9.4: Principal Components

The theoretical details are quite involved, and relevant papers span hundreds of pages. In essence, there are two principal aspects to it:

1. Ontology-Based Data Access systems (static components):
   - An ontology language, for representing the ontology
   - A mapping language, for declaring the mappings between vocabulary in the ontology and the data
   - The data

2. Query answering in Ontology-Based Data Access systems:
   - Reasoning over the TBox
   - Query rewriting
   - Query unfolding
   - Relational database technology

The mapping language and the items under 2 are new notions compared Blocks I and II and a regular database course, which will be described in the following subsections.

The Ontology Language

For the language, we remain, roughly, within the Semantic Web setting, and take a closer look at the OWL 2 QL profile and similar languages in the “DL-lite” family that is at the basis of OWL 2 QL, and \( \\text{DL-Lite}_\{\text{mathcal\{A\}}\} \) in particular [CGL"07]. Most significantly, the trade-off between expressive power and computational complexity of the reasoning
services leans strongly towards the scalability of reasoning (including query answering) over large amounts of data; or:

one can’t say a lot with either OWL 2 QL or \(\text{DL-Lite}_{\mathcal{A}}\).

Syntax of \(\text{DL-Lite}_{\mathcal{A}}\)

As is common in DLs and OWL, we distinguish between (abstract) objects and (data) values. A **class expression** denotes a set of objects, a **datatype** denotes a set of values, an **object property** denotes a binary relationship between objects, and a **data property** denotes a binary relation between objects and values. We assume to have a set \(\{T_{1}, \ldots, T_{n}\}\) of pairwise disjoint and unbounded datatypes, each denoting a set \(\text{val}(T_{i})\) of values (integers, strings, etc.), and \(\text{top}\langle d\rangle\) denotes the set of all values. Class expressions, \(\langle C\rangle\), and object property expressions, \(\langle R\rangle\), are formed according to the following syntax, where \(\langle A\rangle\) denotes a class, \(\langle P\rangle\) an object property, and \(\langle U\rangle\) a data property:

\[
\langle C \longrightarrow \text{longrightarrow} A \mid \exists R \rangle \delta (U), \langle R \longrightarrow \text{longrightarrow} P \mid P^{\rightarrow} \rangle.
\]

Observe that \(\langle \exists R \rangle\) denotes an unqualified existential class expression, \(\langle \delta (U) \rangle\) denotes the domain of \(\langle U \rangle\), and \(\langle P^{\rightarrow} \rangle\) denotes the inverse of \(\langle P \rangle\).

A \(\text{DL-Lite}_{\mathcal{A}}\) ontology \(\langle \mathcal{T}, \mathcal{A} \rangle\), consists of a TBox \(\langle \mathcal{T} \rangle\) and an ABox \(\langle \mathcal{A} \rangle\), where the TBox is constituted by a set of **axioms** of the form

\[
\langle C_{1}\rangle \sqsubseteq \langle C_{2}\rangle, \langle \rho (U) \rangle \sqsubseteq \langle T_{i} \rangle, \langle R_{1} \rangle \sqsubseteq \langle R_{2} \rangle, \langle U_{1} \rangle \sqsubseteq \langle U_{2} \rangle, \langle \text{disj} C_{1} C_{2} \rangle, \langle \text{disj} R_{1} R_{2} \rangle, \langle \text{disj} U_{1} U_{2} \rangle, \langle \text{funct} R \rangle, \langle \text{funct} U \rangle.
\]

The axioms in the first row denote **inclusions**, with \(\langle \rho (U) \rangle\) the range of \(\langle U \rangle\). The axioms in the second row denote **disjointness**; note that distinct datatypes are assumed to be disjoint. The axioms in the third row denote **functionality** (at most one) of an object property expression and of a data property expression, respectively. The ABox is constituted by a set of **assertions** of the form \(\langle A(a), P(a, a^{'}) \rangle\), and \(\langle U(a, \mathcal{l}) \rangle\), where \(a, a^{'}\) are individuals denoting objects and \(\langle \mathcal{l} \rangle\) is a literal (denoting a value). To ensure that \(\text{DL-Lite}_{\mathcal{A}}\) maintains the computationally wellbehaved computational properties of the \(\text{DL-Lite}\) family [CGL\textsuperscript{+}07], the form of the TBox has to be restricted (as we have seen for OWL 2 QL in Chapter 3). In particular, object and data properties occurring in functionality assertions cannot be specialized, i.e., the cannot appear in the right hand side of an inclusion axiom.

Semantics of \(\text{DL-Lite}_{\mathcal{A}}\)

The semantics of \(\text{DL-Lite}_{\mathcal{A}}\) is based on first-order **interpretations** \(\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})\), where \(\Delta^{\mathcal{I}}\) is a nonempty interpretation domain, which is partitioned into a \(\Delta^{\mathcal{I}}_{O}\) of objects and a \(\Delta^{\mathcal{I}}_{V}\) of values. The interpretation function \(\cdot^{\mathcal{I}}\) maps each individual \(\langle a \rangle\) to \(\langle a^{\mathcal{I}} \rangle\) in \(\Delta^{\mathcal{I}}_{O}\), each class \(\langle A \rangle\) to \(\langle A^{\mathcal{I}} \rangle\), each object property \(\langle P \rangle\) to \(\langle P^{\mathcal{I}} \rangle\), and each data property \(\langle U \rangle\) to

\[
\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}) \quad \text{where} \quad \Delta^{\mathcal{I}}_{O} \quad \Delta^{\mathcal{I}}_{V} \quad \cdot^{\mathcal{I}}(a) = \langle a^{\mathcal{I}} \rangle \quad \cdot^{\mathcal{I}}(A) = \langle A^{\mathcal{I}} \rangle \quad \cdot^{\mathcal{I}}(P) = \langle P^{\mathcal{I}} \rangle \quad \cdot^{\mathcal{I}}(U) = \langle U^{\mathcal{I}} \rangle.
\]
\( (U^\mathcal{I}) \subseteq \Delta^\mathcal{I}_O \times \Delta^\mathcal{I}_V \), whereas each literal \( \mathcal{l}^\mathcal{I} = \text{val}(\mathcal{l}) \), each datatype \( T^\mathcal{I}_i = \text{val}(T_i) \), and \( \top^\mathcal{I} = \Delta^\mathcal{I}_V \). The semantics of expressions:

\[ \{(\exists R)^\mathcal{I} = \{o \mid \exists o'. (o, o') \in R^\mathcal{I}\} \}, \{(P ^{-})^\mathcal{I} = \{(o', o) \mid (o', o) \in P^\mathcal{I}\}\}, \]

\[ \{(\delta (U))^\mathcal{I} = \{o \mid \exists v. (o, v) \in U^\mathcal{I}\} \}, \{(\rho (U))^\mathcal{I} = \{v \mid \exists o. (o, v) \in U^\mathcal{I}\}\}. \]

Contrary to OWL, \( DL-Lite_{\mathcal{A}} \) (and its implementation) adopts the *unique name assumption*, meaning that for every interpretation \( \mathcal{I} \) and distinct individuals or values \( c_1, c_2 \), we have that \( c^\mathcal{I}_1 \neq c^\mathcal{I}_2 \) (which is the norm in the database setting).

As for other DLs and OWL species, \( \mathcal{I} \) satisfies \( \alpha_1 \sqsubseteq \alpha_2 \) if \( \alpha^\mathcal{I}_1 \subseteq \alpha^\mathcal{I}_2 \), it satisfies \( (\text{disj} \alpha_1 \alpha_2) \) if \( \alpha^\mathcal{I}_1 \cap \alpha^\mathcal{I}_2 = \emptyset \), and it satisfies \( (\text{funct} S) \) if \( S^\mathcal{I} \) is a function (that is, if \( (o, z_1) \in S^\mathcal{I} \) and \( (o, z_2) \in S^\mathcal{I} \), then \( z_1 = z_2 \)).

\( \mathcal{I} \) satisfies \( A(a) \) if \( a^\mathcal{I} \in A^\mathcal{I} \), it satisfies \( P(a, a') \) if \( (a^\mathcal{I}, a'^\mathcal{I}) \in P^\mathcal{I} \), and it satisfies \( U(a, l) \) if \( (a^\mathcal{I}, \text{val}(l)) \in U^\mathcal{I} \).

Mappings

Here, a few definitions and an example is included; the chapter literature contains further technical details and more examples of mappings.

**Definition \( \PageIndex{1} \)**

*(Mapping assertion between a database and a TBox).* A mapping assertion between a database \( \mathcal{D} \) and a TBox \( \mathcal{T} \) has the form \( \Phi \rightsquigarrow \Psi \) where

- \( \Phi \) is an arbitrary SQL query of arity \( n > 0 \) over \( \mathcal{D} \);
- \( \Psi \) is a conjunctive query over \( \mathcal{T} \) of arity \( n' > 0 \) without non-distinguished variables, possibly involving variable terms.

**Definition \( \PageIndex{2} \)**

*(Mapping assertion in \( \mathcal{M} \) in an OBDA system).* A mapping assertion between a database \( \mathcal{D} \) and a TBox \( \mathcal{T} \) in \( \mathcal{M} \) has the form \( \Phi \overset{\text{overrightarrow}{x}}{\rightsquigarrow} \Psi(\overrightarrow{t}, \overrightarrow{y}) \) where

- \( \Phi \) is an arbitrary SQL query of arity \( n > 0 \) over \( \mathcal{D} \);
- \( \Psi \) is a conjunctive query over \( \mathcal{T} \) of arity \( n' > 0 \) without non-distinguished variables;
- \( \overrightarrow{x}, \overrightarrow{y} \) are variables with \( \overrightarrow{y} \subseteq \overrightarrow{x} \).
• \(\langle\overrightarrow{t}\rangle\) are variable terms of the form \(\langle f(\overrightarrow{z})\rangle\), with \(\langle f\in\Lambda\rangle\) and \(\langle\overrightarrow{z}\subseteq\overrightarrow{x}\rangle\).

Concerning the the semantics of mappings, intuitively: \(\langle\mathcal{I}\rangle\) satisfies \(\langle\Phi\rightsquigarrow\Psi\rangle\) with respect to \(\langle\mathcal{D}\rangle\) if all facts obtained by evaluating \(\langle\Phi\rangle\) over \(\langle\mathcal{D}\rangle\) and then propagating answers to \(\langle\Psi\rangle\), hold in \(\langle\mathcal{I}\rangle\).

Definition \(\PageIndex{3}\)

(Satisfaction of a mapping assertion with respect to a database). An interpretation \(\langle\mathcal{I}\rangle\) satisfies a mapping assertion \(\langle\Phi(\overrightarrow{x})\Psi(\overrightarrow{t},\overrightarrow{y})\rangle\) in \(\langle\mathcal{M}\rangle\) with respect to a database \(\langle\mathcal{D}\rangle\), if for each tuple of values \(\langle\overrightarrow{v}\rangle\in\text{Eval}(\langle\Phi\rangle,\langle\mathcal{D}\rangle)\), and for each ground atom in \(\langle\Psi[\overrightarrow{x}/\overrightarrow{v}]\rangle\) we have that:

• If the ground atom is \(\langle A(s)\rangle\), then \(\langle s^\mathcal{I}\in A^\mathcal{I}\rangle\) ;
• If the ground atom is \(\langle P(s_{1}, s_{2})\rangle\), then \(\langle (s^\mathcal{I}_{1}, s^\mathcal{I}_{2})\in P^\mathcal{I}\rangle\)

Note \(\PageIndex{1}\)

\(\langle\text{Eval}(\langle\Phi\rangle, \langle\mathcal{D}\rangle)\rangle\) denotes the result of evaluating \(\langle\Phi\rangle\) over \(\langle\mathcal{D}\rangle\). \(\langle\Psi[\overrightarrow{x}/\overrightarrow{v}]\rangle\) denotes \(\langle\Psi\rangle\) where each \(\langle\overrightarrow{x}[i]\rangle\) is substituted with \(\langle\overrightarrow{v}[i]\rangle\)

An example is shown in Figure 8.4.1 with the OBDA plugin for Protégé. There is an ontology that happens to have a class PromiscuousBacterium, among other things, and a relational database (HGT) with several tables, such as organisme and flexcount. Now we have to link the two with a mapping, which means (i) constructing a database query such that it retrieves only the promiscuous bacteria, and (ii) solving the ’impedance mismatch’ (recollect Chapter 7) with a functor so that the values returned by the database query become objects in the ontology, which is what getPromBact does. Informally, the functor can be considered as a URI building mechanism for individuals in the ontology taken from the database (theoretically, they are skolem functions).

Query Answering

Recall the outline of the ADOLENA ontology from the exercise and Figure 7.7.7. A query could be “retrieve the devices that ameliorate paraplegia”

\[\text{q}(x) \leftarrow \text{Device}(x), \text{ameliorates}(x,y), \text{Paraplegia}(y)\]

For this to work, we have to introduce three aspects. First, we need a computer-processable serialization of the query, a notion of what kind of queries we can pose, and a way how it will do the answering. The query language is SPARQL (see footnote 4). The kind of queries are (unions of) conjunctive queries. A conjunctive query (CQ) \(\langle q\rangle\) over an ontology \(\langle\mathcal{O}\rangle\) is an expression of the form \(\langle q(\overrightarrow{t})\rangle\) over \(\langle\mathcal{O}\rangle\)

\[\langle q(\overrightarrow{t})\rangle \leftarrow \exists\overrightarrow{y}.\text{conj}(\overrightarrow{t},\overrightarrow{y})\]

the head, \(\langle\text{conj}(\overrightarrow{t},\overrightarrow{y})\rangle\) the body, the variables in \(\langle\overrightarrow{t}\rangle\) are distinguished variables and \(\langle\overrightarrow{y}\rangle\) the non-distinguished variables, and where \(\langle\text{conj}(\overrightarrow{t},\overrightarrow{y})\rangle\) is a conjunction
of atoms of the form \( \langle D(z), S(z, z') \rangle, z = z' \rangle \), where \( \langle D \rangle \) denotes a class or a datatype, \( \langle S \rangle \) an object property or data property in \( \langle \mathcal{O} \rangle \), and \( \langle z, z' \rangle \) are individuals or literals in \( \langle \mathcal{O} \rangle \) or variables in \( \langle \text{overrightarrow}(\vec{x}) \rangle \) or \( \langle \text{overrightarrow}(\vec{y}) \rangle \). Given an interpretation \( \langle \mathcal{I} \rangle = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle \), then \( q^\mathcal{I} \) is the set of tuples of \( \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle \) that, when assigned to the variables \( \langle \text{overrightarrow}(\vec{x}) \rangle \), make the formula \( \exists \vec{y} \cdot \text{conj}(\vec{x}, \vec{y}) \) true in \( \langle \mathcal{I} \rangle \). The set \( \text{cert}(q, \langle \mathcal{O} \rangle) \) —certain answers to \( q \) over \( \langle \mathcal{O} \rangle \) —is the set of tuples \( \langle \text{overrightarrow}(\vec{a}) \rangle \) of individuals or literals appearing in \( \langle \mathcal{O} \rangle \) such that \( \langle \text{overrightarrow}(\vec{a}) \rangle \cap q^\mathcal{I} \) \( \forall \vec{a} \in \langle \mathcal{I} \rangle \), for every model \( \langle \mathcal{I} \rangle \) of \( \langle \mathcal{O} \rangle \).

Regarding query answering, consider Figure 8.4.2, where \( \langle q \rangle \) is our query, \( \langle \mathcal{O} \rangle \) the TBox (ontology or formal conceptual data model), and \( \langle \mathcal{A} \rangle \) the ABox (our instances, practically stored in the relational database). Somehow, this combines to produce the answers, using all components. First, there is a "reformulation" (or: rewriting) step, which computes the perfect reformulation (rewriting), \( \langle q_{pr} \rangle \), of the original query \( \langle q \rangle \) using the inclusion assertions of \( \langle \mathcal{O} \rangle \) so that we have a union of conjunctive queries. That is, it uses the knowledge of the ontology to come up with the 'real' query. For instance, recollect Figure 7.7.7 about the ontology of abilities and disabilities, then a query "retrieve all Devices that assistWith UpperLimbMobility", i.e.,

\[
q(x) ::- \text{Device}(x), \text{assistsWith}(x,y), \text{UpperLimbMobility}(y)
\]

or, in SPARQL notation:

```sparql
SELECT $device
WHERE { $device rdf:type :Device. $device :assistsWith $y. $y rdf:type :UpperLimbMobility}
```

will traverse the hierarchy of devices until it finds those devices that have an object property declared with as range \text{UpperLimbMobility}. In this case, this is \text{MotorizedWheelchair}, hence, the 'real' query concerns only the...
retrieval of the motorized wheelchairs, not first retrieving all devices and then making the selection.

Second, the “unfolding” step computes a new query, \( (q_{(unf)}) \), by using the (split version of) the mappings that link the terms of the ontology to queries over the database.

Third, the “evaluation” step delegates the evaluation of \( (q_{(unf)}) \) to the relational DBMS managing the data, which subsequently returns the answer.

![Diagram](https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_An_Introduction_to_Ontology_Engineering_(Keet)/09%3A_The_OntoWeb_Framework/8%3A_The_Top-Down_Approach_to_Query_Answering/8.4.2.png)

**Figure: 8.4.2: Intuition of the top-down approach to query answering.**

### Examples

Early trials and tribulations of trying to set up an OBDA system (and of which I know the details) are described in [KAGC08] and [CKN+10], which provided useful learning moments [CCKE+17]. The ontology in [KAGC08] practically was developed separately from the existing database, which caused some mismatches; or: the content in the ontology and database really should be compatible (as an aside: the query answering examples in the previous section were based on that system). This was much better aligned in the experiment described in [CKN+10]: the ontology was a reverse engineered physical (SQL) schema and dressed up with neater modeling, having following the steps described in Section 7.1: Relational Databases and Related ‘legacy’ KR. A downside with the knowledge of this particular scenario was that there was a considerable hierarchy, so the system ended up creating very big conjunctive queries, which slowed down the database side in actually answering the SQL query (albeit still faster than the original database). This has been optimized in the meantime, among other things [CCKE+17].

More recent examples from industry and with the ‘next generation’ OBDA, i.e., the extended Ontop setting, can be found in, e.g., [KHS+17], describing industry’s motivations, mappings, a rather large SQL query, and streaming (i.e., temporal) data, and there are several references in [CCKE+17] that describe applications and usage of Ontop. It is, perhaps, also worth mentioning that OBDA has seen many people involved since the early days in 2005, and by now has been shown to be robust, albeit not a trivial matter to understand and set up from a system’s development viewpoint. You can practice with this in the Chapter’s exercises, as all the code and software is freely available online.