	RM, HC, KG	RM, HC, H60, KG	H60	RM, HC, KG	RM,HC, KG	RP	RM, HC, H60, KG	EM
Best	KG	RM	RM	H60	RM	RM	RP	RM	EM

Q_{12} : *Which movies acted by Sean Connery have a soundtrack by John Barry?*

Q_{13} : *In which horror movies of the sixties did Leonard Nimoy act?*

Q_{14} : *In which animation movies did Vincent Price act?*

Q_{15} : *In which movies, except crime movies, did James Cagney act between 1930 and 1940?*

Q_{16} : *Where is Giorgio Moroder from?*

Q_{17} : *In which year the movie 'Witness for the Prosecution' has been released?*

Q_{18} : *Which movies from 1999 have been rated above 90 points (out of 100)?*

Source selection for these queries was performed for the suite of movie databases from above, as well as for modifications. Each process took from few seconds up to some tens of seconds, which is due to the large logic program to be processed. However, performance was not an issue; since our prototype implementation and the tools used are unoptimized, there is large potential for runtime improvements.

9.1 Results

The results of source selection for Q_1–Q_{18} using the source descriptions from above are shown in Table 1. Note that, by the semantics of source-selection programs, per answer set and query, a single source is chosen. Thus, query decomposition is not considered here, even if our method of query abstraction would, in principle, allow for it. The entries show the sources which are selected for querying by different answer sets, where "Candidates" and "Best" mean selection with optimization part dropped (i.e., qualitative selection) and enabled (optimization), respectively.

The results can be informally explained as follows. For Q_1, a specific core source-selection rule, r_{61}, which has highest preference, fires and HC is chosen, as expected.

For Q_2, there is some background knowledge about Marlene Dietrich, but no source can be found special for this query, while generic default source-selection rules trigger for all sources. Nonetheless, RP and EM are recognized as irrelevant for Q_2 and discarded: Q_2 asks for resp. ranges over concepts which these sources are not relevant for ("*Movie*" and "*Person*", respectively). The best source of the candidates RM, H60, HC, KG is RM, since it is the only one highly covering the concept asked for, i.e. movies.

For Q_3, since 'Arsenic and Old Lace' is in the background knowledge (cf. above), as well as that Cary Grant acted in it, we would expect KG to be queried since it is specialized for Cary Grant. This is what actually happens. It is not a specific core source-selection rule that triggers, because Cary Grant does not explicitly appear in the query, but Grant is inferred as a query topic from the background knowledge and, thus, the generic core source selection rule suggesting to query KG has highest priority.

Q_4 is a syntactic variant of Q_1; the same specific core source-selection rule, r_{61}, triggers.

Similar as for Q_2, RM is chosen for Q_5–Q_7, but for Q_5 and Q_6, H60 is recognized as irrelevant on different grounds. Q_5 asks for film noirs, and so H60, which contains horror movies, is eliminated by reasoning over genre information. Q_6 asks for movies of 1940, and thus H60, which contains only movies of the 1960's, is excluded by temporal reasoning.

RP is chosen for Q_8, as expected: A specific default source-selection rule triggers for RP, which has precedence over generic default rules that would trigger for other sources.

Queries Q_9 and Q_{10} are similar to query Q_2. While Q_9 yields the same candidates and result as Q_2, this does not hold for Q_10, which explicitly asks for movies acted by Cary Grant, hence, triggering a specific core source-selection rule for KG.

Q_{11}, Q_{12}, Q_{14}, and Q_{15} are variants of queries Q_5–Q_7, which is also reflected in respective results. For another syntactic variant of these queries, query Q_{13}, however, the outcome differs. It explicitly asks for horror movies of the sixties, for which a special source, H60, as well as a specific core source-selection rule suggesting this source exist.

RP is chosen for Q_{16} for the same reasons as it is chosen for Q_8. Q_{17} is a variation of query Q_3, and, again, the movie asked for, namely 'Witness for the Prosecution', is an instance of the background knowledge (cf. Appendix D). This time however, background reasoning does not infer a special source for the query, yielding RM to be the best source among the candidates because of its high coverage of movies.

Finally, EM is the source selected for Q_{18}, since a specific core source-selection rule, concerning queries asking for movie ratings, has highest priority.

9.2 Results with Modified Selection Bases

In a slightly different scenario, RM highly covers composers and cowboy/western movies, too, and a new random movie source *RandomMoviesNew (RMN)* similar to RM, but with less coverage of genres, release dates, composers, and cowboy/western movies, but with high coverage of directors, dramas, and comedies, is introduced. Respective changes to the source descriptions and addition of a specific default source-selection rule for RMN (similar to rule r_{20} for RM in Section 8.1.3) and according user preferences to the source-selection program yield an "extended" selection base, on which the results for Q_1–Q_{18} are shown in Table 2.

The change has no influence on the results of the source-selection process for Q_1, Q_3, Q_4, Q_8, Q_{10}, Q_{13}, Q_{16}, and Q_{18}; this is intuitive, since the change does not affect the suitability of the sources chosen for these queries. For the other queries, the new source RMN is an additional candidate, since the generic default source-selection rule is also applicable to RMN. By an analogous argument as before, RM and RMN are better than the other candidates. For Q_2, Q_9, and Q_{17} RMN is ranked before RM: it highly covers both, dramas and comedies, which are inferred topics for these queries, since in the background knowledge, drama is a default genre for Marlene Dietrich (and Marlene Dietrich acted in 'Witness for the Prosecution') and comedy is a default genre for Marilyn Monroe. RMN is also ranked before RM for Q_{11}, since its high coverage of comedies.

RM is ranked before RMN for Q_7 and Q_{12} since RM highly covers composers, a concept occurring in

Table 2: Experimental Results for "Extended" Selection Base

Query	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
Candidates	HC	RM, HC, H60, KG, RMN	KG	HC	RM, HC, KG, RMN	RM, HC, KG, RMN	RM, HC, H60, KG, RMN	RP	RM, HC, H60, KG, RMN
Best	HC	RMN	KG	HC	RM, RMN	RM, RMN	RM	RP	RMN

Query	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}
Candidates	KG	RM, HC, KG, RMN	RM, HC, H60, KG, RMN	H60	RM, HC, KG RMN	RM, HC, KG RMN	RP	RM, HC, H60, KG, RMN	EM
Best	KG	RMN	RM	H60	RM, RMN	RM, RMN	RP	RMN	EM

Table 3: Experimental Results for "Reduced" Selection Base

Query	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
Candidates	HC	H60, HC, KG	KG	HC	HC, KG,	HC, KG	H60, HC, KG	RP	H60, HC, KG
Best	HC	KG	KG	HC	KG	HC, KG	KG	RP	KG

Query	Q_{10}	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}
Candidates	KG	HC, KG	H60, HC, KG	H60	HC, KG	HC, KG	RP	H60, HC, KG	EM
Best	KG	KG	KG	H60	KG	HC, KG	RP	KG	EM

the queries (which ask for composers Elmer Bernstein and John Barry, respectively). RM and RMN are ranked equal for Q_5, Q_6, Q_{14}, and Q_{15}: they highly cover actors (and are equally relevant for the thirties), but the background knowledge has no information about Laurel and Hardy, or James Cagey; and the facts, that Marilyn Monroe's default genre is drama and that Vincent Price's default genre is horror, have no consequences for Q_5 and Q_{14}, since they explicitly ask for film noirs and animation movies, respectively.

An alternative change yields a "reduced" selection base where RM is down and, thus, cannot be queried. The query results are given in Table 3. The candidate sources remain the same, except that RM is missing; thus, the change has no impact on Q_1, Q_3, Q_4, Q_8, Q_{10}, Q_{13}, Q_{16}, and Q_{18}, as one would expect. For Q_6 and Q_{15}, the optimization part does not impose preference between HC and KG; interestingly, it selects KG as best for Q_2, Q_5, Q_7, Q_9, Q_{11}, Q_{12}, Q_{14}, and Q_{17}. This is because the background knowledge entails information about the productive period of the persons Dietrich, Monroe, Bernstein, Wayne, Connery, Barry, and Price, respectively, occurring in the queries. Thus, the 1950's and 1960's are inferred to be relevant topics of these queries and KG, covering both decades highly, outperforms HC and H60, which highly cover only one of the decades each.

10 Related Work

Selection of data sources is a component in many information integration systems, e.g. [3, 6, 30, 58, 19, 44]; see also [46] and references therein. However, most center around mappings between a global scheme and local schemes, on query rewriting, and on query planning to optimally reconstruct dispersed information. Our work, instead, is concerned with *qualitative* selection from different alternatives, based on rich meta-knowledge and a formal semantics respecting preference and context information involving heuristic defaults, which is not an issue there. Furthermore, no form of query abstraction similar as in our method is considered in these approaches. In the following subsection, we review some of these information integration systems in more detail, and discuss after that other work more closely related to ours.

10.1 Information integration systems

SIMS (Services and Information Management for Decision Systems) [3, 4, 5] is a data integration system which exploits a semantic model of a problem domain to integrate information dispersed over various heterogeneous information sources, which are typically databases or, more generally, knowledge bases. The domain model is formulated in the Loom knowledge representation language [49], and comprises a declarative description of the objects and activities possible in the specific domain. SIMS aims at providing the user a transparent access to the data, without being aware of the underlying heterogenous data sources. It accepts user queries in the form of a description of a class of objects about which information is required. Any such query on the domain model is mapped to a query over the information sources, by translating the concepts of the domain to corresponding concepts in the data models of the information sources; if a direct translation does not exist, a query rewriting is performed, and if needed, multiple databases are accessed in a query plan. SIMS strives for singling out optimal query plans, for which aspects such as costs of accessing the different sources and combining the results returned are taken into account. This is apparently different from the contributions of our work, which is concerned with selecting a single information source out of a set of candidate sources. Furthermore, aspects of incomplete information and nonmonotonic constructs to overcome it were not addressed in SIMS, nor query abstraction.

The Carnot project at MCC [58, 13, 37] was an early effort to provide a logically unifying view of enterprise-wide, distributed and possibly heterogeneous data. The Carnot system has a layered architecture, whose top layer are semantics services which consist of a suite of tools for enterprise modeling, model integration, data cleaning, and knowledge discovery. The Model Integration and Semantics tool (MIST) is used for creating mappings between local schemata and a common ontology expressed in Cyc or in a specific knowledge representation language, which is done once at the time of integration. Besides relational databases, also knowledge-based systems (with an extensional part of facts and an intensional part of rules) may be integrated, and, moreover, play a mediator role between applications and different databases. As an important feature, local database schemas remain untouched, and queries to them are translated to the global schema and back to (other) local schemata for data retrieval. Similar to SIMS, Carnot aims at providing a uniform and consistent view of heterogeneous data. A selection of information sources for query answering based on similar criteria and methods as in our approach is not evident.

InfoSleuth [6, 28, 51], which has its roots in Carnot, is an agent-based system for information discovery and retrieval in a dynamic, open environment, broadening the focus of database research to the challenge of the World Wide Web. It extends the capabilities of Carnot to such an environment, where the identities of the information sources need not be known at the time of generating the mapping. In this approach, agents are the constituents of the systems, whose knowledge and their relationships with one another are described

in an InfoSleuth ontology. Decisions about user-query decompositions are based on a domain ontology, which is selected by the user and describes knowledge about the relationships of the data stored in by the sources that subscribe to the ontology. As for selection of information sources, special broker agents provide upon request information about which resource agents (i.e., information sources behind them) should be accessed for specific information sought. The broker performs a *semantic matchmaking* of the user request to the provider agents's service descriptions (which may be viewed as an advanced yellow pages service), which aims at ruling out by the use of constraints (e.g., on the range of values, of existing attributes, etc.) all sources which will return a nil result. To this end, it must reason over explicitly advertised information about agent capabilities to determine which agent can provide the requested services. The broker translate KIF statements into queries in the LDL++ deductive database language, which are submitted to an LDL++ engine for evaluation. In this way, rule-based matching is facilitated. Our work differs significantly from the work in InfoSleuth, and is in fact to some extent complementary. Indeed, the descriptions of constraints and other semantic criteria for selecting an information source is at a very low level. Even if the LDL++ language, which can emulate non-stratified negation via choice rules, is used for rule-based matchmaking, there is neither special support for dealing with contexts, nor for user preferences, or for optimization constructs as in our approach. Furthermore, it is not evident that InfoSleuth agents are programmed using a declarative language which provides similar constructs for discriminating between different sources compliant with the constraints. Instead, our formalism might be mapped to LDL++ by a suitable transformation and thus provide a plug-in module for realizing semantically richer and refined brokering in InfoSleuth with a well-defined, formal semantics and provable properties.

The Information Manifold [41, 45, 44] is a system for browsing and querying of multiple networked information sources. Its architecture is based on a rich domain model which enables describing the properties of the information sources, such as their addresses, the protocols used to access them, their structure, etc., with a combination of the CLASSIC description logic [7], Horn rules, and integrity constraints. An external information source is viewed as containing extensions of a collection of relations, on which integrity constraints may be imposed, and which are semantically mapped by rules to the relations in the global knowledge base. Topics may be associated with an information source, which allows to classify it along a hierarchy of topics in the domain model, which can be used for deciding retrieval of this source for related queries. Like in SIMS, the user may pose queries in a high-level language on the global schema, which are mapped to queries over the local sources. The Information Manifold focuses on optimizing the execution of a user query, accessing as few informations sources as necessary, where relevance is judged on criteria involving the (static) semantic mapping, and on combining the results. However, no qualitative selection similar to the one in our approach is made, and in particular no user preferences or nonmonotonic rules (including default contexts) can be expressed by constructs in the language.

Infomaster [19, 32] provides integrated access to multiple distributed heterogeneous information sources on the Internet, which gives the illusion of a centralized, homogeneous information system in a virtual schema. The system handles both structural and content translation to resolve differences between multiple data sources and the multiple applications for the collected data, where mappings between the information sources and the global schema are described by rules and constraints. The user may pose queries on the virtual schema, which are first translated to queries on base relations at the information sources and then further rewritten to queries over site relations, which are views on the base relations, by applying logical abduction. The core of Infomaster is a facilitator that dynamically determines an efficient way to answer the user's query using as few sources as necessary and harmonizes the heterogeneities among these sources. However, like in the other information systems above, neither rich meta-data about the quality of information sources is considered, nor preferences or context information is used to heuristically discriminate between

optional choices.

10.2 Other work

More related to our work than the works in the previous subsection is [36], which outlines an interactive tool for information specialists in query design. It relieves them from searching through data source specifications and can suggest sources to determine trade-offs. However, no formal semantics or richer domain theories, capable of handling incomplete and default information, is presented.

Remotely related to our work is [29], which presents a decision-theoretic model for selecting data sources based on retrieval cost and typical IR parameters.

Goto *et al.* [34] consider a problem setting related to ours, where source descriptions include semantic knowledge about the source. In contrast to our work, however, a query is viewed merely as a set of terms, and a source description is a thesaurus automatically constructed from the documents of the source. A further thesaurus, WordNet, is used for the source evaluation algorithm, which is based on the calculation of weighted similarity measures. The main differences to our approach are that the selection method is not declarative and just numeric, that semantic knowledge is limited to a thesaurus, and that no further background knowledge, reasoning, or semantic query analysis is involved.

Semantic analysis of queries has been incorporated to document retrieval in [61]. Starting from conventional IR methods that accept natural language queries against text collections and calculate similarity measures for query keywords, semantic modeling was introduced by trying to detect entity attributes and thematic roles from the query to the effect of a modified similarity computation. While richer ontological knowledge than thesauri is used, source descriptions have no semantic knowledge. Again, the approach is not declarative and numeric in nature, and neither rich domain theories nor automated reasoning is involved.

FAQ FINDER [11] is a natural language question-answering system that uses files of frequently asked questions as its knowledge base. It uses standard IR methods to narrow the search to one FAQ file and to calculate a term-vector metric for the user's question and question/answer pairs. Moreover, it uses a comparison of question type in a taxonomy derived from the query, and a semantic similarity score in question matching. The latter is calculated by passing through the hypernym links, i.e., is-a links, through WordNet.

Recent proposals for Web-based IR build on ontology-based agents which search for, maintain, and mediate relevant information for a user or other agents [48, 57, 12]. A society of software agents is described in [57], where query processing agents assist users in selecting Web-pages. They search for URLs using search engines and ontological WordNet relations for query specialization or generalization to keep the number of located, relevant URLs within given limits. An architecture for ontology-based information gathering agents appears also in [12], but uses special domain search engines and Web-documents as well. Ontologies are represented in a usual object-oriented language, and queries as partial instances of ontological concepts.

11 Conclusion

In this paper, we have presented a knowledge-based approach to information source selection, which uses meta-knowledge about the quality of the sources for determining a "best" information source to answer a given query, which is posed in a formal query language (as considered here, XML-QL). We have described a rule-based language for expressing source-selection policies in a fully declarative way, which supports reasoning tasks that involve different components such as background and ontological knowledge, source

descriptions, and query constituents. Furthermore, the language provides a number of features which have proven valuable in the context of knowledge representation, namely the capability of dealing with incomplete information, default rules, and preference information.

To this end, we have developed a novel method for automated query analysis and abstraction at a generic level, in which interesting information is distilled from a given query expression in a formal query language (e.g., XML-QL or SQL [56]), as well as an approach to preference handling in source selection, which combines implicit rule priorities, given by the context of rule application, and explicit user preferences. As pointed out previously, context-based rule application is different from inheritance based reasoning; to our knowledge, no similar approach to default context rules has been considered before. We have presented a formal model-theoretic semantics of our approach, which is based on the answer-set semantics of extended logic programs. Furthermore, we have analyzed semantical and computational properties of the approach, where we have shown that source selection programs possess desirable properties which intuitively should be satisfied. We emphasize that for other, related approaches no similar results are evident, since lacking formal semantics makes them less accessible to reason about their behavior.

The results that we have obtained in the implementation of the experimental movie application are encouraging, and suggest several directions for further work.

The current setting of single source selection can be generalized to ranked source selection, in which the k given best sources are computed, for a given k. This can easily be accomplished by computing ranked models in the underlying `DLV` system. Another issue concerns the supply of rich background and common-sense knowledge. The coupling with available ontology and common-sense engines via suitable interfaces (cf. Section 8.1.2) is suggestive for this purpose; we remark that very recent efforts aim at mapping description logics behind ontology languages to logic programs, which helps bridging this gap [35, 50, 60].

Another direction is the application of our results in the context of information integration and query systems. They might be valuable for enriching semantic brokering in open agent-based such systems, but as well for more traditional closed systems in which information sources must be manually registered. In particular, the EU-funded INFOMIX project [38] on advanced information integration, employing extended logic programming tools, is a natural candidate for incorporating a heuristic source-selection component. A further issue concerns adaptive source selection which is customized, e.g., by user profiles. This subject is important for realizing personalized information systems in a dynamic environment, which will to a great deal involve user preferences and reasoning with incomplete information and defaults, as well as dynamic updates of source descriptions. Investigating this subject is part of the agenda of the REWERSE (*Resoning on the Web with Rules and Semantics*) Network of Excellence funded by the EU, which is scheduled to start in April 2004.

References

[1] J. Alferes, L. Pereira, H. Przymusinska, and T. Przymusinski. LUPS - A Language for Updating Logic Programs. *Artificial Intelligence*, 138(1–2):87–116, 2002.

[2] K. Apt, H. Blair, and A. Walker. Towards a Theory of Declarative Knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Inc., Washington DC, 1988.

[3] Y. Arens, C. Chee, C. Hsu, and C. Knoblock. Retrieving and Integrating Data from Multiple Information Sources. *Int. J. of Cooperative Information Systems*, 2(2):127–158, 1993.

[4] Y. Arens and C. Knoblock. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems. In *Proc. 1st Int. Conf. on Information and Knowledge Managements*, 1992.

[5] Y. Arens, C. Knoblock, and W. Shen. Query Reformulation for Dynamic Information Integration. *J. of Intelligent Information Systems*, 6(2–3):99–130, 1996.

[6] R. Bayardo, B. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Semantic Integration of Information in Open and Dynamic Environments (Experience Paper). In *SIGMOD Conf. 1997*, pages 195–206, 1997.

[7] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. CLASSIC: A Structural Data Model for Objects. In J. Clifford, B. G. Lindsay, and D. Maier, editors, *Proc. 1989 ACM SIGMOD International Conference on Management of Data*, pages 58–67. ACM Press, 1989.

[8] G. Brewka and T. Eiter. Preferred Answer Sets for Extended Logic Programs. *Artificial Intelligence*, 109(1–2):297–356, 1999.

[9] F. Buccafurri, N. Leone, and P. Rullo. Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *J. of Logic Programming*, 27(1):5–43, 1996.

[10] F. Buccafurri, N. Leone, and P. Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Trans. on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[11] R. Burke, K. Hammond, and J. Kozlovsky. Knowledge-Based Information Retrieval from Semi-Structured Text. In *Working Notes of AAAI-95 Fall Symposium, Series on AI Applications in Knowlege Navigation and Retrieval, Cambridge, MA*, pages 19–24, 1995.

[12] Y.-J. Chen and V.-W. Soo. Ontology-Based Information Gathering Agents. *Lecture Notes in Artificial Intellingence (LNAI)*, 2198:423–427, 2001.

[13] C. Collet, M. Huhns, and W.-M. Shen. Resoure Integration Using a Large Knowledge Base in Carnot. *IEEE Computer*, 24(12):55–62, 1991.

[14] K. Decker, K. Sycara, and M. Williamson. Middle-Agents for the Internet. In *Proc. 15th Int. Joint Conf. on Artificial Intelligence (IJCAI'97)*, volume 1, pages 578–583. Morgan Kaufmann, 1997.

[15] J. Delgrande, T. Schaub, and H. Tompits. plp: A Generic Compiler for Ordered Logic Programs. In T. Eiter, W. Faber, and M. Truszczyński, editors, *Proc. 6th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'01)*, pages 411–415. Springer, 2001.

[16] J. P. Delgrande, T. Schaub, and H. Tompits. A Framework for Compiling Preferences in Logic Programs. *Journal of the Theory and Practice of Logic Programming*, 3(2):129–187, 2003.

[17] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. In *Proc. 8th International World Wide Web Conference (WWW8)*, 1999. Computer Networks 31(11-16): 1155-1169. See also `http://www.w3.org/TR/NOTE-xml-ql/`.

[18] Y. Dimopoulos and A. Kakas. Information Integration and Computational Logic. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. `http://www.compulog.org/net/Forum/Supportdocs.html`.

[19] O. Duschka and M. Genesereth. Infomaster: An Information Integration Tool. In *Proceedings of the International Workshop on Intelligent Information Integration, 21st German Annual Conference on Artificial Intelligence, KI-97*, 1997. citeseer.nj.nec.com/duschka97infomaster.html.

[20] EachMovie Database. http://www.research.compaq.com/SRC/eachmovie/.

[21] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[22] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On Properties of Update Sequences Based on Causal Rejection. *Journal of the Theory and Practice of Logic Programming*, 2(6):721–777, 2002.

[23] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. Using Methods of Declarative Logic Programming for Intelligent Information Agents. *Journal of the Theory and Practice of Logic Programming*, 2(6):645–719, 2002.

[24] T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, Sept. 1997.

[25] Fast Classification of Terminologies (FaCT). http://www.cs.man.ac.uk/~horrocks/FaCT.

[26] D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, 2001. http://www.computer.org/intelligent/ex2001/x2toc.htm.

[27] R. Fikes, A. Farquhar, and J. Rice. Tools for Assembling Modular Ontologies in Ontolingua, March 1997. http://www.ksl.stanford.edu/software/ontolingua/project-papers.html.

[28] J. Fowler, B. Perry, M. H. Nodine, and B. Bargmeyer. Agent-Based Semantic Interoperability in InfoSleuth. *SIGMOD Record*, 28(1):60–67, 1999.

[29] N. Fuhr. A Decision-Theoretic Approach to Database Selection in Networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.

[30] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. of Intelligent Information Systems*, 8(2):117–132, 1997.

[31] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3–4):365–386, 1991.

[32] M. Genesereth, A. Keller, and O. Duschka. Infomaster: An Information Integration System. In *Proc. ACM SIGMOD Conference*, volume 26(2) of *SIGMOD Record*, pages 539–542, New York, 1997. ACM Press.

[33] Google. http://www.google.com.

[34] S. Goto, T. Ozono, and T. Shintani. A Method for Information Source Selection using Thesaurus for Distributed Information Retrieval. In *Proceedings of the Pacific Asian Conference on Intelligent Systems 2001 (PAIS 2001)*, pages 272–277, November 2001.

[35] B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logics. In *Proceedings WWW-2003*, 2003.

[36] S. B. Huffman and D. Steier. A Navigation Assistant for Data Source Selection and Integration. In *Working Notes of AAAI-95 Fall Symposium Series on AI Applications in Knowledge Navigation and Retrieval, Cambridge, MA*, pages 72–77. AAAI Press, 1995.

[37] M. Huhns and M. Singh. The Semantic Integration of Information Models. In *AAAI Workshop on Cooperation among Heterogeneous Intelligent Agents*, 1992.

[38] Infomix homepage. `http://sv.mat.unical.it/infomix/`.

[39] K. Inoue and C. Sakama. Prioritized Logic Programming and Its Applications to Commonsense Reasoning. *Artificial Intelligence*, 123(1–2):185–222, 2000.

[40] Internet Movie Database (IMDb). `http://www.imdb.org`.

[41] T. Kirk, A. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings AAAI 2001 Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*. AAAI Press, 1995. `http://citeseer.nj.nec.com/16242.html`.

[42] M. Krentel. The Complexity of Optimization Problems. *J. Comput. Syst. Sci.*, 36:490–509, 1988.

[43] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, C. Koch, C. Mateis, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. Technical Report INFSYS RR-1843-02-14, Institut für Informationssysteme, Technische Universität Wien, A-1040 Vienna, Austria, October 2002. Submitted for publication.

[44] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *Proc. 22th Int. Conf. on Very Large Data Bases* (*VLDB'96*), pages 251–262. Morgan Kaufmann, 1996.

[45] A. Levy, D. Srivastava, and T. Kirk. Data Model and Query Evaluation in Global Information Systems. *J. of Intelligent Information Systems*, 5(2):121–143, 1995.

[46] A. Levy and D. Weld. Intelligent Internet Systems. *Artificial Intelligence*, 118(1–2):1–14, 2000.

[47] V. Lifschitz and H. Turner. Splitting a Logic Program. In *Proceedings ICLP-94*, pages 23–38, Santa Margherita Ligure, Italy, June 1994. MIT-Press.

[48] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based Web Agents. In W.L. Johnson, editor, *In Proceedings of the First International Conference on Autonomous Agents (AutonomousAgents97), New York, Association for Computing Machinery*, pages 59–66, 1997.

[49] R. MacGregor and R. Bates. The LOOM knowledge representation language. Technical Report RS-87-188, Information Sciences Institute, Univ. of Southern California, 1987. Project website `http://www.isi.edu/isd/LOOM/`.

[50] B. Motik, R. Volz, and A. Maedche. Optimizing query answering in description logics using disjunctive deductive databases. In F. Bry, C. Lutz, U. Sattler, and M. Schoop, editors, *Proceedings of the 10th International Workshop on Knowledge Representation meets Databases (KRDB 2003), Hamburg, Germany, September 15-16, 2003*, volume 79 of *CEUR Workshop Proceedings*. Technical University of Aachen (RWTH), 2003. Online `http://CEUR-WS.org/Vol79/`.

[51] M. Nodine, A. Ngu, A. Cassandra, and W. Bohrer. Scalable semantic brokering over dynamic heterogeneous data sources in InfoSleuth. *IEEE Trans. on Knowledge and Data Engineering*, 15(5):1082–1098, 2003.

[52] Open Directory Project. `http://dmoz.org`.

[53] A. Provetti and T. C. Son, editors. *Proceedings AAAI 2001 Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning, Stanford, CA (Workshop Technical Report SS-01-01)*. AAAI Press, March 2001.

[54] T. C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufman, Washington DC, 1988.

[55] F. Sadri and F. Toni. Computational Logic and Multi-Agent Systems: a Roadmap. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net*, 2000. Avaliable from `http://www.compulog.org/net/Forum/Supportdocs.html`.

[56] R. Schindlauer. Representation of SQL queries for declarative query analysis. Master's thesis, Institut für Informationssysteme, TU Wien, Austria, Sept. 2002.

[57] K. M. Sim and P. T. Wong. Web-Based Information Retrieval Using Agent and Ontology. *Lecture Notes in Artificial Intellingence (LNAI)*, 2198:384–388, 2001.

[58] M. Singh, P. Cannata, M. Huhns, N. Jacobs, T. Ksiezyk, K. Ong, A. Sheth, C. Tomlinson, and D. Woelk. The Carnot Heterogeneous Database Project: Implemented Applications. *Distributed and Parallel Databases*, 5(2):207–225, 1997.

[59] V. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous Agent Systems: Theory and Implementation*. MIT Press, 2000.

[60] T. Swift. Deduction in ontologies via ASP. In *Proceedings LPNMR-7*. Springer, 2004. To appear.

[61] E. B. Wendlandt and J. R. Driscoll. Incorporating a Semantic Analysis into a Document Retrieval Strategy. In Abraham Bookstein and Yves Chiaramella and Gerard Salton and Vijay V. Raghavan, editor, *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Chicago, Illinois, USA, October 13-16, 1991 (Special Issue of the SIGIR Forum)*, pages 270–279. ACM, 1991.

[62] G. Wiederhold. Intelligent Integration of Information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 434–437, Washington, DC, 1993.

A Appendix: XML DTD for the Movie Databases

```
<!ELEMENT MovieDB (Movie|Actor|Director|Screenwriter|Composer|Person|Award|
                   Filmfestival)*>
<!ELEMENT Movie (Title,AlternativeTitle*,ReleaseDate?,RunningTime?,Culture?,
                 LeadingRole*,Role*,Actor*,Director*,Screenwriter*,Soundtrack*,
                 Review*,Award*)>
<!ATTLIST Movie
          Genre (Action|Animation|Classic|Comedy| CowboyWestern|CultMovie|
                 Documentary|Experimental|FilmNoir|Horror|Romance|SciFiFantasy|
                 Series|Silent|Travel|Other)
          #IMPLIED Language CDATA "English">
<!ELEMENT Person (FirstName*,LastName,BirthDate?,Country?,Biography?)>
<!ATTLIST Person ID ID #REQUIRED Gender (male|female) #IMPLIED>
<!ELEMENT Award (AwardTitle,Date,AwardType?,AwardCategory?)>
<!ELEMENT Character (#PCDATA)>
<!ELEMENT Filmfestival (#PCDATA)>
<!ELEMENT Actor (Award*)>
<!ATTLIST Actor Personalia IDREF #REQUIRED>
<!ELEMENT Director (Award*)>
<!ATTLIST Director Personalia IDREF #REQUIRED>
<!ELEMENT Screenwriter (Award*)>
<!ATTLIST Screenwriter Personalia IDREF #REQUIRED>
<!ELEMENT Composer (Award*)>
<!ATTLIST Composer Personalia IDREF #REQUIRED>
<!ELEMENT Soundtrack (Title,Composer*,Award*)>
<!ELEMENT Biography (#PCDATA)>
<!ELEMENT AlternativeTitle (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT LastName (#PCDATA)>
<!ELEMENT BirthDate (Date)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT AwardCategory (#PCDATA)>
<!ELEMENT AwardType (#PCDATA)>
<!ELEMENT AwardTitle (#PCDATA)>
<!ELEMENT ReleaseDate (Date)>
<!ELEMENT RunningTime (#PCDATA)>
<!ELEMENT LeadingRole (Character,Award*)>
<!ATTLIST LeadingRole Actor IDREF #REQUIRED>
<!ELEMENT Role (Character,Award*)>
<!ATTLIST Role Actor IDREF #REQUIRED>
<!ELEMENT Review (ReviewText,Rating?)>
<!ELEMENT ReviewText (#PCDATA)>
<!ELEMENT Rating (#PCDATA)>
<!ELEMENT Culture (#PCDATA)>
```

B Appendix: Low-level Query-Representation Examples

The low-level representation $R(Q)$ of the query in Example 1, comprises the following set of facts:

$$
\begin{aligned}
\{\ & db_name(\text{“}MovieDB\text{”}), query_cand(q_1), sub_query(q_2, q_1), source(\text{“}MovieDB\text{”}, q_2),\\
& whereRef(o_1, \text{“}Title\text{”}, \text{“ ”}, q_2), whereRef(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_2),\\
& whereRef(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_2),\\
& subpath(o_1, \text{“}Title\text{”}, \text{“ ”}, \text{“}Movie\text{”}, \text{“}Title\text{”}),\\
& subpath(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}),\\
& whereRef(o_2, \text{“}FirstName\text{”}, \text{“ ”}, q_2),\\
& whereRef(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, q_2),\\
& whereRef(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_2),\\
& whereRef(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}, q_2),\\
& whereRef(o_2, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}, q_2),\\
& subpath(o_2, \text{“}FirstName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}),\\
& subpath(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}),\\
& subpath(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”},\\
& \qquad \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}),\\
& subpath(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”},\\
& \qquad \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}),\\
& whereRef(o_3, \text{“}LastName\text{”}, \text{“ ”}, q_2),\\
& whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2),\\
& whereRef(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_2),\\
& whereRef(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_2),\\
& whereRef(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_2),\\
& subpath(o_3, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}LastName\text{”}),\\
& subpath(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}),\\
& subpath(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”},\\
& \qquad \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}),\\
& subpath(o_3, \text{”}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”},\\
& \qquad \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}),\\
& whereCmp(o_2, equal, \text{“}Alfred\text{”}), whereCmp(o_3, equal, \text{“}Hitchcock\text{”}),\\
& consRef(o_1, \text{“}Movie\text{”}, \text{“ ”}), consRef(o_1, \text{“}MovieList\text{”}, \text{“}Movie\text{”})\}
\end{aligned}
$$

The low-level representation $R(Q)$ of the query in Example 7, comprises the following set of facts:

$$
\begin{aligned}
\{\ & db_name(\text{“}MovieDB\text{”}), query_cand(q_1), sub_query(q_2, q_1), source(\text{“}MovieDB\text{”}, q_2),\\
& whereRef(o_1, \text{“}Title\text{”}, \text{“ ”}, q_2), whereRef(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_2),\\
& whereRef(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_2),\\
& subpath(o_1, \text{“}Title\text{”}, \text{“ ”}, \text{“}Movie\text{”}, \text{“}Title\text{”}),\\
& subpath(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}),\\
& whereRef(o_2, \text{“}FirstName\text{”}, \text{“ ”}, q_2),\\
& whereRef(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, q_2),
\end{aligned}
$$

$whereRef(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_2),$
$whereRef(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}, q_2),$
$whereRef(o_2, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}, q_2),$
$subpath(o_2, \text{“}FirstName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}),$
$subpath(o_2, \text{“}Personalia\text{”}, \text{“}FirstName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}),$
$subpath(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”},$
$\text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}),$
$subpath(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”},$
$\text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}),$
$whereRef(o_3, \text{“}LastName\text{”}, \text{“ ”}, q_2),$
$whereRef(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, q_2),$
$whereRef(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_2),$
$whereRef(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_2),$
$whereRef(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_2),$
$subpath(o_3, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Personalia\text{”}, \text{“}LastName\text{”}),$
$subpath(o_3, \text{“}Personalia\text{”}, \text{“}LastName\text{”}, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}),$
$subpath(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”},$
$\text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}),$
$subpath(o_3, \text{”}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”},$
$\text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}),$
$whereRef(o_4, \text{“}MovieDB\text{”}, \text{“}Movie/Actor/Personalia\text{”}, q_2),$
$whereRef(o_4, \text{“}Movie\text{”}, \text{“}Actor/Personalia\text{”}, q_2),$
$whereRef(o_4, \text{“}Actor\text{”}, \text{“}Personalia\text{”}, q_2),$
$whereRef(o_4, \text{“}Personalia\text{”}, \text{“ ”}, q_2),$
$subpath(o_4, \text{“}Personalia\text{”}, \text{“ ”}, \text{“}Actor\text{”}, \text{“}Personalia\text{”}),$
$subpath(o_4, \text{“}Actor\text{”}, \text{“}Personalia\text{”}, \text{“}Movie\text{”}, \text{“}Actor/Personalia\text{”}),$
$subpath(o_4, \text{”}Movie\text{”}, \text{“}Actor/Personalia\text{”}, \text{“}MovieDB\text{”}, \text{“}Movie/Actor/Personalia\text{”}),$
$whereRef(o_5, \text{“}MovieDB\text{”}, \text{“}Person/ID\text{”}, q_2),$
$whereRef(o_5, \text{“}Person\text{”}, \text{“}ID\text{”}, q_2),$
$whereRef(o_5, \text{“}ID\text{”}, \text{“ ”}, q_2),$
$subpath(o_5, \text{“}ID\text{”}, \text{“ ”}, \text{“}Person\text{”}, \text{“}ID\text{”}),$
$subpath(o_5, \text{“}Person\text{”}, \text{“}ID\text{”}, \text{“}MovieDB\text{”}, \text{“}Person/ID\text{”}),$
$whereRef(o_6, \text{“}MovieDB\text{”}, \text{“}Person/FirstName\text{”}, q_2),$
$whereRef(o_6, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_2),$
$whereRef(o_6, \text{“}FirstName\text{”}, \text{“ ”}, q_2),$
$subpath(o_6, \text{“}FirstName\text{”}, \text{“ ”}, \text{“}Person\text{”}, \text{“}FirstName\text{”}),$
$subpath(o_6, \text{“}Person\text{”}, \text{“}FirstName\text{”}, \text{“}MovieDB\text{”}, \text{“}Person/FirstName\text{”}),$
$whereRef(o_7, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}, q_2),$
$whereRef(o_7, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_2),$
$whereRef(o_7, \text{“}LastName\text{”}, \text{“ ”}, q_2),$
$subpath(o_7, \text{“}LastName\text{”}, \text{“ ”}, \text{“}Person\text{”}, \text{“}LastName\text{”}),$
$subpath(o_6, \text{“}Person\text{”}, \text{“}LastName\text{”}, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}),$

$whereCmp(o_2, equal, \text{“}Josef\text{”}), whereCmp(o_3, equal, \text{“}Sternberg\text{”}),$
$whereCmp(o_6, equal, \text{“}Marlene\text{”}), whereCmp(o_3, equal, \text{“}Dietrich\text{”}),$
$consRef(o_7, \text{“}Movie\text{”}, \text{“ ”}), consRef(o_1, \text{“}MovieList\text{”}, \text{“}Movie\text{”}), whereRefCmp(o_4, equal, o_5)\}$

C Appendix: High-level Query-Description Examples

The complete high-level description (except auxiliary predicates and the completion of the subpath predicates) of the query in Example 1 is given by the following set of facts:

$\{\ query(q_1),$
$access(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_1),$
$access(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_1),$
$access(o_2, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1),$
$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1),$
$occurs(o_2, \text{“}Alfred\text{”}), occurs(o_3, \text{“}Hitchcock\text{”}),$
$selects(o_2, equal, \text{“}Alfred\text{”}), selects(o_3, equal, \text{“}Hitchcock\text{”}),$
$constructs(o_1, \text{“}Movie\text{”}, \text{“ ”})\ \}$

The complete high-level description of the query in Example 7 is given by the following set of facts (auxiliary predicates and the completion of the subpath predicates again not included):

$\{\ query(q_1),$
$access(o_1, \text{“}MovieDB\text{”}, \text{“}Movie/Title\text{”}, q_1),$
$access(o_1, \text{“}Movie\text{”}, \text{“}Title\text{”}, q_1),$
$access(o_2, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Movie\text{”}, \text{“}Director/Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Director\text{”}, \text{“}Personalia/FirstName\text{”}, q_1),$
$access(o_2, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1),$
$access(o_3, \text{“}MovieDB\text{”}, \text{“}Movie/Director/Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Movie\text{”}, \text{“}Director/Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Director\text{”}, \text{“}Personalia/LastName\text{”}, q_1),$
$access(o_3, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1),$
$access(o_4, \text{“}MovieDB\text{”}, \text{“}Movie/Actor/Personalia\text{”}, q_1),$
$access(o_4, \text{“}Movie\text{”}, \text{“}Actor/Personalia\text{”}, q_1),$
$access(o_4, \text{“}Actor\text{”}, \text{“}Personalia\text{”}, q_1),$
$access(o_4, \text{“}Person\text{”}, \text{“ ”}, q_1),$
$access(o_5, \text{“}MovieDB\text{”}, \text{“}Person/ID\text{”}, q_1),$

$access(o_5, \text{“}Person\text{”}, \text{“}ID\text{”}, q_1)$,
$access(o_6, \text{“}MovieDB\text{”}, \text{“}Person/FirstName\text{”}, q_1)$,
$access(o_6, \text{“}Person\text{”}, \text{“}FirstName\text{”}, q_1)$,
$access(o_7, \text{“}MovieDB\text{”}, \text{“}Person/LastName\text{”}, q_1)$,
$access(o_7, \text{“}Person\text{”}, \text{“}LastName\text{”}, q_1)$,
$occurs(o_2, \text{“}Josef\text{”}), occurs(o_3, \text{“}Sternberg\text{”})$,
$occurs(o_6, \text{“}Marlene\text{”}), occurs(o_7, \text{“}Dietrich\text{”})$,
$selects(o_2, equal, \text{“}Josef\text{”}), selects(o_3, equal, \text{“}Sternberg\text{”})$,
$selects(o_6, equal, \text{“}Marlene\text{”}), selects(o_7, equal, \text{“}Dietrich\text{”})$,
$constructs(o_1, \text{“}Movie\text{”}, \text{“ ”}), joins(o_4, o_5, equal)$ }

D Appendix: Experimental Movie Doamin Knowledge

In our experiments the following ELP served as the domain theory, Π_{dom}, for the movie domain:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% input for query abstraction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

class0(s_MovieDB).
class0(s_Movie).
class0(s_Director).
class0(s_Actor).
class0(s_Screenwriter).
class0(s_Composer).
class0(s_Person).
class0(s_Soundtrack).
class0(s_Review).
synonym(s_Personalia,s_Person).


%%%%%%%%%%%%%%%%%
% domain ontology
%%%%%%%%%%%%%%%%%

% classes

class(s_Movie).
class(s_Person).
class(name).
class(s_Actor).
class(s_Director).
class(s_Screenwriter).
class(s_Composer).
class(s_Soundtrack).
class(s_Review).

% subclasses

is_a(s_Actor,s_Person).
```

```
is_a(s_Director,s_Person).
is_a(s_Screenwriter,s_Person).
is_a(s_Composer,s_Person).

% attributes

has_att(s_Movie,title).
has_att(s_Movie,alternativeTitles).
has_att(s_Movie,genre).
has_att(s_Movie,language).
has_att(s_Movie,runningTime).
has_att(s_Movie,releaseDate).
has_att(s_Movie,review).

has_att(s_Person,id).
has_att(s_Person,gender).
has_att(s_Person,name).
has_att(s_Person,birthDate).
has_att(s_Person,dateOfDeath).
has_att(s_Person,country).
has_att(s_Person,biography).

has_att(name,firstName).
has_att(name,lastName).

has_att(nameList,name).

has_att(s_Actor,personalia).
has_att(s_Director,personalia).
has_att(s_Screenwriter,personalia).
has_att(s_Composer,personalia).

has_att(s_Soundtrack,title).
has_att(s_Soundtrack,composer).

has_att(s_Review,text).
has_att(s_Review,rating).

% inheritance

instance(I,C2) :- instance(I,C1), is_a(C1,C2).
has_att(C2,A) :- is_a(C1,C2), has_att(C1,A).


%%%%%%%%%%%%%%%%%%%%%%%
% background knowledge
%%%%%%%%%%%%%%%%%%%%%%%

% common sense

#maxint=2020.

genre("Action").
genre("Animation").
genre("Classic").
```

```
genre("Comedy").
genre("CowboyWestern").
genre("CultMovie").
genre("Documentary").
genre("Experimental").
genre("FilmNoir").
genre("Horror").
genre("Romance").
genre("SciFiFantasy").
genre("Series").
genre("Silent").
genre("Travel").
genre("Other").
genre(G) :- att_val(M,genre,G).


calender_year(X,Y) :- <(X,100), Y=X+1900.
%calender_year(X,X) :- #int(X), >=(X,100).
calender_year(X,X) :- #int(X), >=(X,1850).
year(Y) :- calender_year(X,Y).
current_year(2002).

decade_name(twenties).
decade_name(thirties).
decade_name(fourties).
decade_name(fifties).
decade_name(sixties).
decade_name(seventies).
decade_name(eighties).
decade_name(nineties).
decade_name(twothousands).
decade(Y,twenties) :- calender_year(X,Y), >=(Y,1920), <(Y,1930).
decade(Y,thirties) :- calender_year(X,Y), >=(Y,1930), <(Y,1940).
decade(Y,fourties) :- calender_year(X,Y), >=(Y,1940), <(Y,1950).
decade(Y,fifties) :- calender_year(X,Y), >=(Y,1950), <(Y,1960).
decade(Y,sixties) :- calender_year(X,Y), >=(Y,1960), <(Y,1970).
decade(Y,seventies) :- calender_year(X,Y), >=(Y,1970), <(Y,1980).
decade(Y,eighties) :- calender_year(X,Y), >=(Y,1980), <(Y,1990).
decade(Y,nineties) :- calender_year(X,Y), >=(Y,1990), <(Y,2000).
decade(Y,twothousands) :- calender_year(X,Y), >=(Y,2000), <(Y,2010).
decade_before(D1,D2) :- decade(Y1,D1), decade(Y2,D2), !=(D1,D2), <(Y1,Y2).
decade_after(D1,D2) :- decade(Y1,D1), decade(Y2,D2), !=(D1,D2), >(Y1,Y2).
decade_between(D1,D2,D3) :- decade(Y1,D1), decade(Y2,D2), decade(Y3,D3),
                            <(Y2,Y3), <=(Y2,Y1), >=(Y3,Y1).

rating_val(X) :- #int(X), <=(X,100).

involved(P,M) :- directed(P,M).
involved(P,M) :- acted(P,M).
involved(P,M) :- screen_written(P,M).
involved(P,M) :- sound_composed(P,M).

sound_composed(P,M) :- sound_track(S,M), composed(P,S).

instance(P,s_Director) :- directed(P,M).
```

```
instance(P,s_Actor) :- acted(P,M).
instance(P,s_Screenwriter) :- screen_written(P,M).
instance(P,s_Composer) :- sound_composed(P,M).

life_period(P,B,E) :- instance(P,s_Person), att_val(P,birthDate,B1),
                      not dead(P), calender_year(B1,B),
                      current_year(E).
life_period(P,B,E) :- instance(P,s_Person), att_val(P,birthDate,B1),
                      att_val(P,dateOfDeath,E1), calender_year(B1,B),
                      calender_year(E1,E).
dead(P) :- att_val(P,dateOfDeath,D).

defined_releaseDate(M) :- att_val(M,releaseDate,Y).
release_bound(M,X,Y) :- directed(M,D), prod_period(D,B,E), calender_year(B,X),
                        calender_year(E,Y), not defined_releaseDate(M).
release_bound(M,X,Y) :- acted(M,A), prod_period(A,B,E), calender_year(B,X),
                        calender_year(E,Y), not defined_releaseDate(M).
defined_genre(M) :-  att_val(M,genre,G).
possible_genre(M,G) :-  involved(P,M), default_genre(P,G), not defined_genre(M).
possible_genre(P,G) :-  involved(P,M), possible_genre(M,G), not defined_genre(M).
possible_genre(P,G) :-  default_genre(P,G).

att_val(M,genre,"Silent") :- instance(M,s_Movie), att_val(M,releaseDate,D),
                             calender_year(D,R), <(R,1920).
att_val(M,genre,"Silent") :- instance(M,s_Movie), att_val(M,releaseDate,D),
                             calender_year(D,R), <(R,1930),
                             not -att_val(M,genre,"Silent").
-att_val(M,genre,"Silent") :- att_val(M,genre,G), !=(G,"Silent").

:- prod_period(P,X,Y), calender_year(X,B), calender_year(Y,E), >(B,E).
:- prod_period(P,X,Y), att_val(P,birthDate,B), calender_year(X,B1),
   calender_year(B,B2), <(B1,B2).
:- prod_period(P,X,Y), att_val(P,dateOfDeath,D), calender_year(Y,E1),
   calender_year(D,E2), >(E1,E2).


% Persons:
%%%%%%%%%%

% Alfred Hitchcock

instance(perHitchcock,s_Director).
instance(perHitchcock,s_Screenwriter).
instance(perHitchcock,s_Actor).
att_val(perHitchcock,name,nameHitchcock).

instance(nameHitchcock,name).
att_val(nameHitchcock,firstName,"Alfred").
att_val(nameHitchcock,firstName,"Joseph").
att_val(nameHitchcock,firstName,"Sir").
att_val(nameHitchcock,lastName,"Hitchcock").

prod_period(perHitchcock,1920,1980).
default_genre(perHitchcock,"Thriller").
default_genre(perHitchcock,"Crime").
```

```
% Marlene Dietrich

instance(perDietrich,s_Actor).
att_val(perDietrich,name,nameDietrich).
att_val(perDietrich,birthDate,1901).
att_val(perDietrich,dateOfDeath,1992).

instance(nameDietrich,name).
att_val(nameDietrich,firstName,"Marlene").
att_val(nameDietrich,firstName,"Maria").
att_val(nameDietrich,firstName,"Magdalena").
att_val(nameDietrich,lastName,"Dietrich").

prod_period(perDietrich,1920,1960).
default_genre(perDietrich,"Drama").

% Marilyn Monroe

instance(perMonroe,s_Actor).
att_val(perMonroe,name,nameMM).
att_val(perMonroe,name,nameMonroe).
att_val(perMonroe,birthDate,1926).
att_val(perMonroe,dateOfDeath,1962).

instance(nameMM,name).
att_val(nameMM,firstName,"Marilyn").
att_val(nameMM,lastName,"Monroe").

instance(nameMonroe,name).
att_val(nameMonroe,firstName,"Norma").
att_val(nameMonroe,firstName,"Jean").
att_val(nameMonroe,lastName,"Mortensen").

prod_period(perMonroe,1948,1962).
default_genre(perMonroe,"Comedy").

% Cary Grant

instance(perGrant,s_Actor).
att_val(perGrant,name,nameCG).
att_val(perGrant,name,nameGrant).
att_val(perGrant,birthDate,1904).
att_val(perGrant,dateOfDeath,1986).

instance(nameCG,name).
att_val(nameCG,firstName,"Cary").
att_val(nameCG,lastName,"Grant").

instance(nameGrant,name).
att_val(nameGrant,firstName,"Archibald").
att_val(nameGrant,firstName,"Alec").
att_val(nameGrant,lastName,"Leach").

prod_period(perGrant,1920,1970).
```

```
default_genre(perGrant,"Comedy").

% Grace Kelly

instance(perKelly,s_Actor).
att_val(perKelly,name,nameKelly).
att_val(perKelly,birthDate,1929).
att_val(perKelly,dateOfDeath,1982).

instance(nameKelly,name).
att_val(nameKelly,firstName,"Grace").
att_val(nameKelly,firstName,"Patricia").
att_val(nameKelly,lastName,"Kelly").

prod_period(perKelly,1945,1960).

% John Barry

instance(perBarry,s_Composer).
att_val(perBarry,name,nameBarry).
att_val(perBarry,birthDate,1933).

instance(nameBarry,name).
att_val(nameBarry,firstName,"John").
att_val(nameBarry,firstName,"Prendergast").
att_val(nameBarry,lastName,"Barry").

prod_period(perBarry,1955,X):-current_year(X).

% Elmer Bernstein

instance(perBernstein,s_Composer).
att_val(perBernstein,name,nameBernstein).
att_val(perBernstein,birthDate,1922).

instance(nameBernstein,name).
att_val(nameBernstein,firstName,"Elmer").
att_val(nameBernstein,lastName,"Bernstein").

prod_period(perBernstein,1955,2000).

% Vincent Price

instance(perPrice,s_Actor).
att_val(perPrice,name,namePrice).
att_val(perPrice,birthDate,1911).
att_val(perPrice,dateOfDeath,1993).

instance(namePrice,name).
att_val(namePrice,firstName,"Vincent").
att_val(namePrice,lastName,"Price").

prod_period(perPrice,1935,1990).
default_genre(perPrice,"Horror").
```

```
% John Wayne

instance(perWayne,s_Actor).
instance(perWayne,s_Director).
att_val(perWayne,name,nameWayne).
att_val(perWayne,birthDate,1907).
att_val(perWayne,dateOfDeath,1979).

instance(nameWayne,name).
att_val(nameWayne,firstName,"John").
att_val(nameWayne,lastName,"Wayne").

prod_period(perWayne,1926,1975).
default_genre(perWayne,"CowboyWestern").

% Giorgio Moroder

instance(perMoroder,s_Composer).
att_val(perMoroder,name,nameMoroder).
att_val(perMoroder,birthDate,1940).

instance(nameMoroder,name).
att_val(nameMoroder,firstName,"Giorgio").
att_val(nameMoroder,lastName,"Moroder").

prod_period(perMoroder,1970,X):-current_year(X).

% Sean Connery

instance(perConnery,s_Actor).
att_val(perConnery,name,nameConnery).
att_val(perConnery,birthDate,1930).

instance(nameConnery,name).
att_val(nameConnery,firstName,"Sean").
att_val(nameConnery,lastName,"Connery").

prod_period(perConnery,1955,X):-current_year(X).
default_genre(perConnery,"Action").

% Movies:  (largest movie-id used: m65)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

instance(m1,s_Movie).
att_val(m1,title,"There's No Business Like Show Business").
att_val(m1,releaseDate,1954).
acted(perMonroe,m1).

instance(m2,s_Movie).
att_val(m2,title,"Niagara").
acted(perMonroe,m2).

instance(m3,s_Movie).
att_val(m3,title,"River of no Return").
att_val(m3,releaseDate,1954).
```

```
acted(perMonroe,m3).

instance(m4,s_Movie).
att_val(m4,title,"Bus Stop").
acted(perMonroe,m4).

instance(m5,s_Movie).
att_val(m5,title,"Some Like it Hot").
att_val(m5,releaseDate,1959).
acted(perMonroe,m5).


instance(m6,s_Movie).
att_val(m6,title,"Blaue Engel, Der").
att_val(m6,releaseDate,1930).
acted(perDietrich,m6).

instance(m7,s_Movie).
att_val(m7,title,"Shanghai Express").
att_val(m7,releaseDate,1932).
acted(perDietrich,m7).

instance(m8,s_Movie).
att_val(m8,title,"Blonde Venus").
att_val(m8,releaseDate,1932).
acted(perDietrich,m8).

instance(m9,s_Movie).
att_val(m9,title,"Devil Is a Woman, The").
att_val(m9,releaseDate,1935).
acted(perDietrich,m9).

instance(m10,s_Movie).
att_val(m10,title,"Witness for the Prosecution").
acted(perDietrich,m10).

instance(m11,s_Movie).
att_val(m11,title,"Notorious").
acted(perGrant,m11).
directed(perHitchcock,m11).

instance(m12,s_Movie).
att_val(m12,title,"Arsenic and Old Lace").
att_val(m12,releaseDate,1944).
acted(perGrant,m12).

instance(m13,s_Movie).
att_val(m13,title,"I Was a Male War Bride").
att_val(m13,releaseDate,1949).
acted(perGrant,m13).

instance(m14,s_Movie).
att_val(m14,title,"To Catch a Thief").
acted(perGrant,m14).
acted(perKelly,m14).
```

```
directed(perHitchcock,m14).

instance(m15,s_Movie).
att_val(m15,title,"Houseboat").
acted(perGrant,m15).

instance(m16,s_Movie).
att_val(m16,title,"Charade").
att_val(m16,releaseDate,1963).
acted(perGrant,m16).

instance(m17,s_Movie).
att_val(m17,title,"Rear Window").
acted(perKelly,m17).
directed(perHitchcock,m17).

instance(m18,s_Movie).
att_val(m18,title,"Dial M for Murder").
att_val(m18,releaseDate,1954).
acted(perKelly,m18).
directed(perHitchcock,m18).

instance(m19,s_Movie).
att_val(m19,title,"High Noon").
att_val(m19,releaseDate,1952).
acted(perKelly,m19).

instance(m20,s_Movie).
att_val(m20,title,"High Society").
acted(perKelly,m20).

instance(m21,s_Movie).
att_val(m21,title,"From Russia with Love").
sound_composed(perBarry,m21).
acted(perConnery,m21).

instance(m22,s_Movie).
att_val(m22,title,"Goldfinger").
att_val(m22,releaseDate,1964).
sound_composed(perBarry,m22).
acted(perConnery,m22).

instance(m23,s_Movie).
att_val(m23,title,"Ipcress File, The").
sound_composed(perBarry,m23).

instance(m24,s_Movie).
att_val(m24,title,"Born Free").
sound_composed(perBarry,m24).

instance(m25,s_Movie).
att_val(m25,title,"Lion in Winter, The").
sound_composed(perBarry,m25).

instance(m26,s_Movie).
```

```
att_val(m26,title,"Thunderball").
att_val(m26,releaseDate,1965).
sound_composed(perBarry,m26).

instance(m27,s_Movie).
att_val(m27,title,"King Kong").
att_val(m27,releaseDate,1976).
sound_composed(perBarry,m27).

instance(m28,s_Movie).
att_val(m28,title,"Deep, The").
sound_composed(perBarry,m28).

instance(m29,s_Movie).
att_val(m29,title,"Black Hole, The").
sound_composed(perBarry,m29).

instance(m30,s_Movie).
att_val(m30,title,"View to a Kill, A").
att_val(m30,releaseDate,1985).
sound_composed(perBarry,m30).

instance(m31,s_Movie).
att_val(m31,title,"Howard the Duck").
sound_composed(perBarry,m31).

instance(m32,s_Movie).
att_val(m32,title,"Indicent Proposal").
att_val(m30,releaseDate,1993).
sound_composed(perBarry,m32).

instance(m33,s_Movie).
att_val(m33,title,"Ten Commandments, The").
sound_composed(perBernstein,m33).

instance(m34,s_Movie).
att_val(m34,title,"Man with the Golden Arm, The").
sound_composed(perBernstein,m34).

instance(m35,s_Movie).
att_val(m35,title,"Buccaneer, The").
att_val(m35,releaseDate,1958).
sound_composed(perBernstein,m35).

instance(m36,s_Movie).
att_val(m36,title,"Magnificent Seven, The").
att_val(m36,releaseDate,1960).
sound_composed(perBernstein,m36).

instance(m37,s_Movie).
att_val(m37,title,"To Kill a Mockingbird").
sound_composed(perBernstein,m37).

instance(m38,s_Movie).
att_val(m38,title,"Cannon for Cordoba").
```

```
sound_composed(perBernstein,m38).

instance(m39,s_Movie).
att_val(m39,title,"Ghostbusters").
att_val(m39,releaseDate,1984).
sound_composed(perBernstein,m39).

instance(m40,s_Movie).
att_val(m40,title,"Wild Wild West").
att_val(m40,releaseDate,1999).
sound_composed(perBernstein,m40).

instance(m41,s_Movie).
att_val(m41,title,"Laura").
acted(perPrice,m41).

instance(m42,s_Movie).
att_val(m42,title,"Three Musketeers, The").
att_val(m42,releaseDate,1948).
acted(perPrice,m42).

instance(m43,s_Movie).
att_val(m43,title,"Bat, The").
acted(perPrice,m43).

instance(m44,s_Movie).
att_val(m44,title,"Fly, The").
att_val(m44,releaseDate,1958).
acted(perPrice,m44).

instance(m45,s_Movie).
att_val(m45,title,"Return of the Fly").
att_val(m45,releaseDate,1959).
acted(perPrice,m45).

instance(m46,s_Movie).
att_val(m46,title,"House of Usher").
att_val(m46,releaseDate,1960).
acted(perPrice,m46).

instance(m47,s_Movie).
att_val(m47,title,"Raven, The").
acted(perPrice,m47).

instance(m48,s_Movie).
att_val(m48,title,"Stagecoach").
acted(perWayne,m48).

instance(m49,s_Movie).
att_val(m49,title,"Alamo, The").
att_val(m49,releaseDate,1960).
acted(perWayne,m49).
directed(perWayne,m49).

instance(m50,s_Movie).
```

```
att_val(m50,title,"Green Berets, The").
att_val(m50,releaseDate,1968).
acted(perWayne,m50).
directed(perWayne,m50).

instance(m51,s_Movie).
att_val(m51,title,"Rio Grande").
att_val(m51,releaseDate,1950).
acted(perWayne,m51).

instance(m52,s_Movie).
att_val(m52,title,"Sands of Iwo Jima").
acted(perWayne,m52).

instance(m53,s_Movie).
att_val(m53,title,"Flying Tigers").
acted(perWayne,m53).

instance(m54,s_Movie).
att_val(m54,title,"Man Who Shot Liberty Valance, The").
att_val(m54,releaseDate,1962).
acted(perWayne,m54).

instance(m55,s_Movie).
att_val(m55,title,"American Gigolo").
att_val(m55,releaseDate,1980).
sound_composed(perMoroder,m55).

instance(m56,s_Movie).
att_val(m56,title,"Midnight Express").
att_val(m56,releaseDate,1978).
sound_composed(perMoroder,m56).

instance(m57,s_Movie).
att_val(m57,title,"Over the Top").
att_val(m57,releaseDate,1987).
sound_composed(perMoroder,m57).

instance(m58,s_Movie).
att_val(m58,title,"Dr. No").
att_val(m58,releaseDate,1962).
acted(perConnery,m58).

instance(m59,s_Movie).
att_val(m59,title,"Zardoz").
att_val(m59,releaseDate,1974).
acted(perConnery,m59).

instance(m60,s_Movie).
att_val(m60,title,"Bridge Too Far, A").
acted(perConnery,m60).

instance(m61,s_Movie).
att_val(m61,title,"Meteor").
att_val(m61,releaseDate,1979).
```

```
acted(perConnery,m61).

instance(m62,s_Movie).
att_val(m62,title,"Highlander").
att_val(m62,releaseDate,1986).
acted(perConnery,m62).

instance(m63,s_Movie).
att_val(m63,title,"Name der Rose, Der").
att_val(m63,releaseDate,1986).
acted(perConnery,m63).

instance(m64,s_Movie).
att_val(m64,title,"Hunt for Red October, The").
att_val(m64,releaseDate,1990).
acted(perConnery,m64).

instance(m65,s_Movie).
att_val(m65,title,"Avengers, The").
att_val(m65,releaseDate,1998).
acted(perConnery,m65).
```

E Appendix: Experimental Source-Description Program

The following ELP served as the source-description program, Π_{sd}, for the sources used in our experiments:

```
%%%%%%%%%%%%%%%%%%%
% Technical aspects
%%%%%%%%%%%%%%%%%%%

source(s_Eachmovie).
source(s_Hitchcock).
source(s_Horror60).
source(s_KellyGrant).
source(s_RandomMovies).                        % not for "reduced" selection base
source(s_RandomPersons).
source(s_RandomMoviesNew).                     % only for "extended" selection base

up(s_Eachmovie).
up(s_Hitchcock).
up(s_Horror60).
up(s_KellyGrant).
up(s_RandomMovies).                          % not for "reduced" selection base
up(s_RandomPersons).
up(s_RandomMoviesNew).                       % only for "extended" selection base

source_type(S,public,queryable) :- source(S), not -source_type(S,public,queryable).
-source_type(S,public,queryable) :- source_type(S,T1,T2), !=(T1,public).
-source_type(S,public,queryable) :- source_type(S,T1,T2), !=(T2,queryable).

source_language(S,"English") :- source(S), not -source_language(S,"English").
-source_language(S,"English") :- source_language(S,L), !=(L,"English").
```

```
data_format(s_Eachmovie,xml).
data_format(s_Hitchcock,xml).
data_format(s_Horror60,xml).
data_format(s_KellyGrant,xml).
data_format(s_RandomMovies,xml).             % not for "reduced" selection base
data_format(s_RandomPersons,xml).
data_format(s_RandomMoviesNew,xml).          % only for "extended" selection base

reliable(S,high) :- source(S), not -reliable(S,high).
-reliable(S,high) :- reliable(S,V), !=(V,high).

update_frequency(s_Eachmovie,medium).
update_frequency(s_Hitchcock,low).
update_frequency(s_Horror60,low).
update_frequency(s_KellyGrant,low).
update_frequency(s_RandomMovies,high).       % not for "reduced" selection base
update_frequency(s_RandomPersons,high).
update_frequency(s_RandomMoviesNew,high).    % only for "extended" selection base

%%%%%%%%%%%%%%
% Cost aspects
%%%%%%%%%%%%%%

charge(S,no) :- source(S), not -charge(S,no).
-charge(S,no) :- charge(S,V), !=(V,no).

%%%%%%%%%%%%%%%%%%
% Thematic aspects
%%%%%%%%%%%%%%%%%%

topic(T) :- accurate(S,T,V).
topic(T) :- -accurate(S,T,V).
topic(T) :- covers(S,T,V).
topic(T) :- -covers(S,T,V).
topic(T) :- specialized(S,T).
topic(T) :- -specialized(S,T).
topic(T) :- relevant(S,T).
topic(T) :- -relevant(S,T).

accurate(S,T,high) :- source(S), topic(T), not -accurate(S,T,high).
-accurate(S,T,high) :- accurate(S,T,V), !=(V,high).
-accurate(S,T,high) :- -relevant(S,T).

covers(S,T,high) :- specialized(S,T), not -covers(S,T,high).
-covers(S,T,high) :- covers(S,T,V), !=(V,high).
covers(S,T,medium) :- relevant(S,T), not specialized(S,T),
                      not -covers(S,T,medium).
-covers(S,T,medium) :- covers(S,T,V), !=(V,medium).
covers(S,T,low) :- -relevant(S,T), not -covers(S,T,low).
-covers(S,T,low) :- covers(S,T,V), !=(V,low).

covers(s_Hitchcock,s_Movie,low).
covers(s_Hitchcock,s_ReleaseDate,high).
covers(s_Hitchcock,s_Review,high).
covers(s_Hitchcock,s_Soundtrack,low).
```

```
covers(s_Hitchcock,s_Person,high).
covers(s_Hitchcock,s_BirthDate,high).
covers(s_Hitchcock,s_Actor,high).
covers(s_Hitchcock,s_Director,low).
covers(s_Hitchcock,s_Composer,low).
covers(s_Hitchcock,fifties,high).

covers(s_Horror60,s_Movie,low).
covers(s_Horror60,s_ReleaseDate,high).
covers(s_Horror60,s_Genre,low).
covers(s_Horror60,s_Review,high).
covers(s_Horror60,s_Soundtrack,low).
covers(s_Horror60,s_Person,high).
covers(s_Horror60,s_Actor,high).
covers(s_Horror60,s_Composer,low).
covers(s_Horror60,sixties,high).

covers(s_KellyGrant,s_Movie,low).
covers(s_KellyGrant,s_ReleaseDate,high).
covers(s_KellyGrant,s_Person,high).
covers(s_KellyGrant,s_BirthDate,high).
covers(s_KellyGrant,s_Actor,high).
covers(s_KellyGrant,fifties,high).
covers(s_KellyGrant,sixties,high).

> covers(s_RandomMovies,s_Composer,high).
114a120,128
> covers(s_RandomMovies,"CowboyWestern",high).

covers(s_RandomMovies,s_ReleaseDate,high).   % not for "reduced" selection base
covers(s_RandomMovies,s_Genre,high).         % not for "reduced" selection base
covers(s_RandomMovies,s_Review,low).         % not for "reduced" selection base
covers(s_RandomMovies,s_Soundtrack,high).    % not for "reduced" selection base
covers(s_RandomMovies,s_Person,high).        % not for "reduced" selection base
covers(s_RandomMovies,s_Actor,high).         % not for "reduced" selection base
covers(s_RandomMovies,s_Composer,high).      % only for "extended" selection base
covers(s_RandomMovies,s_Screenwriter,high).  % not for "reduced" selection base
covers(s_RandomMovies,"CowboyWestern",high). % only for "extended" selection base

covers(s_RandomMoviesNew,s_Review,low).      % only for "extended" selection base
covers(s_RandomMoviesNew,s_Soundtrack,high). % only for "extended" selection base
covers(s_RandomMoviesNew,s_Person,high).     % only for "extended" selection base
covers(s_RandomMoviesNew,s_Actor,high).      % only for "extended" selection base
covers(s_RandomMoviesNew,s_Director,high).   % only for "extended" selection base
covers(s_RandomMoviesNew,"Drama",high).      % only for "extended" selection base
covers(s_RandomMoviesNew,"Comedy",high).     % only for "extended" selection base

covers(s_RandomPersons,s_Movie,low).
covers(s_RandomPersons,s_Actor,high).
covers(s_RandomPersons,s_Director,high).
covers(s_RandomPersons,s_Screenwriter,high).

specialized(s_Eachmovie,s_Rating).
specialized(s_Hitchcock,"Hitchcock").
specialized(s_Horror60,"Horror").
```

```
specialized(s_KellyGrant,"Kelly").
specialized(s_KellyGrant,"Grant").
specialized(s_RandomMovies,s_Movie).         % not for "reduced" selection base
specialized(s_RandomPersons,s_Person).
specialized(s_RandomMoviesNew,s_Movie).      % only for "extended" selection base

relevant(S,T) :- source(S), topic(T), not -relevant(S,T).

-relevant(s_Eachmovie,s_Person).
-relevant(s_Eachmovie,s_BirthDate).
-relevant(s_Eachmovie,s_Soundtrack).
-relevant(s_Eachmovie,s_RunningTime).

-relevant(s_Hitchcock,eighties).
-relevant(s_Hitchcock,nineties).
-relevant(s_Hitchcock,twothousands).

-relevant(s_Horror60,twenties).
-relevant(s_Horror60,thirties).
-relevant(s_Horror60,fourties).
-relevant(s_Horror60,fifties).
-relevant(s_Horror60,seventies).
-relevant(s_Horror60,eighties).
-relevant(s_Horror60,nineties).
-relevant(s_Horror60,twothousands).

-relevant(s_KellyGrant,seventies).
-relevant(s_KellyGrant,eighties).
-relevant(s_KellyGrant,nineties).
-relevant(s_KellyGrant,twothousands).

-relevant(s_RandomMovies,s_BirthDate).       % not for "reduced" selection base
-relevant(s_RandomMoviesNew,s_BirthDate).    % only for "extended" selection base

-relevant(s_RandomPersons,s_ReleaseDate).
-relevant(s_RandomPersons,s_RunningTime).
-relevant(s_RandomPersons,s_Review).
-relevant(s_RandomPersons,s_Soundtrack).
```

F Appendix: Experimental Source-Selection Program

The following source-selection program, Π_{sel}, has been used for experimentation. Note some slight syntactical differences. Similar to `plp`, our implementation requires the rule name to be the first predicate in the body of a source-selection rule. The arity of the name predicate is determined by the open variables of the source-selection rule.[5] Furthermore, the performance of our current implementation can be enhanced, by using $true$ as a name predicate for all rules, that neither occur in the user preference relation $<_u$, nor contribute to implicit preferences $\preceq_c$.[6]

[5] Note, that a simple preprocessor could perform this syntactical transformation.

[6] Such optimizations could also be automated by means of the aforementioned preprocessor, since a source-selection rule r does not contribute to implicit preferences $\preceq_c$, iff non of the predicates $access$, $default_class$, or $default_path$ occurs in $B(r)$.

```
%%%%%%%%%%%%%%%
% Default rules
%%%%%%%%%%%%%%%

query_source(S,Q) :- r1(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), specialized(S,P).
query_source(S,Q) :- r2(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), covers(S,P,high).
query_source(S,Q) :- r3(S,Q,O,P,V), default_path(O,P,Q), occurs(O,V), relevant(S,P).

query_source(S,Q) :- r4(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V),
                     specialized(S,T).
query_source(S,Q) :- r5(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V),
                     covers(S,T,high).
query_source(S,Q) :- r6(S,Q,O,T,V), default_class(O,T,Q), occurs(O,V), relevant(S,T).

% not for "reduced" selection base
query_source(s_RandomMovies,Q) :- r20(Q,O), default_class(O,s_Movie,Q).

query_source(s_RandomPersons,Q) :- r21(Q,O), default_class(O,s_Person,Q).

% only for "extended" selection base
query_source(s_RandomMoviesNew,Q) :- r22(Q,O), default_class(O,s_Movie,Q).

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Core source-selection rules
%%%%%%%%%%%%%%%%%%%%%%%%%%

query_source(S,Q) :- r40(S,Q), source(S), query(Q), high_coverage(S,Q).
query_source(S,Q) :- r41(S,Q), source(S), query(Q), special(S,Q).


query_source(s_Eachmovie,Q) :- r60(Q,O,C,V),
                               access(O,s_MovieDB,s_Movie_Review_Rating,Q),
                               selects(O,C,V).
query_source(s_Hitchcock,Q) :- r61(Q,O1,O2),
                               access(O1,s_Person,s_LastName,Q),
                               selects(O1,equal,"Hitchcock"),
                               access(O2,s_Person,s_FirstName,Q),
                               selects(O2,equal,"Alfred").
query_source(s_Horror60,Q) :- r62(Q,O), access(O,s_MovieDB,s_Movie_Genre,Q),
                              selects(O,equal,"Horror"),
                              inferred_topic(Q,sixties).
query_source(s_KellyGrant,Q) :-  r63(Q,O1,O2),
                                 access(O1,s_Person,s_LastName,Q),
                                 selects(O1,equal,"Kelly"),
                                 access(O2,s_Person,s_FirstName,Q),
                                 selects(O2,equal,"Grace").
query_source(s_KellyGrant,Q) :-  r64(Q,O1,O2),
                                 access(O1,s_Person,s_LastName,Q),
                                 selects(O1,equal,"Grant"),
                                 access(O2,s_Person,s_FirstName,Q),
                                 selects(O2,equal,"Cary").
```

```
%%%%%%%%%%%%%
% Preferences
%%%%%%%%%%%%%

r2(_,Q,_,_,_)<r1(_,Q,_,_,_).
r3(_,Q,_,_,_)<r1(_,Q,_,_,_).
r3(_,Q,_,_,_)<r2(_,Q,_,_,_).

r5(_,Q,_,_,_)<r4(_,Q,_,_,_).
r6(_,Q,_,_,_)<r4(_,Q,_,_,_).
r6(_,Q,_,_,_)<r5(_,Q,_,_,_).

r1(_,Q,_,_,_)<r40(_,Q).
r1(_,Q,_,_,_)<r41(_,Q).
r2(_,Q,_,_,_)<r40(_,Q).
r2(_,Q,_,_,_)<r41(_,Q).
r3(_,Q,_,_,_)<r40(_,Q).
r3(_,Q,_,_,_)<r41(_,Q).
r4(_,Q,_,_,_)<r40(_,Q).
r4(_,Q,_,_,_)<r41(_,Q).
r5(_,Q,_,_,_)<r40(_,Q).
r5(_,Q,_,_,_)<r41(_,Q).
r6(_,Q,_,_,_)<r40(_,Q).
r6(_,Q,_,_,_)<r41(_,Q).

r20(Q,_)<r41(_,Q).       % not for "reduced" selection base
r21(Q,_)<r41(_,Q).
r22(Q,_)<r41(_,Q).       % only for "extended" selection base

r40(_,Q)<r41(_,Q).

r60(Q,_,_,_)<r40(_,Q).
r60(Q,_,_,_)<r41(_,Q).

r20(Q,_)<r61(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r61(Q,_,_).
r22(Q,_)<r61(Q,_,_).     % only for "extended" selection base
r20(Q,_)<r62(Q,_).       % not for "reduced" selection base
r21(Q,_)<r62(Q,_).
r22(Q,_)<r62(Q,_).       % only for "extended" selection base
r20(Q,_)<r63(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r63(Q,_,_).
r22(Q,_)<r63(Q,_,_).     % only for "extended" selection base
r20(Q,_)<r64(Q,_,_).     % not for "reduced" selection base
r21(Q,_)<r64(Q,_,_).
r22(Q,_)<r64(Q,_,_).     % only for "extended" selection base
r40(_,Q)<r61(Q,_,_).
r41(_,Q)<r61(Q,_,_).
r40(_,Q)<r62(Q,_).
r41(_,Q)<r62(Q,_).
r40(_,Q)<r63(Q,_,_).
r41(_,Q)<r63(Q,_,_).
r40(_,Q)<r64(Q,_,_).
r41(_,Q)<r64(Q,_,_).
```

```
%%%%%%%%%%%%%%%%%%
% Auxiliary rules (largest rule-id used: 103)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

-query_source(s_RandomPersons,Q) :- r70(Q,O),
                     access(O,s_MovieDB,s_Movie_Actor_Personalia,Q).
-query_source(s_RandomPersons,Q) :- r71(Q,O),
                     access(O,s_MovieDB,s_Movie_Director_Personalia,Q).
-query_source(s_RandomPersons,Q) :- r72(Q,O),
                     access(O,s_MovieDB,s_Movie_Composer_Personalia,Q).
-query_source(s_RandomPersons,Q) :- r73(Q,O),
                     access(O,s_MovieDB,s_Movie_Screenwriter_Personalia,Q).

-query_source(S,Q) :- true, irrelevant(S,Q).


high_coverage(S,Q) :- true, high_covered_topic(S,Q,T).
special(S,Q) :- true, special_topic(S,Q,T).

high_covered_topic(S,Q,T) :- true, query_topic(Q,T), covers(S,T,high).
high_covered_topic(S,Q,T) :- true, inferred_topic(Q,T), covers(S,T,high).

special_topic(S,Q,T) :- true, query_topic(Q,T), specialized(S,T).
special_topic(S,Q,T) :- true, inferred_topic(Q,T), specialized(S,T).

query_topic(Q,T) :- r80(Q,T,O,C,P), access(O,C,P,Q), selects(O,equal,T).
query_topic(Q,T) :- r81(Q,T,O,D,Y),
                    access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                    selects(O,equal,D), calender_year(D,Y), decade(Y,T).
query_topic(Q,T) :- r82(Q,T,O), access(O,T,s_Personalia,Q).

inferred_topic(Q,T) :- true, asks_fixed_range(Q,D1,D2),
                       decade_between(T,D1,D2).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       att_val(P,name,N), att_val(N,lastName,T).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), possible_genre(M,T),
                       not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       possible_genre(P,T), not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingPerson(Q,O,P), possible_genre(P,T),
                       not asks_fixed_genre(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), release_bound(M,B,E),
                       decade(B,D1), decade(E,D2), decade_between(T,D1,D2),
                       not asks_fixed_date(Q).
inferred_topic(Q,T) :- true, matchingMovie(Q,M), involved(P,M),
                       prod_period(P,B,E), calender_year(B,B1),
                       calender_year(E,E1), decade(B1,D1), decade(E1,D2),
                       decade_between(T,D1,D2),  not asks_fixed_date(Q).
inferred_topic(Q,T) :- true,  matchingPerson(Q,O,P), prod_period(P,B,E),
                       calender_year(B,B1), calender_year(E,E1),
                       decade(B1,D1), decade(E1,D2), decade_between(T,D1,D2),
                       not asks_fixed_date(Q).
```

```
asks_fixed_date(Q) :- r83(Q,O,D),
                      access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                      selects(O,equal,D).

asks_fixed_genre(Q) :- r84(Q,O,G), access(O,s_MovieDB,s_Movie_Genre,Q),
                       selects(O,equal,G).

asks_fixed_range(Q,D1,D2) :- r85(Q,D1,D2,O,V1,V2,Y1,Y2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greater,V1), selects(O,less,V2),
                       +(V1,1,Y1), V2=Y2+1, calender_year(Y1,B),
                       calender_year(Y2,E), decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r86(Q,D1,D2,O,V1,V2,Y1,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greater,V1), selects(O,lessequal,V2),
                       +(V1,1,Y1), calender_year(Y1,B), calender_year(V2,E),
                       decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r87(Q,D1,D2,O,V1,V2,Y2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greaterequal,V1), selects(O,less,V2),
                       +(Y2,1,V2), calender_year(V1,B), calender_year(Y2,E),
                       decade(B,D1), decade(E,D2).
asks_fixed_range(Q,D1,D2) :- r88(Q,D1,D2,O,V1,V2,B,E),
                       access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                       selects(O,greaterequal,V1), selects(O,lessequal,V2),
                       calender_year(V1,B), calender_year(V2,E),
                       decade(B,D1), decade(E,D2).

irrelevant(S,Q) :- r90(S,Q,O,C,P), access(O,C,P,Q), -relevant(S,C).
irrelevant(S,Q) :- r91(S,Q,O,C,P), access(O,C,P,Q), -relevant(S,P).
irrelevant(S,Q) :- r92(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,equal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D).
irrelevant(S,Q) :- r93(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,greaterequal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D), irrelevant_after(S,D).
irrelevant(S,Q) :- r94(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,greater,V), calender_year(V,Y), decade(Y,D),
                   source(S), irrelevant_after(S,D).
irrelevant(S,Q) :- r95(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,lessequal,V), calender_year(V,Y), decade(Y,D),
                   -relevant(S,D), irrelevant_before(S,D).
irrelevant(S,Q) :- r96(S,Q,O,V,Y,D),
                   access(O,s_MovieDB,s_Movie_ReleaseDate_Date,Q),
                   selects(O,less,V), calender_year(V,Y), decade(Y,D),
                   source(S), irrelevant_before(S,D).
irrelevant(S,Q) :- r97(S,Q,O,V,G), access(O,s_MovieDB,s_Movie_Genre,Q),
                   selects(O,equal,V), specialized(S,G), genre(G),
                   !=(V,G), !=(V,"Other").
irrelevant(S,Q) :- true, matchingAlive(Q,P), prod_period(P,X,Y),
                   calender_year(X,B), calender_year(Y,E), decade(B,D1),
```

```
                    decade(E,D2), irrelevant_range(S,D1,D2).
irrelevant(S,Q) :- true, matchingAlive(Q,P), life_period(P,B,E),
                    decade(B,D1), decade(E,D2),  irrelevant_range(S,D1,D2).

irrelevant_before(S,D) :- true, source(S), decade_name(D), !=(D,twenties),
                           not relevant_before(S,D).
irrelevant_after(S,D) :- true, source(S), decade_name(D), !=(D,twothousands),
                          not relevant_after(S,D).
irrelevant_range(S,D1,D2) :- true, source(S), decade_name(D1), decade_name(D2),
                             not relevant_between(S,D1,D2).
relevant_before(S,D) :- true, decade_before(D1,D), relevant(S,D1).
relevant_after(S,D) :- true, decade_after(D1,D), relevant(S,D1).
relevant_between(S,D1,D2) :- true, decade_between(D,D1,D2), relevant(S,D).


matchingAlive(Q,P) :- r100(Q,P,O,P1), instance(P,s_Person),
                      matchingPerson(Q,O,P), access(O,s_Actor,P1,Q).
matchingAlive(Q,P) :- r101(Q,P,O,P1), instance(P,s_Person),
                      matchingPerson(Q,O,P), access(O,s_Director,P1,Q).

matchingPerson(Q,O,P) :- r102(Q,O,P,L,O1,F,N),
                         access(O,s_Person,s_LastName,Q), selects(O,equal,L),
                         access(O1,s_Person,s_FirstName,Q),
                         selects(O1,equal,F), instance(P,s_Person),
                         att_val(P,name,N), att_val(N,lastName,L),
                         att_val(N,firstName,F).

matchingMovie(Q,M) :- r103(Q,M,O,T), access(O,s_MovieDB,s_Movie_Title,Q),
                      selects(O,equal,T), instance(M,s_Movie),
                      att_val(M,title,T).



%%%%%%%%%%%%%%%%%%
% Weak constraints
%%%%%%%%%%%%%%%%%%

:~ query_source(S,Q), charge(S,high). [3:3]
:~ query_source(S,Q), charge(S,medium). [2:3]
:~ query_source(S,Q), charge(S,low). [1:3]

:~ query_source(S,Q), reliable(S,low). [8:2]
:~ query_source(S,Q), reliable(S,medium). [4:2]

:~ query_source(S,Q), query_topic(Q,T),accurate(S,T,low). [15:1]
:~ query_source(S,Q), special(S1,Q), not special(S,Q). [12:1]
:~ query_source(S,Q), high_covered_topic(S1,Q,T), not decade_name(T),
                      not class0(T), not high_coverage(S,Q). [10:1]
:~ query_source(S,Q), query_topic(Q,T), not class0(T), not specialized(S,T). [8:1]
:~ query_source(S,Q), query_topic(Q,T), not class0(T), not covers(S,T,high). [7:1]
:~ query_source(S,Q), inferred_topic(Q,T), not decade_name(T),
                      not specialized(S,T). [6:1]
:~ query_source(S,Q), inferred_topic(Q,T), not decade_name(T),
                      not covers(S,T,high). [5:1]
:~ query_source(S,Q), default_class(O,T,Q), constructs(O,C,P),
```

```
                    specialized(S1,T), not specialized(S,T). [4:1]
:~ query_source(S,Q), default_class(O,T,Q), constructs(O,C,P),
                    covers(S1,T,high), not covers(S,T,high). [3:1]
:~ query_source(S,Q), high_covered_topic(S1,Q,T), class0(T),
                    not high_covered_topic(S,Q,T). [2:1]
:~ query_source(S,Q), high_covered_topic(S1,Q,T), decade_name(T),
                    not high_covered_topic(S,Q,T). [1:1]
```

G Appendix: Experimental XML-QL Queries

Below, the XML-QL representation of all experimental queries is given, except for queries Q_1 and Q_2 which appear in Examples 1 and 7:

Q_3 : *In which year the movie 'Arsenic and Old Lace' has been released?*

```
FUNCTION MovieQuery03($MovieDB:"movie.dtd") {
  CONSTRUCT <ArsenicAndOldLace> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> "Arsenic and Old Lace" </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
        </Movie>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <released> $d </released>
  } </ArsenicAndOldLace> }
```

Q_4 : *In which movies directed by Alfred Hitchcock acted Marlene Dietrich?*

```
FUNCTION MovieQuery04($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Director Personalia=$pid>
          </Director>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
        <Person ID=$pid>
          <FirstName.PCDATA>"Alfred"</>
          <LastName.PCDATA>"Hitchcock"</>
        </Person>
        <Person ID=$pact>
          <FirstName.PCDATA>"Marlene"</>
          <LastName.PCDATA>"Dietrich"</>
```

```
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

Q_5 : *In which film noirs did Marilyn Monroe act?*

```
FUNCTION MovieQuery05($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="FilmNoir">
          <Title> $t </Title>
          <ReleaseDate> $d </ReleaseDate>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_6 : *Which movies did Laurel and Hardy act in 1940?*

```
FUNCTION MovieQuery06($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact1> </Actor>
          <Actor Personalia=$pact2> </Actor>
        </Movie>
        <Person ID=$pact1>
          <FirstName.PCDATA>"Stan"</>
          <LastName.PCDATA>"Laurel"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Oliver"</>
          <LastName.PCDATA>"Hardy"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $d = 1940
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_7 : *Which movies acted by Frank Sinatra have a soundtrack by Elmer Bernstein?*

```
FUNCTION MovieQuery07($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
          <Soundtrack>
            <Composer Personalia=$pcomp> </Composer>
          </Soundtrack>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Frank"</>
          <LastName.PCDATA>"Sinatra"</>
        </Person>
        <Person ID=$pcomp>
          <FirstName.PCDATA>"Elmer"</>
          <LastName.PCDATA>"Bernstein"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_8 : *When was James Dean born?*

```
FUNCTION MovieQuery08($MovieDB:"movie.dtd") {
  CONSTRUCT <JamesDean> {
     WHERE
      <MovieDB>
        <Person>
          <FirstName.PCDATA>"James"</>
          <LastName.PCDATA>"Dean"</>
          <BirthDate>
            <Date> $d </Date>
          </BirthDate>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <born> $d </born>
  } </JamesDean> }
```

Q_9 : *In which movies did Jane Russell and Marilyn Monroe act together?*

```
FUNCTION MovieQuery09($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Actor Personalia=$pact1>
          </Actor>
          <Actor Personalia=$pact2>
          </Actor>
        </Movie>
        <Person ID=$pact1>
          <FirstName.PCDATA>"Jane"</>
          <LastName.PCDATA>"Russell"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

Q_{10} : *In which movies did Cary Grant and Marilyn Monroe act together?*

```
FUNCTION MovieQuery10($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <Actor Personalia=$pact1>
          </Actor>
          <Actor Personalia=$pact2>
          </Actor>
        </Movie>
        <Person ID=$pact1>
          <FirstName.PCDATA>"Cary"</>
          <LastName.PCDATA>"Grant"</>
        </Person>
        <Person ID=$pact2>
          <FirstName.PCDATA>"Marilyn"</>
          <LastName.PCDATA>"Monroe"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <Movie> $t </Movie>
  } </MovieList> }
```

Q_{11} : *In which comedies did John Wayne act?*

```
FUNCTION MovieQuery11($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Comedy">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact>
          </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"John"</>
          <LastName.PCDATA>"Wayne"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_{12} : *Which movies acted by Sean Connery have a soundtrack by John Barry?*

```
FUNCTION MovieQuery12($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
          <Soundtrack>
            <Composer Personalia=$pcomp> </Composer>
          </Soundtrack>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Sean"</>
          <LastName.PCDATA>"Connery"</>
        </Person>
        <Person ID=$pcomp>
          <FirstName.PCDATA>"John"</>
          <LastName.PCDATA>"Barry"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_{13} : *In which horror movies of the sixties did Leonard Nimoy act?*

```
FUNCTION MovieQuery13($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Horror">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Leonard"</>
          <LastName.PCDATA>"Nimoy"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $d > 1959,
      $d < 1970
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_{14} : *In which animation movies did Vincent Price act?*

```
FUNCTION MovieQuery14($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie Genre="Animation">
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"Vincent"</>
          <LastName.PCDATA>"Price"</>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_{15} : *In which movies, except crime movies, did James Cagney act between 1930 and 1940?*

```
FUNCTION MovieQuery15($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
```

```
      <MovieDB>
        <Movie Genre=$gen>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Actor Personalia=$pact> </Actor>
        </Movie>
        <Person ID=$pact>
          <FirstName.PCDATA>"James"</>
          <LastName.PCDATA>"Cagney"</>
        </Person>
      </MovieDB> IN source($MovieDB),
      $gen != "Crime",
      $d > 1930,
      $d < 1940
    CONSTRUCT
      <Movie> $t </Movie>
      <Date> $d </Date>
  } </MovieList> }
```

Q_{16} : *Where is Giorgio Moroder from?*

```
FUNCTION MovieQuery16($MovieDB:"movie.dtd") {
  CONSTRUCT <Moroder> {
     WHERE
      <MovieDB>
        <Person>
          <FirstName.PCDATA>"Giorgio"</>
          <LastName.PCDATA>"Moroder"</>
          <Country> $c </Country>
        </Person>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT <from> $c </from>
  } </Moroder> }
```

Q_{17} : *In which year the movie 'Witness for the Prosecution' has been released?*

```
FUNCTION MovieQuery17($MovieDB:"movie.dtd") {
  CONSTRUCT <WitnessForTheProsecution> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> "Witness for the Prosecution" </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
        </Movie>
      </MovieDB> IN source($MovieDB)
    CONSTRUCT
      <released> $d </released>
  } </WitnessForTheProsecution> }
```

Q_{18} : *Which movies from 1999 have been rated above 90 points (out of 100)?*

```
FUNCTION MovieQuery18($MovieDB:"movie.dtd") {
  CONSTRUCT <MovieList> {
     WHERE
      <MovieDB>
        <Movie>
          <Title> $t </Title>
          <ReleaseDate><Date> $d </Date></ReleaseDate>
          <Review><Rating> $r </Rating></Review>
        </Movie>
      </MovieDB> IN source($MovieDB),
      $d = 1999,
      $r > 90
    CONSTRUCT
      <Movie> $t </Movie>
      <rated> $r </rated>
  } </MovieList> }
```