3.2: Important Description Logics

There are very many DLs, of which some are used more often than others. In this section, we will first look at \(\mathcal{ALC}\), for it is typically one of the languages used in DL courses, a basis to add various language features to, and it is much easier for showing how the principles of tableau work for DLs. Subsequently, we list the more expressive \(\mathcal{SROIQ}\), and finally comment on leaner fragments that are computationally better behaved.

A Basic DL to Start With: \(\mathcal{ALC}\)

The DL language \(\mathcal{ALC}\)—which stands for Attributive Language with Concept negation—contains the following elements:

- **Concepts** denoting entity types/classes/unary predicates/universals, including top \(\top\) and bottom \(\bot\);
- **Roles** denoting relationships/associations/binary predicates/properties;
- **Constructors**: and \(\sqcap\), or \(\sqcup\), and not \(\neg\); quantifiers \(\forall\) and \(\exists\);
- **Complex concepts** using constructors: Let \(C\) and \(D\) be concept names, \(R\) a role name, then
  - \(\neg C, C\sqcap D,\) and \(C\sqcup D\) are concepts, and
  - \(\forall R.C\) and \(\exists R.C\) are concepts
- **Individuals**

Some examples that can be represented in \(\mathcal{ALC}\) are, respectively:
- Concepts (primitive, atomic); e.g., Book, Course

- Roles; e.g., enrolled, reads

- Complex concepts; e.g.,
  - \(\exists \text{enrolled.}(\text{Course} \cup \text{DegreeProgramme})\) (this is a primitive concept)
  - \(\exists \text{ParentOf.Person}\)
  - \(\text{Parent} \equiv (\text{Male} \cup \text{Female}) \cap \exists \text{ParentOf.Mammal} \cap \exists \text{caresFor.Mammal}\) (this is a defined concept)

- Individuals; e.g., Student(Andile), Mother(Katniss), \(\neg \text{Student(Katniss)}\), enrolled(Andile, COMP101)

As usual, the meaning is defined by the semantics of \(\mathcal{ALC}\), and it follows the same approach as we have seen for the other languages that have passed the revue (recollect FOL and model-theoretic semantics from "2.1: First Order Logic Syntax and Semantics"). First, there is a domain of interpretation, and an interpretation, where:

- Domain \(\Delta\) is a non-empty set of objects

- Interpretation: \(\cdot^\mathcal{I}\) is the interpretation function, domain \(\Delta^\mathcal{I}\)
  - \(\cdot^\mathcal{I}\) maps every concept name \(A\) to a subset \(A^\mathcal{I}\subseteq\Delta^\mathcal{I}\)
  - \(\cdot^\mathcal{I}\) maps every role name \(R\) to a subset \(R^\mathcal{I}\subseteq\Delta^\mathcal{I} \times \Delta^\mathcal{I}\)
  - \(\cdot^\mathcal{I}\) maps every individual name \(a\) to elements of \(\Delta^\mathcal{I}\): \(a^\mathcal{I} \in \Delta^\mathcal{I}\)

Note: \(\PageIndex{1}\)

\(\top^\mathcal{I} = \Delta^\mathcal{I}\) and \(\bot^\mathcal{I} = \emptyset\)

Using the typical notation where \(\text{C}\) and \(\text{D}\) are concepts, \(\text{R}\) a role, and \(\text{a}\) and \(\text{b}\) are individuals, then they have the following meaning, with on the left-hand side of the "\(\equiv\)" the syntax of \(\mathcal{ALC}\) under an interpretation and on the right-hand side its semantics:

- \(\neg C^\mathcal{I} = \Delta^\mathcal{I} \backslash C^\mathcal{I}\)

- \((C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}\)

- \((C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}\)
\[(\forall R.C)^{\mathcal{I}} = \{ x | (\forall y.R^{\mathcal{I}}(x,y) \to C^{\mathcal{I}}(y)) \}\]

\[(\exists R.C)^{\mathcal{I}} = \{ x | (\exists y.R^{\mathcal{I}}(x,y) \wedge C^{\mathcal{I}}(y)) \}\]

Observe that this list is a subset of those listed in Table 3.1.1, as there are fewer features in \(\mathcal{ALC}\) cf. \(\mathcal{SROIQ}\).

One also can specify the notion of satisfaction:

- An interpretation \(\mathcal{I}\) satisfies the statement \((C \sqsubseteq D)\) if \(C^{\mathcal{I}} \subseteq D^{\mathcal{I}}\)
- An interpretation \(\mathcal{I}\) satisfies the statement \((C \equiv D)\) if \(C^{\mathcal{I}} = D^{\mathcal{I}}\)
- \((C(a))\) is satisfied by \(\mathcal{I}\) if \(a^{\mathcal{I}} \in C^{\mathcal{I}}\)
- \((R(a, b))\) is satisfied by \(\mathcal{I}\) if \(\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}\)

An interpretation \(\mathcal{I}\) is a model of a knowledge base \(\mathcal{KB}\) if every axiom of \(\mathcal{KB}\) is satisfied by \(\mathcal{I}\).

Many DLs have been defined over the past 25 years and their complexity proved. For instance, one could add inverses to \(\mathcal{ALC}\), giving \(\mathcal{ALCI}\), or a hierarchy of roles, \(\mathcal{ALCH}\), or qualified cardinality restrictions; the appendix of the DL Handbook [BCM*08] has the full list of letters and the features they denote. You also may like to have a look at the DL Complexity Navigator. In the next chapter about OWL 2, we shall introduce a few more expressive languages, whereas ontology-based data access in Chapter 8 introduces DL-Lite that is less expressive than \(\mathcal{ALC}\).

The DL \(\mathcal{SROIQ}\)

In this section, we summarise the various features that have been introduced informally above to provide a comprehensive definition of DL syntax. Doing so yields the description logic called \(\mathcal{SROIQ}\), which is one of the most expressive DLs commonly considered today. It also largely agrees in expressivity with the ontology language OWL 2 DL, though there are some differences as will be discussed in Chapter 4.

Formally, every DL ontology is based on three finite sets of signature symbols: a set \(N_{I}\) of individual names, a set \(N_{C}\) of concept names and a set \(N_{R}\) of role names. Usually these sets are assumed to be fixed for some application and are therefore not mentioned explicitly. Now the set of \(\mathcal{SROIQ}\) role expressions \(\textbf{R}\) (over this signature) is defined by the following grammar:

\[\textbf{R} ::= U | N_{R} | N_{R}^{-}\]

where \(U\) is the universal role. Based on this, the set of \(\mathcal{SROIQ}\) concept expressions \(\textbf{C}\) is
defined as:
\[
\textbf{C} ::= \text{N}_{C} | (\textbf{C}\sqcap\textbf{C} ) | (\textbf{C}\sqcup\textbf{C} ) | \neg\textbf{C} | \top | \bot | \exists\textbf{R.C} | \forall\textbf{R.C} | \geqslant n\textbf{R.C} | \leqslant n\textbf{R.C} | \exists\textbf{R.}
\]
where \((n)\) is a non-negative integer. As usual, expressions like \((\textbf{C}\sqcap\textbf{C} )\) represent any expression of the form \((C\sqcap D)\) with \(C, D \in \textbf{C}\). It is common to omit parentheses if this cannot lead to confusion with expressions of different semantics. For example, parentheses do not matter for \((A\sqcup B\sqcup C)\) whereas the expressions \((A\sqcap B\sqcup C)\) and \((\exists R.A\sqcap B)\) are ambiguous.

Using the above sets of individual names, roles and concepts, the axioms of \(\mathcal{SROIQ}\) can be defined to be of the following basic forms:

ABox: \(\textbf{C} (\text{N}_{I} ) \textbf{R} (\text{N}_{I} ,\text{N}_{I} ) \text{N}_{I}\approx\\text{N}_{I} \text{N}_{I}\not\approx\text{N}_{I}\)

TBox: \(\textbf{C}\sqsubseteq\textbf{C} \textbf{C}\equiv\textbf{C}\)

RBox: \(\textbf{R}\sqsubseteq\textbf{R} \textbf{R}\equiv\textbf{R} \textbf{R}\circ\textbf{R}\sqsubseteq\textbf{R} \text{Disjoint(\textbf{R} ,\textbf{R} )}\)

with the intuitive meanings as explained in 3.1: DL Primer.

Roughly speaking, a \(\mathcal{SROIQ}\) ontology (or knowledge base) is simply a set of such axioms. To ensure the existence of reasoning algorithms that are correct and terminating, however, additional syntactic restrictions must be imposed on ontologies. These restrictions refer not to single axioms but to the structure of the ontology as a whole, hence they are called structural restrictions. The two such conditions relevant for \(\mathcal{SROIQ}\) are based on the notions of simplicity and regularity. Notably, both are automatically satisfied for ontologies that do not contain complex role inclusion axioms.

A role \(\textbf{R}\) in an ontology \(\mathcal{O}\) is called non-simple if some complex role inclusion axiom (i.e., one that uses role composition \((\circ)\)) in \(\mathcal{O}\) implies instances of \(\textbf{R}\); otherwise it is called simple. A more precise definition of the non-simple role expressions of the ontology \(\mathcal{O}\) is given by the following rules:

- if \(\mathcal{O}\) contains an axiom \(S\circ T\sqsubseteq R\), then \(R\) is non-simple,
- if \(R\) is non-simple, then its inverse \(R\{−\}\) is also non-simple,
- if \(R\) is non-simple and \(\mathcal{O}\) contains any of the axioms \(R\sqsubseteq S\), \(S\equiv R\) or \(R\equiv S\), then \(S\) is also non-simple.

All other roles are called simple. Now for a \(\mathcal{SROIQ}\) ontology it is required that the following axioms and concepts contain simple roles only:

Restricted axioms: \(\exists\text{Disjoint(\textbf{R} ,\textbf{R} )}\)

Restricted concept expressions: \(\exists\textbf{R.}\!\text{Self} | \!\geqslant n\textbf{R.}\!\textbf{C} | \!\leqslant n\textbf{C}\!\textbf{R.} \)
The other structural restriction that is relevant for $\mathcal{SROIQ}$ is called regularity and is concerned with RBox axioms only. Roughly speaking, the restriction ensures that cyclic dependencies between complex role inclusion axioms occur only in a limited form. For details, please see [HKS06]. For the introductory treatment in this paper, it suffices to note that regularity, just like simplicity, is a property of the ontology as a whole that cannot be checked for each axiom individually. An important practical consequence is that the union of two regular ontologies may no longer be regular. This must be taken into account when merging ontologies in practice.

The semantics of $\mathcal{SROIQ}$ is shown in Tables 3.1.1 and 3.1.2.

### Important Fragments of $\mathcal{SROIQ}$

Many different description logics have been introduced in the literature. Typically, they can be characterised by the types of constructors and axioms that they allow, which are often a subset of the constructors in $\mathcal{SROIQ}$. For example, the description logic $\mathcal{ALC}$ is the fragment of $\mathcal{SROIQ}$ that allows no RBox axioms and only $\sqcap$, $\sqcup$, $\neg$, $\exists$ and $\forall$ as its concept constructors. It is often considered the most basic DL. The extension of $\mathcal{ALC}$ with transitive roles is traditionally denoted by the letter $\mathcal{S}$. Some other letters used in DL names hint at a particular constructor, such as inverse roles $\mathcal{I}$, nominals $\mathcal{O}$, qualified number restrictions $\mathcal{Q}$, and role hierarchies (role inclusion axioms without composition) $\mathcal{H}$. So, for example, the DL named $\mathcal{ALCHIQ}$ extends $\mathcal{ALC}$ with role hierarchies, inverse roles and qualified number restrictions. The letter $\mathcal{R}$ most commonly refers to the presence of role inclusions, local reflexivity $\textbf{Self}$, and the universal role $\textbf{U}$, as well as the additional role characteristics of transitivity, symmetry, asymmetry, role disjointness, reflexivity, and irreflexivity. This naming scheme explains the name $\mathcal{SROIQ}$.

In recent years, fragments of DLs have been specifically developed in order to obtain favourable computational properties. For this purpose, $\mathcal{ALC}$ is already too large, since it only admits reasoning algorithms that run in worst-case exponential time. More light-weight DLs can be obtained by further restricting expressivity, while at the same time a number of additional $\mathcal{SROIQ}$ features can be added without losing the good computational properties. The three main approaches for obtaining light-weight DLs are $\mathcal{EL}$, $\mathcal{DLP}$ and $\mathcal{DL-Lite}$, which also correspond to language fragments OWL EL, OWL RL and OWL QL of the Web Ontology Language.

The $\mathcal{EL}$ family of description logics is characterised by allowing unlimited use of existential quantifiers and concept intersection. The original description logic $\mathcal{EL}$ allows only those features and $\top$ but no unions, complements or universal quantifiers, and no RBox axioms. Further extensions of this language are known as $\mathcal{EL}^+\text{ and }\mathcal{EL}^{++}$. The largest such extension allows the constructors $\neg$, $\top$, $\bot$, $\exists$, $\forall$, $\textbf{Self}$, nominals and the universal role, and it supports all types of axioms other than role symmetry, asymmetry and irreflexivity. Interestingly, all standard reasoning tasks for this DL can still be solved in worst-case polynomial time. One can even drop the structural restriction of regularity that is important for $\mathcal{SROIQ}$. $\mathcal{EL}$-type ontologies have been used to model large but light-weight ontologies that consist mainly of terminological data, in particular in the life sciences. A number of reasoners are specifically optimised for handling $\mathcal{EL}$-type ontologies, the most recent of which is the ELK reasoner for OWL 2 EL.

https://eng.libretexts.org/Bookshelves/Computer_Science/Book%3A_An_Introduction_to_Ontology_Engineering_(Keet)/3%3A…
Updated: Sun, 13 Oct 2019 16:29:42 GMT
Powered by
DLP is short for *Description Logic Programs* and comprises various DLs that are syntactically restricted in such a way that axioms could also be read as rules in first-order Horn logic without function symbols. Due to this, DLP-type logics can be considered as kinds of rule languages (hence the name OWL 2 RL) contained in DLs. To accomplish this, one has to allow different syntactic forms for subconcepts and superconcepts in concept inclusion axioms. We do not provide the details here. While DLs in general may require us to consider domain elements that are not denoted by individual names, for DLP one can always restrict attention to models in which all domain elements are denoted by individual names. This is why DLP is often used to augment databases (interpreted as sets of ABox axioms), e.g., in an implementation of OWL 2 RL in the Oracle 11g database management system.

DL-Lite is a family of DLs that is also used in combination with large data collections and existing databases, in particular to augment the expressivity of a query language that retrieves such data. This approach, known as Ontology Based Data Access, considers ontologies as a language for constructing *views* or *mapping rules* on top of existing data. The core feature of DL-Lite is that data access can be realised with standard query languages such as SQL that are not aware of the DL semantics. Ontological information is merely used in a query preprocessing step. Like DLP, DL-Lite requires different syntactic restrictions for subconcepts and superconcepts. It is the basis for the OWL 2 QL species of OWL ontology languages, and we will present its DL definition and its use in Chapter 8.

---

**Footnotes**


9. If \( (R=S^{-}) \) already is an inverse role, then \( (R^{-}) \) should be read as \( (S) \). We do not allow expressions like \( (S^{-^{-}}) \).

10. Whether the universal role \( (U) \) is simple or not is a matter of preference that does not affect the computational properties of the logic [RKH08b]. However, the universal role in OWL 2 is considered non-simple.