- [MPSP09] is the central document which introduces OWL 2 in functional style syntax.
- [PSM09] describes how the functional style syntax translates from and to the RDF syntax.
- [MCGH+09] specifies the different profiles of OWL 2.
- [SHK09] describes conformance conditions for OWL 2 and introduces the format of OWL 2 test cases which are provided along with the OWL 2 documents.
- [HKP+09] is a general introduction to OWL 2.

Exercises 4.1 to 4.3 were inspired by [RDH+04].

Chapter 5

OWL Formal Semantics

In Chapter 4 we introduced OWL syntactically, and have discussed intuitively how to derive logical inferences from OWL ontologies. This derivation of implicit knowledge is at the heart of logic-based semantics, and we give this a thorough and formal treatment in this chapter. We start with description logics in Section 5.1, which provide a logical view on OWL. In Section 5.2, we then present two equivalent ways of defining the formal semantics of OWL. In Section 5.3 we present the most successful algorithmic approach, the so-called tableaux method, for automated reasoning with OWL ontologies. In this chapter, the reader will benefit from some background in predicate logic, which can be found in Appendix C.

5.1 Description Logics

OWL DL can be identified with a decidable fragment of first-order predicate logic and thus OWL draws on the long history of philosophical and mathematical logic, which is a well-established and well-understood theory. As such, it is also in the tradition of logic-based artificial intelligence research, where the development of suitable knowledge representation formalisms plays an important part.

Historically, OWL DL can be traced back to so-called *semantic networks*, which can be used for the modeling of simple relationships between individuals and classes via roles, roughly comparable to RDFS. In the beginning, the meaning of such semantic networks was vague, which necessitated a formalization of their semantics. Eventually, this led to the development of *description logics* which we will deal with prominently in this chapter. OWL DL is essentially a description logic, which in turn can be understood as a fragment of first-order predicate logic.

Description logics have been designed in order to achieve favorable trade-offs between expressivity and scalability. Also, they are usually decidable and there exist efficient algorithms for reasoning with them.

To make this introduction more accessible, we will sometimes refrain from a complete treatment of OWL DL, and instead restrict our attention to sub-languages which suffice for conveying the key insights.

5.1.1 The Description Logic $\mathcal{ALC}$

By *description logics* we understand a family of logics for knowledge representation, derived from semantic networks and related to so-called *frame logics*. Description logics are usually fragments of first-order predicate logic and their development is usually driven by considerations concerning computational complexity: Given a complexity class, find a description logic which is as expressive as possible concerning its language constructs, but remains within the given complexity class. We will return to computational complexity later.

Researchers have developed a simple and useful notation for description logics which makes working with them much easier. We will use it in the following. We start by introducing the basic description logic $\mathcal{ALC}$.

5.1.1.1 Building Blocks of $\mathcal{ALC}$

Just as in OWL, the basic building blocks of $\mathcal{ALC}$ are classes, roles, and individuals, which can be put into relationships with each other. The expression

```
Professor(rudiStuder)
```

describes that the individual `rudiStuder` belongs to the class `Professor`. The expression

```
hasAffiliation(rudiStuder,aifb)
```

describes that `rudiStuder` is affiliated with `aifb`. The role `hasAffiliation` is an abstract role – we will discuss concrete roles later.

Subclass relations are expressed using the symbol $\sqsubseteq$. The expression

```
Professor ⊑ FacultyMember
```

says that `Professor` is a subclass of the class `FacultyMember`. Equivalence between classes is expressed using the symbol $\equiv$, e.g., as

```
Professor ≡ Prof
```


In order to express complex class relationships, $\mathcal{ALC}$ provides logical class constructors which we already know from OWL. The symbols for conjunction, disjunction, and negation are $\sqcap$, $\sqcup$, and $\neg$, respectively. The constructors can be nested arbitrarily, as in the following example.

```
Professor ⊑ (Person ⊓ FacultyMember) ⊔ (Person ⊓ ¬PhDStudent)
```

These logical constructors correspond to class constructors we already know from the OWL RDF syntax, namely, `owl:intersectionOf`, `owl:unionOf`, and `owl:complementOf`, respectively. The example just given corresponds to that from Fig. 4.11.

Complex classes can also be defined by using quantifiers, which correspond to role restrictions in OWL. If R is a role and C a class expression, then $\forall R.C$ and $\exists R.C$ are also class expressions.

The statement

```
Exam ⊑ ∀hasExaminer.Professor
```

states that all examiners of an exam must be professors, and corresponds to the example from page 127 using `owl:allValuesFrom`. The statement

```
Exam ⊑ ∃hasExaminer.Professor
```

says that every exam must have at least one examiner who is a professor, and corresponds to the example using `owl:someValuesFrom` from page 127.

Quantifiers and logical constructors can be nested arbitrarily.

5.1.1.2 Modeling Part of OWL in $\mathcal{ALC}$

We have already seen that many OWL DL language constructs can be expressed directly in $\mathcal{ALC}$. Some others can be expressed indirectly, as we will now demonstrate.

The empty class `owl:Nothing`, denoted in $\mathcal{ALC}$ using the symbol $\bot$, can be expressed by

$$\bot \equiv C \sqcap \neg C,$$

where C is some arbitrary class. Analogously, the class $\top$, which corresponds to `owl:Thing`, can be expressed by

$$\top \equiv C \sqcup \neg C,$$

or equivalently by

$$\top \equiv \neg \bot.$$

Disjointness of two classes C and D can be expressed using

$$C \sqcap D \sqsubseteq \bot,$$

or equivalently by

$$C \sqsubseteq \neg D,$$

corresponding to `owl:disjointWith`.

Domain and range of roles can also be expressed: The expression

$$\top \sqsubseteq \forall R.C$$

states that C is the `rdfs:range` of R, and the expression

$$\exists R.\top \sqsubseteq C$$

states that C is the `rdfs:domain` of R.

5.1.1.3 Formal Syntax of $\mathcal{ALC}$

Formally, the following syntax rules define $\mathcal{ALC}$. We first define how complex classes are constructed. Let A be an *atomic class*, i.e. a class name, and let R be an (abstract) role. Then *class expressions* C, D are constructed using the following rule.

$$C, D ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$$

Another common name for class expressions in description logics is "concept" or "concept expression" but we will adhere to the terminology that is used in OWL. Statements in $\mathcal{ALC}$ – and in other description logics – are divided into two groups, namely into *TBox* statements and *ABox* statements. The TBox is considered to contain terminological (or schema) knowledge, while the ABox contains assertional knowledge about instances (i.e. individuals). Remember that we distinguished between these two types of knowledge already in the case of RDFS (cf. the example in Section 2.6). Separating TBox and ABox becomes a bit academic when considering certain more expressive description logics, but it is still useful, and the distinction is well-defined for $\mathcal{ALC}$. Statements of either kind are often called *axioms* in description logics.[1]

Formally, a *TBox* consists of statements of the form $C \equiv D$ or $C \sqsubseteq D$, where C and D are class expressions. Statements $C \sqsubseteq D$ are called *(general) class inclusion axioms*. An *ABox* consists of statements of the form $C(a)$ and $R(a, b)$, where C is a class expression, R is a role, and a, b are individuals. An *$\mathcal{ALC}$ knowledge base* consists of an ABox and a TBox.

[1] The term "formula" would be more accurate than "axiom" in cases where a statement is not required to be true, but "axiom" is widely used in the literature.

5.1.2 OWL DL as Description Logic

We have already seen that the following OWL DL language constructs can be represented in $\mathcal{ALC}$:

- classes, roles, and individuals
- class membership and role instances
- `owl:Thing` and `owl:Nothing`
- class inclusion, class equivalence, and class disjointness
- conjunction, disjunction, and negation of classes
- role restrictions using `owl:allValuesFrom` and `owl:someValuesFrom`
- `rdfs:domain` and `rdfs:range`

The other OWL DL language constructs cannot be expressed in $\mathcal{ALC}$. Instead, we need to extend $\mathcal{ALC}$ to the description logic $\mathcal{SHOIN}$(D), which encompasses $\mathcal{ALC}$ and also provides further expressive means. We will present them in the following.

5.1.2.1 Class Constructors and Relationships

Closed class expressions using `owl:oneOf` can be expressed in $\mathcal{SHOIN}$(D) as follows: The class containing exactly the individuals $a_1, \ldots, a_n$ is written as $\{a_1, \ldots, a_n\}$. When talking about description logics, closed classes are called *nominals*.[2]

We have already seen on page 129 that `owl:hasValue` can be expressed by making use of `owl:someValuesFrom` and `owl:oneOf`, i.e. `owl:hasValue` is expressible in $\mathcal{SHOIN}$(D).

$\mathcal{SHOIN}$(D) further provides cardinality restrictions via the following notation: The statement

$$\texttt{Exam} \sqsubseteq \leq 2\texttt{hasExaminer}$$

says that each exam has at most two examiners. More generally, we can express `owl:maxCardinality` via $\leq nR$, where n is a non-negative integer, and R is an (abstract) role. Likewise, `owl:minCardinality` is written using $\geq nR$. As already exemplified in Fig. 4.12, `owl:cardinality` can be expressed using the intersection of `owl:minCardinality` and `owl:maxCardinality`.

[2] To be precise, a nominal is a class which contains exactly one individual. Closed classes correspond to unions of nominals then.

5.1.2.2 Relationships Between Individuals

Equality of individuals a and b is expressed indirectly as $\{a\} \equiv \{b\}$ using nominals and class equivalence. Inequality of individuals a and b is expressed likewise by saying that the classes $\{a\}$ and $\{b\}$ are disjoint, i.e. by stating $\{a\} \sqcap \{b\} \sqsubseteq \bot$.

5.1.2.3 Role Constructors, Role Relationships, and Role Characteristics

The statement that R is a subrole of S is written as $R \sqsubseteq S$, and is called a *role inclusion axiom*. Equivalence between these roles is written as $R \equiv S$. The inverse role to R is denoted by R^-, i.e. $S \equiv R^-$ states that S is the inverse of R. In $\mathcal{SHOIN}$(D), inverse role descriptions may be used in all the places where roles may occur, basically as in OWL 2.

Transitivity of a role R is stated as Trans(R). Symmetry of R can be declared indirectly using $R \equiv R^-$. Functionality of R is stated as $\top \sqsubseteq \leq 1R$ while inverse functionality of R is stated as $\top \sqsubseteq \leq 1R^-$.

5.1.2.4 Datatypes

$\mathcal{SHOIN}$(D) allows the use of data values, i.e. of elements of datatypes, in the second argument of concrete roles. It is also possible to form closed classes using such data values. This straightforward use of datatypes does not have any significant impact on the logical underpinnings, so we will not go into more detail here.

There exist more powerful uses of datatypes, known as *concrete domains*, in the theory of description logics. But concrete domains are not part of the OWL standard, so we only refer the interested reader to the literature given in Section 5.6.

5.1.2.5 $\mathcal{SHOIN}$(D) and OWL DL

Let us summarize the expressive means available in $\mathcal{SHOIN}$(D), as they cover OWL DL. We have

- all language constructs from $\mathcal{ALC}$,
- equality and inequality between individuals,
- closed classes (i.e. disjunctions of nominals),
- cardinality restrictions,
- role inclusion axioms and role equivalences (i.e. role hierarchies),
- inverse roles,
- transitivity, symmetry, functionality, and inverse functionality of roles,
- datatypes.

5.1.3 Naming Description Logics – and How They Relate to the OWL Sublanguages

We have already introduced and used some of the strange names description logics have, such as $\mathcal{ALC}$ or $\mathcal{SHOIN}$(D). The terminology behind these names is in fact systematic: the letters describe which language constructs are allowed in a particular description logic. $\mathcal{ALC}$ is short for *Attributive Language with Complement*, and has its name for historical reasons. $\mathcal{ALC}$ is considered to be the most fundamental description logic,[3] and is usually the starting point for theoretical investigations. We have formally defined $\mathcal{ALC}$ in Section 5.1.1.3.

Expressive means beyond $\mathcal{ALC}$ are now indicated by certain letters. The following explains $\mathcal{SHOIN}$(D).

- $\mathcal{S}$ stands for $\mathcal{ALC}$ plus role transitivity.
- $\mathcal{H}$ stands for role hierarchies, i.e. for role inclusion axioms.
- $\mathcal{O}$ stands for nominals, i.e. for closed classes with one element.
- $\mathcal{I}$ stands for inverse roles.
- $\mathcal{N}$ stands for cardinality restrictions.
- D stands for datatypes.

We also give the letters for some other language constructs which are of particular importance, and will explain them below.

- $\mathcal{F}$ stands for role functionality.
- $\mathcal{Q}$ stands for qualified cardinality restrictions.
- $\mathcal{R}$ stands for generalized role inclusion axioms.
- $\mathcal{E}$ stands for the use of existential role restrictions.

[3] $\mathcal{ALC}$ is often said to be *Boolean closed*, which means that conjunction, disjunction, negation and both quantifiers can be used without any restrictions. Description logics without this feature are called *sub-Boolean*.

5.1.3.1 Role Functionality

We have already said that OWL DL corresponds to $\mathcal{SHOIN}$(D). But functionality of roles can be declared in OWL DL, so why didn't we say that it corresponds to $\mathcal{SHOINF}$(D)? The reason is that redundant letters are usually left out. We have seen on page 164 that functionality can be expressed by means of cardinality restrictions, so functionality is implicit in $\mathcal{SHOIN}$(D), i.e. the letter $\mathcal{F}$ is omitted. Likewise, there is no letter for inverse functionality simply because it can be expressed using cardinality restrictions and inverse roles. Likewise, symmetry of roles can be expressed using inverse roles and role hierarchies.

So why do we need the letter $\mathcal{F}$ at all? Because having description logics with functionality but without, e.g., cardinality restrictions can be meaningful. Indeed, OWL Lite corresponds to the description logic $\mathcal{SHIF}$(D).

5.1.3.2 Qualified Cardinality Restrictions

Qualified cardinality restrictions are a generalization of the cardinality restrictions which we already know from $\mathcal{SHOIN}$. They allow us to make declarations like $\leq nR.C$ and $\geq nR.C$ which are similar to $\leq nR$ and $\geq nR$ (sometimes called *unqualified* cardinality restrictions) but furthermore allow us to specify to which class the second arguments in the role R belong – we have already encountered them in our discussion of OWL 2 in Section 4.3.1.6. This usage of qualified cardinality restrictions is thus analogous to the role restrictions $\forall R.C$ or $\exists R.C$.

Qualified cardinality restrictions encompass unqualified ones: $\geq nR$, for example, can be expressed using $\geq nR.\top$. It is also a fact that extending from unqualified to qualified cardinality restrictions hardly makes a difference in terms of theory, algorithms, or system runtimes. Description logic literature is thus usually concerned with $\mathcal{SHIQ}$ or $\mathcal{SHOIQ}$ rather than $\mathcal{SHIN}$ or $\mathcal{SHOIN}$.

5.1.3.3 Generalized Role Inclusions

We have already encountered generalized role inclusions in our discussion of OWL 2 in Section 4.3.1.5. The notation used for description logics is $R_1 \circ \cdots \circ R_n \sqsubseteq R$, meaning that the concatenation of $R_1, \ldots, R_n$ is a subrole of R. A typical example of this would be

$$\texttt{hasParent} \circ \texttt{hasBrother} \sqsubseteq \texttt{hasUncle}.$$

OWL 2 DL is essentially the description logic $\mathcal{SROIQ}$(D). Note that generalized role inclusions encompass role hierarchies, so that $\mathcal{SROIQ}$(D) contains $\mathcal{SHOIN}$(D), i.e. OWL 2 DL contains OWL DL.

OWL Full	is not a description logic
OWL DL	$\mathcal{SHOIN}$(D)
OWL Lite	$\mathcal{SHIF}$(D)
OWL 2 Full	is not a description logic
OWL 2 DL	$\mathcal{SROIQ}$(D)
OWL 2 EL	$\mathcal{EL}^{++}$
OWL 2 QL	DL-Lite
OWL 2 RL	DLP

FIGURE 5.1: Correspondence between OWL variants and description logics

5.1.3.4 Existential Role Restrictions

Since existential role restrictions are contained in $\mathcal{ALC}$, this symbol is only useful when discussing sub-Boolean description logics which are *properly* contained in $\mathcal{ALC}$. This is the case for the description logics corresponding to some of the tractable profiles of OWL 2, as discussed in Section 4.3.2: The description logic $\mathcal{EL}$ allows conjunction and existential role restrictions.[4] $\mathcal{EL}^{++}$ additionally allows generalized role inclusions and nominals. It corresponds to the OWL 2 EL profile from Section 4.3.2.1. The tractable fragment DL-Lite imposes more complicated restrictions on the use of language constructs, and we will not treat it in more detail here. It corresponds to the OWL 2 QL profile from Section 4.3.2.2. The OWL 2 RL profile from Section 4.3.2.3 corresponds to a naive intersection between $\mathcal{SROIQ}$ and datalog (see Section 6.2) and is very closely related to so-called Description Logic Programs (DLP). DLP is also a tractable fragment of $\mathcal{SROIQ}$, but we refrain from covering it in more detail here: we will have much more to say about OWL and Rules in Chapter 6.

5.1.3.5 OWL Sublanguages and Description Logics

The mentioned letters for describing description logics have to be taken carefully, since minor modifications are imposed in some cases. It is therefore necessary to revert to the formal definitions when details matter. We have given a formal definition of $\mathcal{ALC}$ in Section 5.1.1.3 above, and will give the formal definition of $\mathcal{SROIQ}$ and $\mathcal{SHIQ}$ in Section 5.1.4 below.

We summarize the relationships between different versions and sublanguages of OWL and description logics in Fig. 5.1.

[4] The letter $\mathcal{L}$ does not really carry a specific meaning.

5.1.4 Formal Syntax of $\mathcal{SROIQ}$

We will now formally define the complete syntax of the $\mathcal{SROIQ}$ description logic. By doing this, we will encounter some details which we have not mentioned so far in our rather intuitive treatment. The definition we will give is one of several possible logically equivalent definitions. It is the one most convenient for the rest of our treatment in this chapter. Its formal semantics will be presented in Section 5.2.

For $\mathcal{SROIQ}$, it is customary and convenient to distinguish between RBox, for roles, TBox, for terminological knowledge, and ABox, for assertional knowledge.

5.1.4.1 $\mathcal{SROIQ}$ RBoxes

A *$\mathcal{SROIQ}$ RBox* is based on a set $\mathbf{R}$ of *atomic roles*, which contains all role names, all inverses of role names (i.e. R^- for any role name R), and the *universal role* U. The universal role is something like the $\top$ element for roles: It is a superrole of all roles and all inverse roles, and can intuitively be understood as relating all possible pairs of individuals. It is the top abstract role which we have already encountered in Section 4.3.1.3.

A *generalized role inclusion axiom* is a statement of the form $S_1 \circ \cdots \circ S_n \sqsubseteq R$, and a set of such axioms is called a *generalized role hierarchy*. Such a role hierarchy is called *regular* if there exists a strict partial order[5] $\prec$ on $\mathbf{R}$, such that the following hold:

- $S \prec R$ if and only if $S^- \prec R$
- every role inclusion axiom is of one of the forms

$$R \circ R \sqsubseteq R, \qquad R^- \sqsubseteq R, \qquad S_1 \circ \cdots \circ S_n \sqsubseteq R,$$
$$R \circ S_1 \circ \cdots \circ S_n \sqsubseteq R, \qquad S_1 \circ \cdots \circ S_n \circ R \sqsubseteq R$$

  such that R is a non-inverse role name, and $S_i \prec R$ for $i = 1, \ldots, n$.

Regularity is a way to restrict the occurrence of cycles in generalized role hierarchies. It needs to be imposed in order to guarantee decidability of $\mathcal{SROIQ}$.

[5] A *partial order* $\leq$ on a set X satisfies the following conditions for all $x, y, z \in X$: $x \leq x$; if $x \leq y$ and $y \leq x$, then $x = y$; and if $x \leq y$ and $y \leq z$, then $x \leq z$. If $\leq$ is a partial order, then we can define a *strict partial order* $<$ by setting $x < y$ if and only if $x \leq y$ and $x \neq y$.

We give an example of a role hierarchy which is *not* regular:

$$\texttt{hasParent} \circ \texttt{hasHusband} \sqsubseteq \texttt{hasFather}$$
$$\texttt{hasFather} \sqsubseteq \texttt{hasParent}$$

This role hierarchy is not regular because regularity would enforce both $\texttt{hasParent} \prec \texttt{hasFather}$ and $\texttt{hasFather} \prec \texttt{hasParent}$, which is impossible because $\prec$ must be strict.

Note that regular role hierarchies must not contain role equivalences: If we had $R \sqsubseteq S$ and $S \sqsubseteq R$, then regularity would enforce $R \prec S$ and $S \prec R$, which is impossible because $\prec$ must be strict. Formally, however, this restriction is not severe, since it basically means that we do not allow roles to have synonyms, i.e. if a knowledge base would contain two roles S and R which are equivalent, then we could simply replace all occurrences of S by R without losing any substantial information.

We now turn to the notion of *simple* role, which is also needed in order to guarantee decidability. Given a role hierarchy, the set of simple roles of this hierarchy is defined inductively, as follows.

- If a role does not occur on the right-hand side of a role inclusion axiom – and neither does the inverse of this role –, then it is simple.
- The inverse of a simple role is simple.
- If a role R occurs only on the right-hand side of role inclusion axioms of the form $S \sqsubseteq R$ with S being simple, then R is also simple.

Simplicity of a role essentially means that it does not occur on the right-hand side of a role inclusion axiom containing a role concatenation $\circ$.

To give an example, the set of simple roles of the role hierarchy $\{R \sqsubseteq R_1; R_1 \circ R_2 \sqsubseteq R_3; R_3 \sqsubseteq R_4\}$ is $\{R, R^-, R_1, R_1^-, R_2, R_2^-\}$.

Note that regular role hierarchies allow us to express transitivity ($R \circ R \sqsubseteq R$) and symmetry ($R^- \sqsubseteq R$). In $\mathcal{SROIQ}$, we additionally allow the explicit declaration of reflexivity of a role by $\mathrm{Ref}(R)$, of antisymmetry of a role by $\mathrm{Asy}(S)$, and of disjointness of two roles S_1 and S_2 by $\mathrm{Dis}(S_1, S_2)$. However, we have to impose the condition that S, S_1 and S_2 are simple in order to ascertain decidability. These declarations are called *role characteristics*.[6]

[6] In the description logic literature, role characteristics are often called *role assertions*.

Let us explain the intuition behind the three new $\mathcal{SROIQ}$ role characteristics which we have already encountered in Section 4.3.1.3. Reflexivity of a role means that everything is related to itself by this role; a typical example would be `isIdenticalTo`. Antisymmetry of a role R means that whenever a is related to b via R, then b is *not* related to a via R. Most roles are antisymmetric; an example would be the role `hasParent`. Disjointness of two roles means that they do not share any pair of instances. The two roles `hasParent` and `hasChild`, for example, would be disjoint, while `hasParent` and `hasFather` would not be disjoint.

A *$\mathcal{SROIQ}$ RBox* is the union of a set of role characteristics and a role hierarchy. A $\mathcal{SROIQ}$ RBox is *regular* if its role hierarchy is regular.

5.1.4.2 $\mathcal{SROIQ}$ Knowledge Bases

Given a $\mathcal{SROIQ}$ RBox $\mathcal{R}$, we now define the set of class expressions $\mathbf{C}$ inductively as follows.

- Every class name is a class expression.
- $\top$ and $\bot$ are class expressions.
- If C, D are class expressions, $R, S \in \mathbf{R}$ with S being simple, a is an individual, and n is a non-negative integer, then the following are class expressions:

$$\neg C \quad C \sqcap D \quad C \sqcup D \quad \{a\} \quad \forall R.C \quad \exists R.C$$
$$\exists S.\text{Self} \quad \leq n S.C \quad \geq n S.C$$

From our discussion of $\mathcal{SHOIN}(\mathrm{D})$, these language constructs are already familiar. An exception is the $\exists S.\text{Self}$ expression, which we have already encountered for OWL 2 in Section 4.3.1.7. Intuitively, an individual a is an instance of $\exists S.\text{Self}$ if a is related to itself via the S role. A typical example would be the class inclusion

$$\texttt{PersonCommittingSuicide} \sqsubseteq \exists \texttt{kills.Self}.$$

Concerning nominals, i.e. the use of the construct $\{a\}$, note that closed classes with more than one individual can be constructed using disjunction, i.e. $\{a_1, \ldots, a_n\}$ can be written as $\{a_1\} \sqcup \cdots \sqcup \{a_n\}$.

A *$\mathcal{SROIQ}$ TBox* is a set of *class inclusion axioms* of the form $C \sqsubseteq D$, where C and D are class expressions.

A *$\mathcal{SROIQ}$ ABox* is a set of *individual assignments* – of one of the forms $C(a)$, $R(a, b)$, or $\neg R(a, b)$, where $C \in \mathbf{C}$, $R \in \mathbf{R}$ and a, b are individuals.

Note that $\mathcal{SROIQ}$ allows negated role assignments $\neg R(a, b)$, which we also know from OWL 2 and Section 4.3.1.8. This allows us to state explicitly, e.g., that John is *not* the father of Mary, namely by $\neg$`hasFather(Mary,John)`.

A *$\mathcal{SROIQ}$ knowledge base* is the union of a regular RBox $\mathcal{R}$, an ABox, and a TBox for $\mathcal{R}$.

5.1.4.3 $\mathcal{SHIQ}$

The description logic $\mathcal{SHIQ}$ is of particular importance for research around OWL. From the perspective of computational complexity, which we discuss more closely in Section 5.3.5, $\mathcal{SHIQ}$ is not more complicated than $\mathcal{ALC}$. At the same time, only nominals are missing from $\mathcal{SHIQ}$ in order to encompass OWL DL.[7] $\mathcal{SHOIN}$, however, which is essentially OWL DL, is much more complex than $\mathcal{SHIQ}$, and $\mathcal{SROIQ}$ is even worse.

For research into reasoning issues around OWL, methods and algorithms are often first developed for $\mathcal{ALC}$, and then lifted to $\mathcal{SHIQ}$, before attempting $\mathcal{SHOIQ}$ or even $\mathcal{SROIQ}$. We do the same in Section 5.3, and require a formal definition of $\mathcal{SHIQ}$.

We define $\mathcal{SHIQ}$ by restricting $\mathcal{SROIQ}$. $\mathcal{SHIQ}$ RBoxes are $\mathcal{SROIQ}$ RBoxes restricted to axioms of the form $R \circ R \sqsubseteq R$ (written as Trans(R)), $R^- \sqsubseteq R$ (written as Sym(R)), and $S \sqsubseteq R$. Regularity does not need to be imposed for $\mathcal{SHIQ}$. Simplicity of roles is defined as for $\mathcal{SROIQ}$, but note that we can give a simpler definition of simplicity for $\mathcal{SHIQ}$: A role is simple unless it is transitive, its inverse is transitive, it has a transitive subrole, or its inverse has a transitive subrole. Note that we do not allow any of the additional role characteristics from $\mathcal{SROIQ}$.

$\mathcal{SHIQ}$ TBoxes are $\mathcal{SROIQ}$ TBoxes where Self and nominals of the form $\{a\}$, for a an individual, do not occur.

$\mathcal{SHIQ}$ ABoxes contain statements of the form $C(a)$, $R(a, b)$, or $a \neq b$, where $C \in \mathbf{C}$, $R \in \mathbf{R}$, and a, b are individuals, i.e. $\mathcal{SHIQ}$ Aboxes are $\mathcal{SROIQ}$ ABoxes where $\neg$ does not occur and where inequality of individuals may be explicitly stated. Note that there is no need to explicitly allow inequality of individuals in $\mathcal{SROIQ}$ ABoxes, since a statement like $a \neq b$ can be expressed in a $\mathcal{SROIQ}$ TBox using nominals as $\{a\} \sqcap \{b\} \sqsubseteq \bot$.

A *$\mathcal{SHIQ}$ knowledge base* is the union of a $\mathcal{SHIQ}$ RBox, a $\mathcal{SHIQ}$ TBox, and a $\mathcal{SHIQ}$ Abox.

For completeness, let us remark that the only difference between $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ is that nominals are allowed in class expressions.

[7] Datatypes are also missing, but they do not pose any particular difficulties to the theory.

5.2 Model-Theoretic Semantics of OWL

We now define formally the semantics of $\mathcal{SROIQ}$, i.e. for OWL 2 DL. Since $\mathcal{SROIQ}$ encompasses $\mathcal{SHOIN}$, this also means that we essentially define the formal semantics of OWL DL.

We present the semantics in two versions, which are equivalent. In Section 5.2.1 we give the extensional semantics, sometimes also called the direct model-theoretic semantics. In Section 5.2.2, we define the semantics by a translation into first-order predicate logic.

5.2.1 Extensional Semantics of $\mathcal{SROIQ}$

The direct model-theoretic semantics which we now define is similar to the model-theoretic semantics of RDF(S) given in Chapter 3. We will make remarks about similarities and differences at the appropriate places.

5.2.1.1 Interpreting Individuals, Classes, and Roles

As for RDF(S), we first need to fix notation for the vocabulary used. We assume

- a set $\mathbf{I}$ of symbols for individuals,
- a set $\mathbf{C}$ of symbols for class names, and
- a set $\mathbf{R}$ of symbols for roles.

There is a significant difference from the situation for RDF(S) (and OWL Full): The sets $\mathbf{I}$, $\mathbf{C}$, and $\mathbf{R}$ must be mutually disjoint. This means that we enforce type separation as discussed for OWL DL on page 139. OWL 2 punning as in Section 4.3.1.1 is not needed, although this would not change the theory. We avoid the issue of punning here simply for convenience.

We next define the notion of $\mathcal{SROIQ}$ interpretation. As for RDF(S), we start with a set of entities, which can be thought of as resources, individuals, or single objects. We denote this set, called the *domain of the interpretation*, by Δ. We now declare how individuals, class names, and roles are interpreted, namely, by means of the functions

- $\mathrm{I}_\mathbf{I}$, which maps individuals to elements of the domain: $\mathrm{I}_\mathbf{I} : \mathbf{I} \to \Delta$,
- $\mathrm{I}_\mathbf{C}$, which maps class names to subsets of the domain: $\mathrm{I}_\mathbf{C} : \mathbf{C} \to 2^\Delta$ (the *class extension*), and
- $\mathrm{I}_\mathbf{R}$, which maps roles to binary relations on the domain, i.e. to sets of pairs of domain elements: $\mathrm{I}_\mathbf{R} : \mathbf{R} \to 2^{\Delta \times \Delta}$ (the *property extension*).

There are many choices possible, which we do not further restrict: The set Δ may be arbitrary, and how exactly the functions $\mathrm{I}_\mathbf{I}$, $\mathrm{I}_\mathbf{C}$, and $\mathrm{I}_\mathbf{R}$ assign their values also bears a lot of freedom.

We note that we do not map class names and role names to single elements as done in RDF(S). The function $\mathrm{I}_\mathbf{C}$, however, could be understood as the concatenation of the functions I_S and I_CEXT from an RDF(S) interpretation. Likewise, $\mathrm{I}_\mathbf{R}$ could be understood as the concatenation of the functions I_S and I_EXT. Figure 5.2 graphically depicts a DL interpretation.

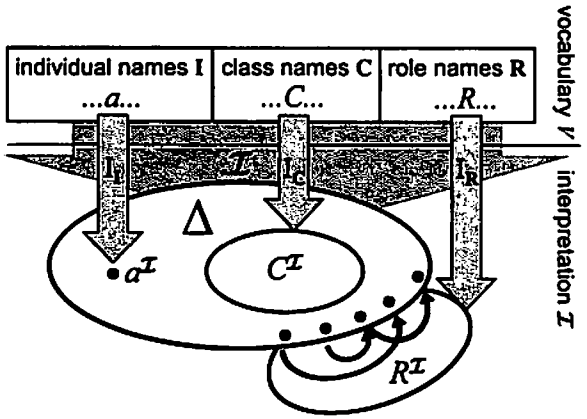


FIGURE 5.2: Schematic representation of a DL interpretation

We next define an interpretation function $\cdot^\mathcal{I}$, which lifts the interpretation of individuals, class names, and role names just given to complex class and role expressions.

- We set $\top^\mathcal{I} = \Delta$ and $\bot^\mathcal{I} = \emptyset$.
- $\neg C$ describes those things which are not in C, i.e. $(\neg C)^\mathcal{I} = \Delta \setminus C^\mathcal{I}$.
- $C \sqcap D$ describes those things which are both in C and in D, i.e. $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$.
- $C \sqcup D$ describes those things which are in C or in D, i.e. $(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$.
- $\exists R.C$ describes those things which are connected via R with something in C, i.e. $(\exists R.C)^\mathcal{I} = \{x \mid \text{there is some } y \text{ with } (x,y) \in R^\mathcal{I} \text{ and } y \in C^\mathcal{I}\}$.
- $\forall R.C$ describes those things x for which every y which connects from x via a role R is in the class C, i.e. $(\forall R.C)^\mathcal{I} = \{x \mid \text{for all } y \text{ with } (x,y) \in R^\mathcal{I} \text{ we have } y \in C^\mathcal{I}\}$.

- $\leq nR.C$ describes those things which are connected via R to at most n things in C, i.e.[8] $(\leq nR.C)^{\mathcal{I}} = \{x \mid \#\{(x,y) \in R^{\mathcal{I}} \mid y \in C^{\mathcal{I}}\} \leq n\}$.
- $\geq nR.C$ describes those things which are connected via R to at least n things in C, i.e. $(\geq nR.C)^{\mathcal{I}} = \{x \mid \#\{(x,y) \in R^{\mathcal{I}} \mid y \in C^{\mathcal{I}}\} \geq n\}$.
- $\{a\}$ describes the class containing only a, i.e. $\{a\}^{\mathcal{I}} = \{a^{\mathcal{I}}\}$.
- $\exists S.\text{Self}$ describes those things which are connected to themselves via S, i.e. $(\exists S.\text{Self})^{\mathcal{I}} = \{x \mid (x,x) \in S^{\mathcal{I}}\}$.
- For $R \in \mathbf{R}$, we set $(R^-)^{\mathcal{I}} = \{(b,a) \mid (a,b) \in R^{\mathcal{I}}\}$.
- For the universal role U, we set $U^{\mathcal{I}} = \Delta \times \Delta$.

Given a $\mathcal{SROIQ}$ knowledge base, an *interpretation* consists of a domain Δ and an interpretation function which satisfies the constraints just given. Note that due to the many degrees of freedom in choosing Δ and the functions I_I, I_C, and I_R, it is not necessary that interpretations are intuitively meaningful.

If we consider, for example, the knowledge base consisting of the axioms

$$\texttt{Professor} \sqsubseteq \texttt{FacultyMember}$$
$$\texttt{Professor(rudiStuder)}$$
$$\texttt{hasAffiliation(rudiStuder, aifb)}$$

then we could set

$$\Delta = \{a, b, \text{Ian}\}$$
$$I_I(\texttt{rudiStuder}) = \text{Ian}$$
$$I_I(\texttt{aifb}) = b$$
$$I_C(\texttt{Professor}) = \{a\}$$
$$I_C(\texttt{FacultyMember}) = \{a, b\}$$
$$I_R(\texttt{hasAffiliation}) = \{(a,b), (b, \text{Ian})\}$$

Intuitively, these settings are nonsense, but they nevertheless determine a valid interpretation.

Let us dwell for a bit on the point that the interpretation just given is intuitively nonsense. There are actually two aspects to this. The first is the choice of names for the elements in Δ, e.g., that rudiStuder is interpreted as Ian, which seems to be quite far-fetched. Note, however, that this aspect

[8] Recall from Appendix B that $\#A$ denotes the cardinality of the set A.

relates only to the names of elements in a set, while in logic we would usually abstract from concrete names, i.e. we would usually be able to rename things without compromising logical meanings. The second aspect is more severe, as it is structural: It is about the question whether the interpretation faithfully captures the relations between entities as stated in the knowledge base. This is not the case in this example: $I_I(\texttt{rudiStuder})$ is not contained in $I_C(\texttt{Professor})$, although the knowledge base states that it should. Similarly, $I_R(\texttt{hasAffiliation})$ does not contain the pair $(I_I(\texttt{rudiStuder}), I_I(\texttt{aifb}))$, although it should according to the knowledge base.

Interpretations which *do* make sense for a knowledge base in the structural manner just described are called *models* of the knowledge base, and we introduce them formally next. Note, however, that we ignore the first aspect, as commonly done in logic.

5.2.1.2 Interpreting Axioms

Models capture the structure of a knowledge base in the sense that they give a truthful representation of the axioms in terms of sets. Formally, models of a knowledge base are interpretations which satisfy additional constraints which are determined by the axioms of the knowledge base. The constraints are as follows: An interpretation $\mathcal{I}$ of a $\mathcal{SROIQ}$ knowledge base K is a *model* of K, written $\mathcal{I} \models K$, if the following hold.

- If $C(a) \in K$, then $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- If $R(a,b) \in K$, then $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.
- If $\neg R(a,b) \in K$, then $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$.
- If $C \sqsubseteq D \in K$, then $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.
- If $S \sqsubseteq R \in K$, then $S^{\mathcal{I}} \subseteq R^{\mathcal{I}}$.
- If $S_1 \circ \cdots \circ S_n \sqsubseteq R \in K$, then $\{(a_1, a_{n+1}) \in \Delta \times \Delta \mid \text{there are } a_2, \ldots, a_n \in \Delta \text{ such that } (a_i, a_{i+1}) \in S_i^{\mathcal{I}} \text{ for all } i = 1, \ldots, n\} \subseteq R^{\mathcal{I}}$.
- If $\text{Ref}(R) \in K$, then $\{(x,x) \mid x \in \Delta\} \subseteq R^{\mathcal{I}}$.
- If $\text{Asy}(R) \in K$, then $(x,y) \notin R^{\mathcal{I}}$ whenever $(y,x) \in R^{\mathcal{I}}$.
- If $\text{Dis}(R,S) \in K$, then $R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$.

	Model 1	Model 2	Model 3
Δ	$\{a, r, s\}$	$\{1, 2\}$	$\{\spadesuit\}$
I_I(rudiStuder)	r	1	$\spadesuit$
I_I(aifb)	a	2	$\spadesuit$
I_C(Professor)	$\{r\}$	$\{1\}$	$\{\spadesuit\}$
I_C(FacultyMember)	$\{a, r, s\}$	$\{1, 2\}$	$\{\spadesuit\}$
I_R(hasAffiliation)	$\{(r, a)\}$	$\{(1, 1), (1, 2)\}$	$\{(\spadesuit, \spadesuit)\}$

FIGURE 5.3: Models for the example knowledge base from page 174

We can now see that the example interpretation from page 174 is not a model: For it to be a model, we would need to have $(\texttt{rudiStuder}^{\mathcal{I}}, \texttt{aifb}^{\mathcal{I}}) \in \texttt{hasAffiliation}^{\mathcal{I}}$, i.e. we would need to have

$$(\text{Ian}, b) \in \{(a, b), (b, \text{Ian})\},$$

which is not the case.

The following determines an interpretation which is also a model of the example knowledge base from page 174.

$$\Delta = \{a, r, s\}$$
$$I_I(\texttt{rudiStuder}) = r$$
$$I_I(\texttt{aifb}) = a$$
$$I_C(\texttt{Professor}) = \{r\}$$
$$I_C(\texttt{FacultyMember}) = \{r, s\}$$
$$I_R(\texttt{hasAffiliation}) = \{(r, a)\}$$

Let us remark on a difference to RDF(S): for the $\mathcal{SROIQ}$ (i.e. OWL) semantics, we need to consider many different kinds of axioms. For RDF(S), however, we had to consider only one kind of axiom, namely triples.

5.2.1.3 Logical Consequences

Models capture the structure of a knowledge base in set-theoretic terms. However, a knowledge base can still have many models. Each of these models describes a meaningful interpretation of the knowledge base. Figure 5.3 lists several example models for the knowledge base from page 174.

So how do we make the step from models to a notion of logical consequence, i.e. how do we define what *implicit knowledge* a knowledge base entails? Figure 5.3 shows that it does not suffice to consider one or a few models.

$K \models C \sqsubseteq D$	iff $(C \sqsubseteq D)^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$	iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$
$K \models C(a)$	iff $(C(a))^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$	iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$
$K \models R(a, b)$	iff $(R(a, b))^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$	iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$
$K \models \neg R(a, b)$	iff $(\neg R(a, b))^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$	iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$ f.a. $\mathcal{I} \models K$

FIGURE 5.4: Logical consequences of a knowledge base. The first line states that $C \sqsubseteq D$ is a logical consequence of K if and only if $(C \sqsubseteq D)^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of K, which is the case if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of K.

For example, we have $\texttt{aifb}^{\mathcal{I}} \in \texttt{FacultyMember}^{\mathcal{I}}$ in all three models in Fig. 5.3, but we would not expect the conclusion from the knowledge base that aifb is a faculty member.

The right perspective on different models is the following: Each model of a knowledge base provides a *possible view* or *realization* of the knowledge base. The model captures all necessary structural aspects of the knowledge base, but it may add additional relationships which are not generally intended. In order to get rid of these additional relationships, we consider *all* models of a knowledge base when defining the notion of logical consequence. The rationale behind this idea is the following: If the models capture all possible views, or possible realizations, of a knowledge base, then those things common to *all* models must be universally valid logical consequences from the knowledge base. This leads us to the following formal definition.

Let K be a $\mathcal{SROIQ}$ knowledge base and α be a general inclusion axiom or an individual assignment. Then α is a *logical consequence* of K, written $K \models \alpha$, if $\alpha^{\mathcal{I}}$, as defined in Fig. 5.4, holds in every model I of K. Figure 5.5 contains an example related to logical consequence.

Let us introduce some further notions which are useful when dealing with model-theoretic semantics. A knowledge base is called *satisfiable* or *consistent* if it has at least one model. It is *unsatisfiable*, or *contradictory*, or *inconsistent*, if it is not satisfiable. A class expression C is called *satisfiable* if there is a model $\mathcal{I}$ of the knowledge base such that $C^{\mathcal{I}} \neq \emptyset$, and it is called *unsatisfiable* otherwise. Examples of these notions are given in Fig. 5.6.

Unsatisfiability of a knowledge base or of a named class usually points to modeling errors. But unsatisfiability also has other uses, which we will encounter in Section 5.3.

Returning to our running example knowledge base, let us show formally that FacultyMember(aifb) is not a logical consequence. This can be done by giving a model M of the knowledge base where $\texttt{aifb}^M \notin \texttt{FacultyMember}^M$. The following determines such a model:

$$\Delta = \{a, r\}$$
$$\texttt{I}_\texttt{I}(\texttt{rudiStuder}) = r$$
$$\texttt{I}_\texttt{I}(\texttt{aifb}) = a$$
$$\texttt{I}_\texttt{C}(\texttt{Professor}) = \{r\}$$
$$\texttt{I}_\texttt{C}(\texttt{FacultyMember}) = \{r\}$$
$$\texttt{I}_\texttt{R}(\texttt{hasAffiliation}) = \{(r, a)\}$$

FIGURE 5.5: Example of logical consequence

We give examples of these notions. The knowledge base consisting of the axioms

$$\texttt{Unicorn}(\texttt{beautyTheUnicorn})$$
$$\texttt{Unicorn} \sqsubseteq \texttt{Fictitious}$$
$$\texttt{Unicorn} \sqsubseteq \texttt{Animal}$$
$$\texttt{Fictitious} \sqcap \texttt{Animal} \sqsubseteq \bot$$

is inconsistent because beautyTheUnicorn would be a Fictitious Animal, which is forbidden by the last axiom. If we leave out the first individual assignment, then the resulting knowledge base is consistent, but Unicorn is unsatisfiable (i.e. is necessarily empty), as the existence of a Unicorn would lead to a contradiction.

FIGURE 5.6: Examples of notions of consistency and satisfiability

5.2.2 $\mathcal{SROIQ}$ Semantics via Predicate Logic

We now briefly present an alternative perspective on the semantics of OWL, namely by translating $\mathcal{SROIQ}$ knowledge bases into first-order predicate logic. This perspective serves two purposes:

- it shows that the formal semantics of OWL is based on the long-standing tradition of mathematical logic, and
- it helps to convey the semantics of OWL to those readers who already have some background in formal logic.

More precisely, the translation is into first-order predicate logic with equality, which is a mild generalization of first-order predicate logic with an equality predicate $=$ and with the unary $\top$ and $\bot$ predicates, with the obvious meaning and formal semantics. Every $\mathcal{SROIQ}$ knowledge base thus translates to a theory in first-order predicate logic with equality.

We give the translation of a $\mathcal{SROIQ}$ knowledge base K by means of a function π which is defined by $\pi(K) = \bigcup_{\alpha \in K} \pi(\alpha)$. How $\pi(\alpha)$ is defined depends on the type of the axiom α, and is specified in the following.

5.2.2.1 Translating Class Inclusion Axioms

If α is a class inclusion axiom of the form $C \sqsubseteq D$, then $\pi(\alpha)$ is defined inductively as in Fig. 5.7, where A is a class name.

5.2.2.2 Translating Individual Assignments

If α is an individual assignment, then $\pi(\alpha)$ is defined as

$$\pi(C(a)) = C(a),$$
$$\pi(R(a,b)) = R(a,b),$$
$$\pi(\neg R(a,b)) = \neg R(a,b),$$

i.e. the translation does nothing, due to the notational similarity of individual assignments in $\mathcal{SROIQ}$ to standard predicate logic notation.

5.2.2.3 Translating RBoxes

If α is an RBox statement, then $\pi(\alpha)$ is defined inductively as stated in Fig. 5.8, where S is a role name.

5.2.2.4 Properties of the Translation and an Example

The function π translates $\mathcal{SROIQ}$ knowledge bases to first-order predicate logic theories in such a way that K and $\pi(K)$ are very intimately related. Indeed, K and $\pi(K)$ have essentially identical models, where the models of $\pi(K)$ are defined as usual for first-order predicate logic. This means that we

$$\pi(C \sqsubseteq D) = (\forall x)(\pi_x(C) \rightarrow \pi_x(D))$$
$$\pi_x(A) = A(x)$$
$$\pi_x(\neg C) = \neg\pi_x(C)$$
$$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$$
$$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$$
$$\pi_x(\forall R.C) = (\forall x_1)(R(x, x_1) \rightarrow \pi_{x_1}(C))$$
$$\pi_x(\exists R.C) = (\exists x_1)(R(x, x_1) \wedge \pi_{x_1}(C))$$
$$\pi_x(\geq n S.C) = (\exists x_1) \ldots (\exists x_n) \left(\bigwedge_{i \neq j} (x_i \neq x_j) \wedge \bigwedge_i (S(x, x_i) \wedge \pi_{x_i}(C)) \right)$$
$$\pi_x(\leq n S.C) = \neg(\exists x_1) \ldots (\exists x_{n+1}) \left(\bigwedge_{i \neq j} (x_i \neq x_j) \wedge \bigwedge_i (S(x, x_i) \wedge \pi_{x_i}(C)) \right)$$
$$\pi_x(\{a\}) = (x = a)$$
$$\pi_x(\exists S.\mathrm{Self}) = S(x, x)$$

FIGURE 5.7: Translating $\mathcal{SROIQ}$ general inclusion axioms into first-order predicate logic with equality. Note that $\pi_x(\geq 0 S.C) = \top(x)$. We use auxiliary functions π_x, π_{x_1}, etc., where x, x_1, etc. are variables. Also note that variables $x_1 \ldots, x_{n+1}$ introduced on the right-hand sides should always be variables which are new, i.e. which have not yet been used in the knowledge base. Obviously, renamings are possible – and indeed advisable for better readability. The axiom $D \sqsubseteq \exists R.\exists S.C$, for example, could be translated to $(\forall x)((D(x)) \rightarrow (\exists y)(R(x, y) \wedge (\exists z)(S(y, z) \wedge C(z))))$.

$$\pi(R_1 \sqsubseteq R_2) = (\forall x)(\forall y)(\pi_{x,y}(R_1) \rightarrow \pi_{x,y}(R_2))$$
$$\pi_{x,y}(S) = S(x, y)$$
$$\pi_{x,y}(R^-) = \pi_{y,x}(R)$$
$$\pi_{x,y}(R_1 \circ \ldots \circ R_n) = (\exists x_1) \ldots (\exists x_{n-1}) \left(\pi_{x,x_1}(R_1) \wedge \bigwedge_{i=1}^{n-2} \pi_{x_i,x_{i+1}}(R_{i+1}) \wedge \pi_{x_{n-1},y}(R_n) \right)$$
$$\pi(\mathrm{Ref}(R)) = (\forall x)\pi_{x,x}(R)$$
$$\pi(\mathrm{Asy}(R)) = (\forall x)(\forall y)(\pi_{x,y}(R) \rightarrow \neg\pi_{y,x}(R)$$
$$\pi(\mathrm{Dis}(R_1, R_2)) = \neg(\exists x)(\exists y)(\pi_{x,y}(R_1) \wedge \pi_{x,y}(R_2))$$

FIGURE 5.8: Translating $\mathcal{SROIQ}$ RBoxes into first-order predicate logic

can understand $\mathcal{SROIQ}$ essentially as a fragment of first-order predicate logic, which means that it is in the tradition of mathematical logic, and results which have been achieved in this mathematical field can be carried over directly.

We have left out the treatment of datatypes in the translation, since it is unusual to consider predicate logic with datatypes. However, adding datatypes to predicate logic does not pose any particular problems unless complex operators on the datatype are allowed – which is not the case for OWL.

We close our discussion of the translation to predicate logic with an example, given in Fig. 5.9. It also shows that the established description logic notation is much easier to read than the corresponding first-order logic formulae.

5.3 Automated Reasoning with OWL

The formal model-theoretic semantics which we presented in Section 5.2 provides us with the logical underpinnings of OWL. At the heart of the formal semantics is that it provides means for accessing implicit knowledge, by the notion of logical consequence.

The definition of *logical consequence* given on page 177, however, does not lend itself easily to casting into an algorithm. Taken literally, it would necessitate examining *every* model of a knowledge base. Since there might be many models, and in general even infinitely many, a naive algorithmization of the definition of logical consequence is not feasible.

With OWL being a fragment of first-order predicate logic,it appears natural

Let K be the knowledge base containing the following axioms.

$$\texttt{Professor} \sqsubseteq \texttt{FacultyMember}$$
$$\texttt{Professor} \sqsubseteq (\texttt{Person} \sqcap \texttt{FacultyMember}) \sqcup (\texttt{Person} \sqcap \neg\texttt{PhDStudent})$$
$$\texttt{Exam} \sqsubseteq \forall \texttt{hasExaminer}.\texttt{Professor}$$
$$\texttt{Exam} \sqsubseteq \leq 2\texttt{hasExaminer}$$
$$\texttt{hasParent} \circ \texttt{hasBrother} \sqsubseteq \texttt{hasUncle}$$
$$\texttt{Professor}(\texttt{rudiStuder})$$
$$\texttt{hasAffiliation}(\texttt{rudiStuder}, \texttt{aifb})$$

Then $\pi(K)$ contains the following logical formulae.

$$(\forall x)(\texttt{Professor}(x) \rightarrow \texttt{FacultyMember}(x)),$$
$$(\forall x)(\texttt{Professor}(x) \rightarrow ((\texttt{Person}(x) \wedge \texttt{FacultyMember}(x)) \vee (\texttt{Person}(x) \wedge \neg\texttt{PhDStudent})(x))),$$
$$(\forall x)(\texttt{Exam}(x) \rightarrow (\forall y)(\texttt{hasExaminer}(x, y) \rightarrow \texttt{Professor}(y))),$$
$$(\forall x)(\texttt{Exam}(x) \rightarrow \neg(\exists x_1)(\exists x_2)(\exists x_3)((x_1 \neq x_2) \wedge (x_2 \neq x_3) \wedge (x_1 \neq x_3) \wedge \texttt{hasExaminer}(x, x_1) \wedge \texttt{hasExaminer}(x, x_2) \wedge \texttt{hasExaminer}(x, x_3))),$$
$$(\forall x)(\forall y)(((\exists x_1)(\texttt{hasParent}(x, x_1) \wedge \texttt{hasBrother}(x_1, y))) \rightarrow \texttt{hasUncle}(x, y)),$$
$$\texttt{Professor}(\texttt{rudiStuder}),$$
$$\texttt{hasAffiliation}(\texttt{rudiStuder}, \texttt{aifb})$$

FIGURE 5.9: Example of translation from description logic syntax to first-order predicate logic syntax

to employ deduction algorithms from predicate logic and to simply adjust them to the description logic setting. This has indeed been done for all the major inference systems from predicate logic.

By far the most successful approach for description logics to date is based on tableaux algorithms, suitably adjusted to OWL. We present this in the following. Since these algorithms are somewhat sophisticated, we do this first for $\mathcal{ALC}$, and then extend the algorithm to $\mathcal{SHIQ}$. We refrain from presenting the even more involved algorithm for $\mathcal{SROIQ}$, as $\mathcal{SHIQ}$ allows us to convey the central ideas.

But before coming to the algorithms, we need some preparation.

5.3.1 Inference Problems

In Section 4.1.10 we introduced the typical types of inferences which are of interest in the context of OWL. Let us recall them here from a logical perspective.

- *Subsumption.* To find out whether a class C is a subclass of D (i.e. whether C is *subsumed* by D), we have to find out whether $C \sqsubseteq D$ is a logical consequence of the given knowledge base.
- *Class equivalence.* To find out whether a class C is equivalent to a class D, we have to find out if $C \equiv D$ is a logical consequence of the given knowledge base.
- *Class disjointness.* To find out whether two classes C and D are disjoint, we have to find out whether $C \sqcap D \sqsubseteq \bot$ is a logical consequence of the given knowledge base.
- *Global consistency.* To find out whether the given knowledge base is globally consistent, we have to show that it has a model.
- *Class consistency.* To find out whether a given class D is consistent, we have to show that $C \sqsubseteq \bot$ is *not* a logical consequence of the given knowledge base.
- *Instance checking.* To find out if an individual a belongs to a class C, we have to check whether $C(a)$ is a logical consequence of the knowledge base.
- *Instance retrieval.* To find all individuals belonging to a class C, we have to check for all individuals whether they belong to C.

It would be very inconvenient if we had to devise a separate algorithm for each inference type. Fortunately, description logics allow us to reduce these inference problems to each other. For the tableaux algorithms, we need to reduce them to the checking of knowledge base satisfiability, i.e. to the question whether a knowledge base has at least one model. This is done as follows, where K denotes a knowledge base.

- *Subsumption.* $K \models C \sqsubseteq D$ if and only if $K \cup \{(C \sqcap \neg D)(a)\}$ is unsatisfiable, where a is a new individual not occurring in K.
- *Class equivalence.* $K \models C \equiv D$ if and only if we have $K \models C \sqsubseteq D$ and $K \models D \sqsubseteq C$.
- *Class disjointness.* $K \models C \sqcap D \sqsubseteq \bot$ if and only if $K \cup \{(C \sqcap D)(a)\}$ is unsatisfiable, where a is a new individual not occurring in K.
- *Global consistency.* K is globally consistent if it has a model.
- *Class consistency.* $K \models C \sqsubseteq \bot$ if and only if $K \cup \{C(a)\}$ is unsatisfiable, where a is a new individual not occurring in K.
- *Instance checking.* $K \models C(a)$ if and only if $K \cup \{\neg C(a)\}$ is unsatisfiable
- *Instance retrieval.* To find all individuals belonging to a class C, we have to check for all individuals a whether $K \models C(a)$.

Note that, strictly speaking, statements such as $\neg C(a)$ or $(C \sqcap \neg D)(a)$ are not allowed according to our definition of ABox in Section 5.1.1.3. However, complex class expressions like $C(a)$ in the ABox, where C is an arbitrary class expression, can easily be transformed to comply with our formal definition, namely, by introducing a new class name, say A, and rewriting $C(a)$ to the two statements $A(a)$ and $A \equiv C$. This technique is known as *ABox reduction*, and can also be applied to $\mathcal{SROIQ}$. The knowledge bases before and after the reduction are essentially equivalent. Without loss of generality, we will therefore allow complex classes in the ABox in this chapter.

We have now reduced all inference types to satisfiability checking. In principle, we could now use the transformation into predicate logic from Section 5.2.2 and do automated reasoning on OWL using predicate logic reasoning systems. This approach, however, is not very efficient, so special-purpose algorithms tailored to description logics are preferable. But there is also a more fundamental problem with the translational approach: $\mathcal{SROIQ}$, and also the description logics it encompasses, are decidable, while first-order predicate logic is not. This means that, in general, termination of description logic reasoning cannot be guaranteed by using reasoning algorithms for first-order predicate logic.

Nevertheless, the tableaux algorithms which we present in the following are derived from the corresponding first-order predicate logic proof procedures. And we will return to the termination issue later.

5.3.2 Negation Normal Form

Before presenting the actual algorithms, we do a preprocessing on the knowledge base known as negation normal form transformation, i.e. we transform the knowledge base into a specific syntactic form known as negation normal

$$\mathrm{NNF}(K) = A \sqcup \mathcal{R} \cup \bigcup_{C \sqsubseteq D \in K} \mathrm{NNF}(C \sqsubseteq D) \quad \text{where } A \text{ and } \mathcal{R} \text{ are the ABox and the RBox of } K$$

$$\mathrm{NNF}(K)(C \sqsubseteq D) = \mathrm{NNF}(\neg C \sqcup D)$$
$$\mathrm{NNF}(C) = C \quad \text{if } C \text{ is a class name}$$
$$\mathrm{NNF}(\neg C) = \neg C \quad \text{if } C \text{ is a class name}$$
$$\mathrm{NNF}(\neg\neg C) = \mathrm{NNF}(C)$$
$$\mathrm{NNF}(C \sqcup D) = \mathrm{NNF}(C) \sqcup \mathrm{NNF}(D)$$
$$\mathrm{NNF}(C \sqcap D) = \mathrm{NNF}(C) \sqcap \mathrm{NNF}(D)$$
$$\mathrm{NNF}(\neg(C \sqcup D)) = \mathrm{NNF}(\neg C) \sqcap \mathrm{NNF}(\neg D)$$
$$\mathrm{NNF}(\neg(C \sqcap D)) = \mathrm{NNF}(\neg C) \sqcup \mathrm{NNF}(\neg D)$$
$$\mathrm{NNF}(\forall R.C) = \forall R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\exists R.C) = \exists R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\neg\forall R.C) = \exists R.\mathrm{NNF}(\neg C)$$
$$\mathrm{NNF}(\neg\exists R.C) = \forall R.\mathrm{NNF}(\neg C)$$
$$\mathrm{NNF}(\leq n R.C) = \leq n R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\geq n R.C) = \geq n R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\neg\leq n R.C) = \geq (n+1) R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\neg\geq (n+1) R.C) = \leq n R.\mathrm{NNF}(C)$$
$$\mathrm{NNF}(\neg\geq 0 R.C) = \bot$$

FIGURE 5.10: Transformation of a $\mathcal{SHIQ}$ knowledge base K into negation normal form

form. It is not absolutely necessary to do this, and the algorithms could also be presented without this preprocessing step, but they are already complicated enough as they are, and restricting our attention to knowledge bases in negation normal form eases the presentation considerably.

In a nutshell, the negation normal form $\mathrm{NNF}(K)$ of a knowledge base K is obtained by first rewriting all $\sqsubseteq$ symbols in an equivalent way, and then moving all negation symbols down into subformulae until they only occur directly in front of class names. How this is done formally is presented in Fig. 5.10 for $\mathcal{SHIQ}$. Note that only the TBox is transformed.

In the negation normal form transformation, subclass relationships like $C \sqsubseteq D$ become class expressions $\neg C \sqcup D$ which, intuitively, may look strange at first sight. Cast into first-order predicate logic, however, they become

$(\forall x)(C(x) \rightarrow D(x))$ and $(\forall x)(\neg C(x) \vee D(x))$ – and these two formulae are logically equivalent.

By slight abuse of terminology, we will henceforth refer to $\mathrm{NNF}(C \sqsubseteq D)$ as a *TBox statement* whenever $C \sqsubseteq D$ is contained in the TBox of the knowledge base currently under investigation.

The knowledge bases K and $\mathrm{NNF}(K)$ are logically equivalent, i.e. they have identical models. We assume for the rest of this chapter that all knowledge bases are given in negation normal form.

5.3.3 Tableaux Algorithm for $\mathcal{ALC}$

The tableaux algorithm determines if a knowledge base is satisfiable. It does this by attempting to construct a generic representation of a model. If this construction fails, the knowledge base is unsatisfiable.

Obviously, it requires formal proofs to verify that such an algorithm indeed does what it claims. In this book, however, we do not have the space or the means to present this verification, which is based on comprehensive mathematical proofs. We refer the interested reader to the literature listed in Section 5.6. Nevertheless, by keeping in mind that tableaux algorithms essentially attempt to construct models, it should become intuitively clear why they indeed implement automated reasoning.

We now start with the description logic $\mathcal{ALC}$. The presentation of the corresponding tableaux algorithm is done in three stages to make this introduction easier to follow. We first informally discuss some examples. Then we formally define the *naive* tableaux algorithm for $\mathcal{ALC}$. It only is a small step then to provide the full tableaux algorithm.

5.3.3.1 Initial Examples

Consider a very simple case, where we have only class names, conjunction, disjunction, negation, and only one individual. We are given such a knowledge base and we are to determine whether it is satisfiable. Let us have a look at an example.

Assume the knowledge base K consists of the following two statements.

$$C(a) \qquad (\neg C \sqcap D)(a)$$

Then obviously $C(a)$ is a logical consequence of K. From the statement $(\neg C \sqcap D)(a)$ we also obtain $\neg C(a)$ as logical consequence – this is due to the semantics of conjunction. But this means that we have been able to derive $C(a)$ and $\neg C(a)$, which is a contradiction. So K cannot have a model and is therefore unsatisfiable.

What we have just constructed is essentially a part of a tableau. Informally speaking, a tableau is a structured way of deriving and representing logical consequences of a knowledge base. If in this process a contradiction is found, then the initial knowledge base is unsatisfiable.

Let us consider a slightly more difficult case. Assume the negation normal form of a knowledge base K consists of the following three statements.

$$C(a) \qquad \neg C \sqcup D \qquad \neg D(a)$$

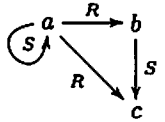
We are now going to derive knowledge about class membership for a, as done in the previous example. The set of all classes for which we have derived class membership of a will be called $\mathcal{L}(a)$. We use the notation $\mathcal{L}(a) \leftarrow C$ to indicate that $\mathcal{L}(a)$ is updated by adding C. For example, if $\mathcal{L}(a) = \{D\}$ and we update via $\mathcal{L}(a) \leftarrow C$, then $\mathcal{L}(a)$ becomes $\{C, D\}$. Similarly, $\mathcal{L}(a) \leftarrow \{C, D\}$ denotes the subsequent application of $\mathcal{L}(a) \leftarrow C$ and $\mathcal{L}(a) \leftarrow D$, i.e. both C and D get added to $\mathcal{L}(a)$.

From the example knowledge base just given, we immediately obtain $\mathcal{L}(a) = \{C, \neg D\}$. The TBox statement $\neg C \sqcup D$ corresponds to $C \sqsubseteq D$ and must hold for all individuals, i.e. in particular for a, so we obtain $\mathcal{L}(a) \leftarrow \neg C \sqcup D$. Now consider the expression $(\neg C \sqcup D) \in \mathcal{L}(a)$, which states that we have $\neg C(a)$ or $D(a)$. So we distinguish two cases. (1) In the first case we assume $\neg C(a)$ and obtain $\mathcal{L}(a) \leftarrow \neg C = \{C, \neg D, \neg C \sqcup D, \neg C\}$, which is a contradiction. (2) In the second case we assume $D(a)$ and obtain $\mathcal{L}(a) \leftarrow D = \{C, \neg D, \neg C \sqcup D, D\}$, which is also a contradiction. In either case, we arrive at a contradiction which indicates that K is unsatisfiable.

Note the *branching* we had to do in the example in order to deal with disjunction. This and similar situations lead to nondeterminism of the tableaux algorithm, and we will return to this observation later.

In the previous section we have provided examples of how to deal with class membership information for individuals in a tableau, i.e. how to derive contradictions from this information. Our examples were restricted to single individuals, and we did not use any roles.

So how do we represent role information? We represent it graphically as arrows between individuals. Consider an ABox consisting of the assignments $R(a, b)$, $S(a, a)$, $R(a, c)$, $S(b, c)$. This would be represented as the following figure.



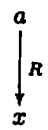
Likewise, we use arrows to represent roles between unknown individuals, the existence of which is ascertained by the knowledge base: Consider the single statement $\exists R.\exists S.C(a)$. Then there is an arrow labeled with R leading from a to an unknown individual x, from which in turn there is an arrow labeled with S to a second unknown individual y. The corresponding picture would be the following.

$$a \xrightarrow{R} x \xrightarrow{S} y$$

Let us give an example tableau involving roles. Consider the knowledge base $K = \{C(a), C \sqsubseteq \exists R.D, D \sqsubseteq E\}$, so that $\mathrm{NNF}(K) = \{C(a), \neg C \sqcup \exists R.D, \neg D \sqcup$

$E\}$. We would like to know if $(\exists R.E)(a)$ is a logical consequence of K.

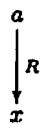
We first reduce the instance checking problem to a satisfiability problem as described in Section 5.3.1: $\neg\exists R.E$ in negation normal form becomes $\forall R.\neg E$, and we obtain the knowledge base $\{C(a), \neg C \sqcup \exists R.D, \neg D \sqcup E, \forall R.\neg E(a)\}$, of which we have to show that it is unsatisfiable. We start with the node a with label $\mathcal{L}(a) = \{C, \forall R.\neg E\}$, which is information we take from the ABox. The first TBox statement results in $\mathcal{L}(a) \leftarrow \neg C \sqcup \exists R.D$. We can now resolve the disjunction as we have done above, i.e. we have to consider two cases. Adding $\neg C$ to $\mathcal{L}(a)$, however, results in a contradiction since $C \in \mathcal{L}(a)$, so we do not have to consider this case, i.e. we end up with $\mathcal{L}(a) \leftarrow \exists R.D$. So, since $\exists R.D \in \mathcal{L}(a)$, we create a new individual x and a connection labeled R from a to x, and we set $\mathcal{L}(x) = \{D\}$. The situation is as follows.



$\mathcal{L}(a) = \{C, \forall R.\neg E, \neg C \sqcup \exists R.D, \exists R.D\}$

$\mathcal{L}(x) = \{D\}$

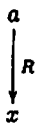
The TBox information $\neg D \sqcup E$ can now be added to $\mathcal{L}(x)$, i.e. $\mathcal{L}(x) \leftarrow \neg D \sqcup E$, and expanded using the already known case distinction because of the disjunction. As before, however, selecting the left hand side $\neg D$ results in a contradiction because $D \in \mathcal{L}(x)$, so we have to put $\mathcal{L}(x) \leftarrow E$. The situation is now as follows.



$\mathcal{L}(a) = \{C, \forall R.\neg E, \neg C \sqcup \exists R.D, \exists R.D\}$

$\mathcal{L}(x) = \{D, \neg D \sqcup E, E\}$

Now note that $\forall R.\neg E \in \mathcal{L}(a)$, which means that everything to which a connects using the R role must be contained in $\neg E$. Since a connects to x via an arrow labeled R, we set $\mathcal{L}(x) \leftarrow \neg E$, which results in a contradiction because we already have $E \in \mathcal{L}(x)$. Thus, the knowledge base is unsatisfiable, and the instance checking problem is solved, i.e. $(\exists R.E)(a)$ is indeed a logical consequence of K. The final tableau is depicted below.



$\mathcal{L}(a) = \{C, \forall R.\neg E, \neg C \sqcup \exists R.D, \exists R.D\}$

$\mathcal{L}(x) = \{D, \neg D \sqcup E, E, \neg E\}$

It is now time to leave the intuitive introduction and to formalize the tableau procedure.

5.3.3.2 The Naive Tableaux Algorithm for $\mathcal{ALC}$

A tableau for an $\mathcal{ALC}$ knowledge base consists of

- a set of nodes, labeled with individual names or variable names,

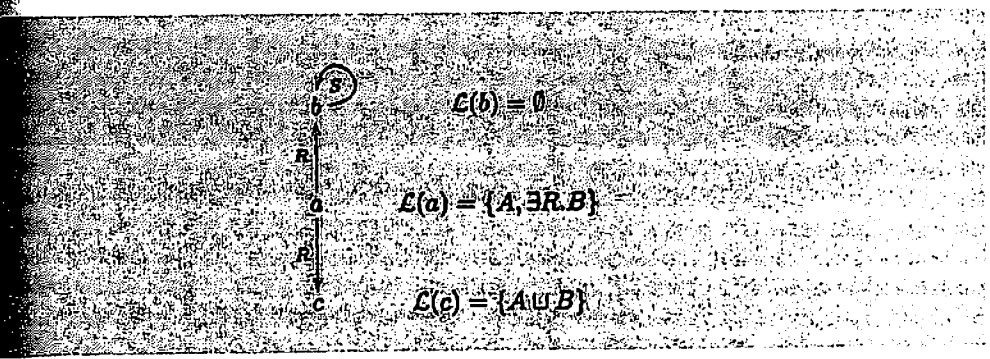


FIGURE 5.11: Example of an initial tableau given the knowledge base $K = \{A(a), (\exists R.B)(a), R(a,b), R(a,c), S(b,b), (A \sqcup B)(c), \neg A \sqcup (\forall S.B)\}$

- directed edges between some pairs of nodes,
- for each node labeled x, a set $\mathcal{L}(x)$ of class expressions, and
- for each pair of nodes x and y, a set $\mathcal{L}(x, y)$ of role names.

When we depict a tableau, we omit edges which are labeled with the empty set. Also, we make the agreement that $\top$ is contained in $\mathcal{L}(x)$, for any x, but we often do not write it down, and in fact the algorithm does not explicitly derive this.

Given an $\mathcal{ALC}$ knowledge base K in negation normal form, the *initial tableau* for K is defined by the following procedure.

1. For each individual a occurring in K, create a node labeled a and set $\mathcal{L}(a) = \emptyset$.
2. For all pairs a, b of individuals, set $\mathcal{L}(a, b) = \emptyset$.
3. For each ABox statement $C(a)$ in K, set $\mathcal{L}(a) \leftarrow C$.
4. For each ABox statement $R(a, b)$ in K, set $\mathcal{L}(a, b) \leftarrow R$.

An example of an initial tableau can be found in Fig. 5.11.

After initialization, the tableaux algorithm proceeds by nondeterministically applying the rules from Fig. 5.12. This means that at each step one of the rules is selected and executed. The algorithm terminates if

- either there is a node x such that $\mathcal{L}(x)$ contains a contradiction, i.e. if there is $C \in \mathcal{L}(x)$ and at the same time $\neg C \in \mathcal{L}(x)$,[9]

[9] This includes the case when both $\bot$ and $\top$ are contained in $\mathcal{L}(x)$, which is also a contradiction as $\top \equiv \neg\bot$.

⊓-rule: If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

⊔-rule: If $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

∃-rule: If $\exists R.C \in \mathcal{L}(x)$ and there is no y with $R \in \mathcal{L}(x, y)$ and $C \in \mathcal{L}(y)$, then

1. add a new node with label y (where y is a new node label),
2. set $\mathcal{L}(x, y) = \{R\}$, and
3. set $\mathcal{L}(y) = \{C\}$.

∀-rule: If $\forall R.C \in \mathcal{L}(x)$ and there is a node y with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

TBox-rule: If C is a TBox statement and $C \notin \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$.

FIGURE 5.12: Expansion rules for the naive $\mathcal{ALC}$ tableaux algorithm

- or none of the rules from Fig. 5.12 is applicable.

The knowledge base K is satisfiable if the algorithm terminates without producing a contradiction, i.e. if there is a selection of subsequent rule applications such that no contradiction is produced and the algorithm terminates. Otherwise, K is unsatisfiable. Note that due to the nondeterminism of the algorithm we do not know which choice of subsequent rule applications leads to termination without producing a contradiction. Implementations of this algorithm thus have to guess the choices, and possibly have to backtrack to choice points if a choice already made has led to a contradiction.

Let us explain this point in more detail since it is critical to understanding the algorithm. There are two sources of nondeterminism, namely (1) which expansion rule to apply next and (2) the choice which has to be made when applying the ⊔-rule, namely whether to set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$ (using the notation from Fig. 5.12). There is a fundamental difference between these two: The choice made in (1) is essentially a choice about the sequence in which the rules are applied, i.e. whatever results from such a choice could also be obtained by doing the same expansion later. Intuitively speaking, we cannot get "on the wrong track" by a bad choice, although some choices will cause the algorithm to take more steps before termination. Hence, if such a choice causes a contradiction, then this contradiction cannot be avoided by making a different choice, simply because the original choice can still be made later – and entries are never removed from node labels during execution. This kind of nondeterminism is usually called *don't care* nondeterminism. In contrast to this, (2) is a *don't know* nondeterminism, since a bad choice can indeed

get us "on the wrong track." This is because, if we choose to set $\mathcal{L}(x) \leftarrow C$, then it is no longer possible to also add $\mathcal{L}(x) \leftarrow D$ by applying the same rule – the condition $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ prevents this. So if we have chosen to add $\mathcal{L}(x) \leftarrow C$ and this leads to a contradiction, then we have to go back to this choice and try the other alternative as well, because this other alternative may not lead to a contradiction.

To sum this up, note the following. If the sequence of choices (of both types) leads to termination without producing a contradiction, then the original knowledge base is satisfiable. However, if the algorithm produces a contradiction, then we do not yet know if there is a sequence of choices which avoids the contradiction. Hence, we have to check on all choices made due to (2) and see if we also get contradictions if we alter these choices – in other words, we have to *backtrack* to these *choice points*. But it is not necessary to reconsider the choices made due to (1). We recommend the reader to go back to the initial examples in Section 5.3.3.1 and observe how we have done this: It occurs in all the cases where we have dismissed one of the choices from applying the ⊔-rule because it would lead to a contradiction.

We will see in the next section that the naive tableaux algorithm does not necessarily terminate. This will be fixed then. But we first present, in Fig. 5.13, another worked example.

5.3.3.3 The Tableaux Algorithm with Blocking for $\mathcal{ALC}$

We have already remarked that the naive tableaux algorithm for $\mathcal{ALC}$ does not always terminate. To see this, consider $K = \{\exists R.\top, \top(a_1)\}$. First note that K is satisfiable: consider the interpretation $\mathcal{I}$ with infinite domain $\{a_1, a_2, \dots\}$ such that $a_1^{\mathcal{I}} = a_1$ and $(a_i, a_{i+1}) \in R^{\mathcal{I}}$ for all $i = 1, 2, \dots$. Then $\mathcal{I}$ is obviously a model.

Now we construct a tableau for K, as depicted below. Initialization leaves us with one node a_1 and $\mathcal{L}(a_1) = \{\top\}$. Applying the TBox-rule yields $\mathcal{L}(a_1) \leftarrow \exists R.\top$. Then we apply the ∃-rule and create a node x with $\mathcal{L}(a_1, x) = \{R\}$ and $\mathcal{L}(x) = \{\top\}$. Again we apply the TBox-rule which yields $\mathcal{L}(x) \leftarrow \exists R.\top$, and then the ∃-rule allows us to create yet another new node y with $\mathcal{L}(x, y) = \{R\}$ and $\mathcal{L}(y) = \{\top\}$. Obviously, this process repeats and does not terminate.

$$a_1 \xrightarrow{R} x \xrightarrow{R} y \xrightarrow{R} \cdots$$

$$\mathcal{L}(a_1) = \{\top, \exists R.\top\} \qquad \mathcal{L}(x) = \{\top, \exists R.\top\} \qquad \mathcal{L}(y) = \{\top, \exists R.\top\}$$

But we remarked earlier that $\mathcal{ALC}$ (and actually also $\mathcal{SROIQ}$) is decidable, i.e. algorithms exist which allow reasoning with $\mathcal{ALC}$ and which are always guaranteed to terminate! To ensure termination in all cases, we have to modify the naive tableaux algorithm. The technique used for this purpose is called *blocking*, and rests on the observation that in the above example, the process is essentially repeating itself: The newly created node x has the same properties as the node a_1, so instead of expanding x to a new node y it should be possible to "reuse" a_1 in some sense.

Consider the following knowledge base K.

$$\texttt{Human} \sqsubseteq \exists \texttt{hasParent.Human}$$
$$\texttt{Orphan} \sqsubseteq \texttt{Human} \sqcap \forall \texttt{hasParent.}\neg\texttt{Alive}$$
$$\texttt{Orphan(harrypotter)}$$
$$\texttt{hasParent(harrypotter, jamespotter)}$$

We want to know if $\alpha = \neg\texttt{Alive(jamespotter)}$ is a logical consequence of K.

We first add $\neg\neg\texttt{Alive(jamespotter)}$ to K and call the result K'. In order to show that α is a logical consequence of K, we have to show that K' is unsatisfiable. We now transform K' into negation normal form. We also use some straightforward shortcuts to ease the notation. So NNF(K') is the following:

$$\neg H \sqcup \exists P.H$$
$$\neg O \sqcup (H \sqcap \forall P.\neg A)$$
$$O(h)$$
$$P(h,j)$$
$$A(j)$$

The initial tableau for NNF(K') is depicted below:

We now apply the TBox-rule and set $\mathcal{L}(h) \leftarrow \neg O \sqcup (H \sqcap \forall P.\neg A)$. Applying the $\sqcup$-rule to the same TBox axiom leaves us with a choice how to resolve the disjunction. However, choosing the left hand side $\neg O$ immediately results in a contradiction since $O \in \mathcal{L}(h)$, so, backtracking, we choose to set $\mathcal{L}(h) \leftarrow H \sqcap \forall P.\neg A$. Applying the $\sqcap$-rule results in $\mathcal{L}(h) \leftarrow \{H, \forall P.\neg A\}$. Finally, we apply the $\forall$-rule to $\forall P.\neg A \in \mathcal{L}(h)$, which results in $\mathcal{L}(j) \leftarrow \neg A$. Since $A \in \mathcal{L}(j)$, we have thus arrived at an unavoidable contradiction, i.e. K' is unsatisfiable, and therefore $\neg A(j)$ is a logical consequence of K. The final tableau is depicted below.

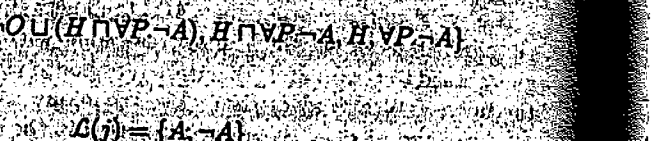


FIGURE 5.13: Worked example of $\mathcal{ALC}$ tableau

The formal definition is as follows: A node with label x is *directly blocked* by a node with label y if

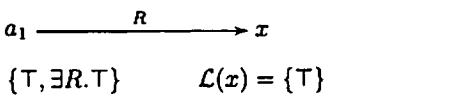
- x is a variable (i.e. not an individual),
- y is an ancestor of x, and
- $\mathcal{L}(x) \subseteq \mathcal{L}(y)$.

The notion of *ancestor* is defined inductively as follows: Every z with $\mathcal{L}(z,x) \neq \emptyset$ is called a *predecessor* of x. Every predecessor of x, which is not an individual, is an ancestor of x, and every predecessor of an ancestor of x, which is not an individual, is also an ancestor of x.

A node with label x is *blocked* if it is directly blocked or one of its ancestors is blocked.

The naive tableaux algorithm for $\mathcal{ALC}$ is now modified as follows, resulting in the (full) tableaux algorithm for $\mathcal{ALC}$: The rules in Fig. 5.12 may only be applied *if x is not blocked.* Otherwise, the algorithm is exactly the naive algorithm.

Returning to the example above, we note that $\mathcal{L}(x) \subseteq \mathcal{L}(a_1)$, so x is blocked by a_1. This means that the algorithm terminates with the following tableau, and therefore shows that the knowledge base is satisfiable.



Recall the model for this knowledge base which we gave on page 191. Intuitively, the blocked node x is a representative for the infinite set $\{a_2, a_3, \dots\}$. Alternatively, we could view the tableau as standing for the model $\mathcal{J}$ with domain $\{a_1, a\}$ such that $a_1^{\mathcal{J}} = a_1$, $x^{\mathcal{J}} = a$ and $R^{\mathcal{J}} = \{(a_1, a), (a, a)\}$, i.e. the model would be cyclic.

5.3.3.4 Worked Examples

We give a number of worked examples which show some aspects of the algorithm in more detail.

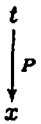
5.3.3.4.1 Blocking Consider $K = \{H \sqsubseteq \exists P.H, B(t)\}$ as knowledge base, which stands for

$$\texttt{Human} \sqsubseteq \exists \texttt{hasParent.Human}$$
$$\texttt{Bird(tweety)}$$

We try to show that `tweety` is *not* in the class $\neg$Human, i.e. that $\neg H(t)$ is *not* a logical consequence of K. To do this, we add $\neg\neg H(t)$ to K, resulting in K', and attempt to show that K' is unsatisfiable. Obviously, this attempt

will not be successful, which shows that tweety *could* be in the class Human according to the knowledge base.

We obtain $\text{NNF}(K') = \{\neg H \sqcup \exists P.H, B(t), H(t)\}$. The tableau is initialized with one node t and $\mathcal{L}(t) = \{B, H\}$. Applying the TBox rule yields $\mathcal{L}(t) \leftarrow \neg H \sqcup \exists P.H$. Expanding this TBox axiom using the $\sqcup$-rule results in $\mathcal{L}(t) \leftarrow \exists P.H$ since the addition of $\neg H$ to $\mathcal{L}(t)$ would immediately yield a contradiction. We now apply the $\exists$-rule and create a node with label x, $\mathcal{L}(t, x) = \{P\}$, and $\mathcal{L}(x) = \{H\}$. At this stage, the node x is blocked by t, and no further expansion of the tableau is possible.



$\mathcal{L}(t) = \{H, B, \neg H \sqcup \exists P.H, \exists P.H\}$

$\mathcal{L}(x) = \{H\}$

5.3.3.4.2 Open World Consider the knowledge base

$$K = \{h(j,p), h(j,a), m(p), m(a)\},$$

which consists only of an ABox. The knowledge base stands for the following.

hasChild(john, peter)
hasChild(john, alex)
Male(peter)
Male(alex)

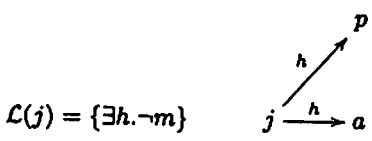
We want to show that $\forall$hasChild.male(john) is *not* a logical consequence of the knowledge base. We do this by adding the negation of the statement, $\neg\forall h.m(j)$, resulting in the knowledge base K'. We then need to show that K' is satisfiable.

Let us first try to understand why K' is satisfiable. Due to the Open World Assumption as discussed on page 131, the knowledge base contains no information whether or not john has only peter and alex as children. It is entirely possible that john has additional children who are not listed in the knowledge base. Therefore, it is not possible to infer that all of john's children are Male. We will see how the tableaux algorithm mimics this.

Transformation into negation normal form yields

$$\text{NNF}(K') = \{h(j,p), h(j,a), m(p), m(a), \exists h.\neg m(j)\}.$$

The initial tableau for $\text{NNF}(K')$ can be depicted as follows.

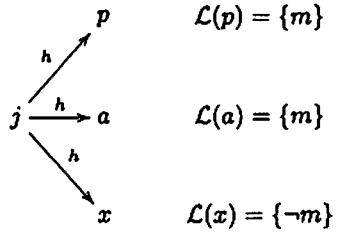


$\mathcal{L}(p) = \{m\}$

$\mathcal{L}(j) = \{\exists h.\neg m\}$

$\mathcal{L}(a) = \{m\}$

Now, application of the $\exists$-rule yields a new node x with $\mathcal{L}(j, x) = \{h\}$ and $\mathcal{L}(x) = \{\neg m\}$, as depicted below.



$\mathcal{L}(p) = \{m\}$

$\mathcal{L}(j) = \{\exists h.\neg m\}$

$\mathcal{L}(a) = \{m\}$

$\mathcal{L}(x) = \{\neg m\}$

At this stage, the algorithm terminates since none of the rules is applicable. This means that K' is satisfiable.

The new node x represents a potential child of john who is not male. Note how the constructed tableau indeed corresponds to a model of the knowledge base K'.

5.3.3.4.3 A Sophisticated Example We close our discussion of the $\mathcal{ALC}$ tableaux algorithm with a more sophisticated example. We start with the knowledge base K containing the statements

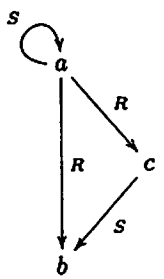
$$C(a), \quad C(c), R(a,b), \quad R(a,c), \quad S(a,a), \quad S(c,b),$$
$$C \sqsubseteq \forall S.A, \quad A \sqsubseteq \exists R.\exists S.A, \quad A \sqsubseteq \exists R.C,$$

and want to show that $\exists R.\exists R.\exists S.A(a)$ is a logical consequence of K.

We first add $\neg\exists R.\exists R.\exists S.A(a)$, which results in K'. The knowledge base $\text{NNF}(K')$ then consists of

$$C(a), \quad C(c), R(a,b), \quad R(a,c), \quad S(a,a), \quad S(c,b),$$
$$\neg C \sqcup \forall S.A, \quad \neg A \sqcup \exists R.\exists S.A, \quad \neg A \sqcup \exists R.C, \quad \forall R.\forall R.\forall S.\neg A(a),$$

and the initial tableau for $\text{NNF}(K')$ is the following.



$\mathcal{L}(a) = \{C, \forall R.\forall R.\forall S.\neg A\}$

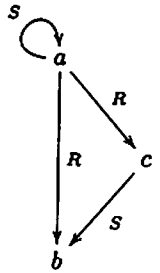
$\mathcal{L}(c) = \{C\}$

$\mathcal{L}(b) = \emptyset$

At this stage, there are many choices of which rules to apply and at which node. We urge the reader to attempt solving the tableau by herself before reading on. Indeed, the tableau can grow considerably larger if the expansion rules are chosen more or less randomly.

We choose to first use the TBox-rule and set $\mathcal{L}(c) \leftarrow \neg C \sqcup \forall S.A$. Expanding this TBox axiom using the $\sqcup$-rule gives us a choice, but we refrain from adding $\neg C$ to $\mathcal{L}(c)$ as this would contradict $C \in \mathcal{L}(c)$. So we set $\mathcal{L}(c) \leftarrow \forall S.A$.

As the next step, we choose $\forall S.A \in \mathcal{L}(c)$ and apply the $\forall$-rule, resulting in $\mathcal{L}(b) \leftarrow A$. Then we choose $\forall R.\forall R.\forall S.\neg A \in \mathcal{L}(a)$ and again apply the $\forall$-rule, resulting in $\mathcal{L}(b) \leftarrow \forall R.\forall R.\forall S.\neg A$. Applying the TBox-rule we set $\mathcal{L}(b) \leftarrow \neg A \sqcup \exists R.\exists S.A$. The following picture shows the current situation.



$$\mathcal{L}(a) = \{C, \forall R.\forall R.\forall S.\neg A\}$$

$$\mathcal{L}(c) = \{C, \neg C \sqcup \forall S.A, \forall S.A\}$$

$$\mathcal{L}(b) = \{A, \forall R.\forall S.\neg A, \neg A \sqcup \exists R.\exists S.A\}$$

The node b is now going to be the key to finding a contradiction. Using the $\sqcup$-rule on $\neg A \sqcup \exists R.\exists S.A \in \mathcal{L}(b)$ yields $\mathcal{L}(b) \leftarrow \exists R.\exists S.A$ since the addition of $\neg A$ would already result in a contradiction. We apply the $\exists$-rule to $\exists R.\exists S.A \in \mathcal{L}(b)$, creating a new node x with $\mathcal{L}(b,x) = \{R\}$ and $\mathcal{L}(x) = \{\exists S.A\}$. The $\forall$-rule for $\forall R.\forall S.\neg A$ yields $\mathcal{L}(x) \leftarrow \forall S.\neg A$.

Finally, we use the $\exists$-rule on $\exists S.A \in \mathcal{L}(x)$, resulting in a new node y with $\mathcal{L}(x,y) = \{S\}$ and $\mathcal{L}(y) = \{A\}$. Application of the $\forall$-rule to $\forall S.\neg A \in \mathcal{L}(x)$ yields $\mathcal{L}(y) \leftarrow \neg A$. We end up with $\mathcal{L}(y) = \{A, \neg A\}$, i.e. with a contradiction. The final tableau is depicted in Fig. 5.14.

5.3.4 Tableaux Algorithm for $\mathcal{SHIQ}$

The algorithm which we have presented in Section 5.3.3 displays the central concepts of tableaux algorithms for description logics. In order to convey an idea about the modifications to the $\mathcal{ALC}$ algorithm which need to be made to generalize it to more expressive description logics, we now provide the tableaux algorithm for $\mathcal{SHIQ}$.

We give a self-contained presentation of the $\mathcal{SHIQ}$ tableaux algorithm, and at the same time discuss the differences to the $\mathcal{ALC}$ algorithm.

A tableau for a $\mathcal{SHIQ}$ knowledge base consists of

- a set of nodes, labeled with individual names or variable names,
- directed edges between some pairs of nodes,
- for each node labeled x, a set $\mathcal{L}(x)$ of class expressions,
- for each pair of nodes x and y, a set $\mathcal{L}(x,y)$ containing role names or inverses of role names, and

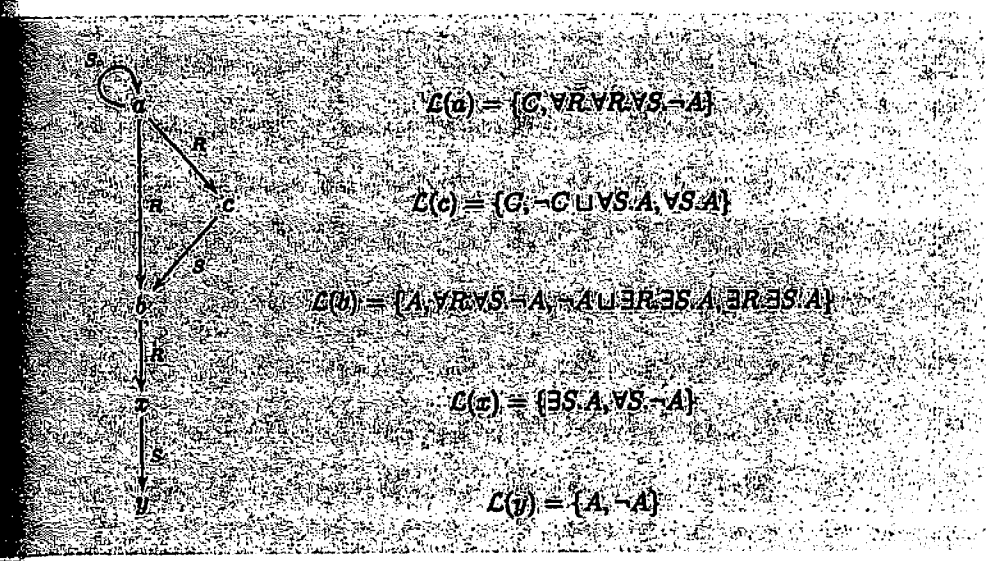


FIGURE 5.14: Final tableau for the example from Section 5.3.3.4.3

- two relations between nodes, denoted by $\approx$ and $\not\approx$.

The relations $\approx$ and $\not\approx$ are implicitly assumed to be symmetrical, i.e. whenever $x \approx y$, then $y \approx x$ also holds, and likewise for $\not\approx$. When we depict a tableau, we omit edges which are labeled with the empty set. We indicate the relations $\approx$ and $\not\approx$ by drawing undirected edges, labeled with $\approx$ or $\not\approx$.

There are only two differences to $\mathcal{ALC}$ tableaux. The first is that edges may be labeled with inverse role names, which accommodates the use of inverse roles in $\mathcal{SHIQ}$. The second is the presence of the relations $\approx$ and $\not\approx$, which are used to keep track of equality or inequality of nodes.[10] This accommodates the fact that $\mathcal{SHIQ}$, when translated to predicate logic as in Section 5.2.2, actually translates to predicate logic with equality. From the discussion in Section 5.2.2 it can easily be seen that $\mathcal{ALC}$ translates into equality-free predicate logic, so $\approx$ and $\not\approx$ are not needed for the $\mathcal{ALC}$ algorithm.

Given a $\mathcal{SHIQ}$ knowledge base K in negation normal form, the *initial tableau* for K is defined by the following procedure, which differs from the $\mathcal{ALC}$ procedure only in the final two steps.

1. For each individual a occurring in K, create a node labeled a and set $\mathcal{L}(a) = \emptyset$. These nodes are called *root nodes*.

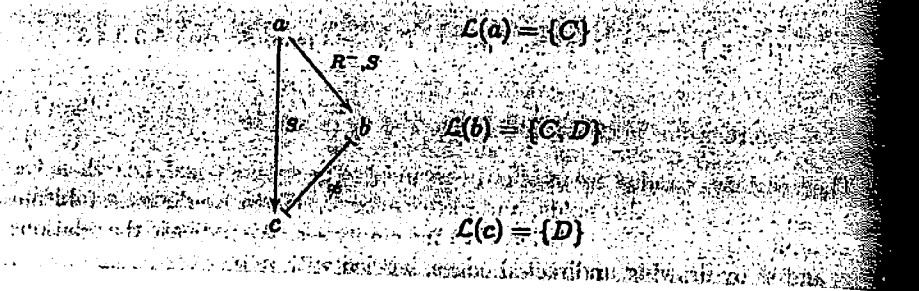
[10] In fact, the relation $\approx$ used for equality is not really needed. It can easily be removed from the algorithm description, where it occurs only once. We keep it, though, because it makes understanding the algorithm a bit easier.

2. For all pairs a, b of individuals, set $\mathcal{L}(a,b) = \emptyset$.
3. For each ABox statement $C(a)$ in K, set $\mathcal{L}(a) \leftarrow C$.
4. For each ABox statement $R(a,b)$ in K, set $\mathcal{L}(a,b) \leftarrow R$.
5. For each ABox statement $a \neq b$ in K, set $a \not\approx b$.
6. Set $\approx$ to be the empty relation, i.e. initially, no two nodes are considered to be equal.

Again, we make the agreement that $\top$ is contained in $\mathcal{L}(x)$, for any x, but we will often not write it down, and in fact the algorithm will not explicitly derive this.

We exemplify the visualization of initial tableaux by considering the knowledge base

$$K = \{R^-(a,b), S(a,b), S(a,c), c \neq b, C(a), C(b), D(b), D(c)\}.$$



For convenience, we use the following notation: If $R \in \mathbf{R}$ (i.e. if R is a role name), then set $\mathrm{Inv}(R) = R^-$ and $\mathrm{Inv}(R^-) = \mathrm{Inv}(R)$. Furthermore, call $R \in \mathbf{R}$ *transitive* if $R \circ R \sqsubseteq R$ or $\mathrm{Inv}(R) \circ \mathrm{Inv}(R) \sqsubseteq R$. This, and the following somewhat involved definitions, are needed to accommodate inverse roles.

Consider a tableau for a knowledge base K. Let H_K be the set of all statements of the form $R \sqsubseteq S$ and $\mathrm{Inv}(R) \sqsubseteq \mathrm{Inv}(S)$, where $R, S \in \mathbf{R}$ and $R \sqsubseteq S \in K$. We now call R a *subrole* of S if $R = S$, if $R \sqsubseteq S \in H_K$, or if there are $S_1, \ldots, S_n \in \mathbf{R}$ with $\{R \sqsubseteq S_1, S_1 \sqsubseteq S_2, \ldots, S_{n-1} \sqsubseteq S_n, S_n \sqsubseteq S\} \subseteq H_K$. In other words, R is a subrole of S if and only if R and S are related via the reflexive-transitive closure of $\sqsubseteq$ in H_K.

If $R \in \mathcal{L}(x,y)$ for two nodes x and y, and if R is a subrole of S, then y is called an *S-successor* of x, and x is called an *S-predecessor* of y. If y is an S-successor or an $\mathrm{Inv}(S)$-predecessor of x, then y is called an *S-neighbor* of x. Furthermore, inductively, every predecessor of x, which is not an individual, is called an *ancestor* of x, and every predecessor of an ancestor of x, which is not an individual, is also called an ancestor of x. Examples of these notions are given in Fig. 5.15.

Consider a knowledge base $H = \{R \sqsubseteq S^-, S \sqsubseteq S_1^-, S_1 \sqsubseteq S_2^-\}$ which consists only of a role hierarchy. Then $H_K = H \cup \{R^- \sqsubseteq S, S^- \sqsubseteq S_1, S_1^- \sqsubseteq S_2^-\}$. Furthermore, R is a subrole of S^-, S_1, and S_2. S is a subrole of S_1^- and S_2^-; S_1 is a subrole of S_2; R^- is a subrole of S, S_1^- and S_2^-, etc.
Now consider the knowledge base

$$K = \{R^-(a,b), R(b,c), S(c,d), S^-(d,e), S_1(e,f), R \sqsubseteq S\},$$

which, apart from $R \sqsubseteq S$, can be depicted by the following diagram.

$$a \xrightarrow{R^-} b \xrightarrow{R} c \xrightarrow{S} d \xrightarrow{S^-} e \xrightarrow{S_1} f$$

Then a is an R^--predecessor of b, and hence also an S^--predecessor of b. At the same time, c is an R-successor of b, and hence also an S-successor of b. We thus have that the S-neighbors of b are a and c. Also, we have that c is an S-predecessor of d and e is an S^--successor of d. Hence d has S^--neighbors c and e. Note that f has no ancestors because ancestors must not be individuals.

FIGURE 5.15: Example of notions of successor, predecessor, neighbor and ancestor

5.3.4.1 Blocking for $\mathcal{SHIQ}$

The blocking mechanism we used for $\mathcal{ALC}$ in Section 5.3.3.3 is not sufficient for $\mathcal{SHIQ}$, and we will give an example of this in Section 5.3.4.3.5 below. For $\mathcal{SHIQ}$, we need to employ *pairwise blocking*: While in $\mathcal{ALC}$, a node x is blocked by a node y if y essentially repeats x, in $\mathcal{SHIQ}$ a node x is blocked if it has a predecessor x' and there exists a node y with predecessor y', such that the *pair* (y', y) essentially repeats the pair (x', x). We formalize this as follows.

A node x is *blocked* if it is not a root node and any one of the following hold.

- There exist ancestors x', y, and y' of x such that
  - y is not a root node,
  - x is a successor of x' and y is a successor of y',
  - $\mathcal{L}(x) = \mathcal{L}(y)$ and $\mathcal{L}(x') = \mathcal{L}(y')$, and
  - $\mathcal{L}(x', x) = \mathcal{L}(y', y)$.
- An ancestor of x is blocked.
- There is no node y with $\mathcal{L}(y, x) \neq \emptyset$.

If only the first case applies, then x is called *directly blocked* by y. In all other cases, x is called *indirectly blocked*. Note that we require $\mathcal{L}(x) = \mathcal{L}(y)$ if x is (directly) blocked by y, which is stronger than the required $\mathcal{L}(x) \subseteq \mathcal{L}(y)$ in the case of $\mathcal{ALC}$.

As an example of blocking, consider the following part of a tableau.

$$a \xrightarrow{R} x \xrightarrow{R} y \xrightarrow{R} z \xrightarrow{R} w$$

$\mathcal{L}(a) = \{D\}$ $\mathcal{L}(x) = \{C\}$ $\mathcal{L}(y) = \{C\}$ $\mathcal{L}(z) = \{C\}$ $\mathcal{L}(w) = \{D\}$

Then z is directly blocked by y, since the pair (y, z) essentially repeats the pair (x, y). The node w is indirectly blocked because its ancestor z is blocked.

5.3.4.2 The Algorithm

The $\mathcal{SHIQ}$ tableaux algorithm is a nondeterministic algorithm which essentially extends the $\mathcal{ALC}$ algorithm. It decides whether a given knowledge base K is satisfiable.

Given a $\mathcal{SHIQ}$ knowledge base K, the tableaux algorithm first constructs the initial tableau as given above. Then the initial tableau is expanded by nondeterministically applying the rules from Figs. 5.16 and 5.17. The algorithm terminates if

- there is a node x such that $\mathcal{L}(x)$ contains a contradiction, i.e. if there is $C \in \mathcal{L}(x)$ and at the same time $\neg C \in \mathcal{L}(x)$,
- or there is a node x with $\leq nS.C \in \mathcal{L}(x)$, and x has $n+1$ S-neighbors $y_1, \ldots, y_{n+1}$ with $C \in \mathcal{L}(y_i)$ and $y_i \not\approx y_j$ for all $i, j \in \{1, \ldots, n+1\}$ with $i \neq j$,
- or none of the rules from Figs. 5.16 and 5.17 is applicable.

In the first two cases, we say that the tableau *contains a contradiction*. In the third case, we say that the tableau is *complete*. The knowledge base K is satisfiable if and only if there is a selection of subsequent expansion rule applications which leads to a complete and contradiction-free tableau. Otherwise, K is unsatisfiable.

5.3.4.3 Worked Examples

We continue with some worked examples which help to explain the different expansion rules.

5.3.4.3.1 Cardinalities We first give an example of the application of the $\leq$-rule. Consider the knowledge base

$$K = \{h(j,p), h(j,a), m(p), m(a), \leq 2h.\top(j)\},$$

$\sqcap$-rule: If x is not indirectly blocked, $C \sqcap D \in \mathcal{L}(x)$, and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

$\sqcup$-rule: If x is not indirectly blocked, $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \sqcap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

$\exists$-rule: If x is not blocked, $\exists R.C \in \mathcal{L}(x)$, and there is no y with $R \in \mathcal{L}(x, y)$ and $C \in \mathcal{L}(y)$, then

1. add a new node with label y (where y is a new node label),
2. set $\mathcal{L}(x, y) = \{R\}$ and $\mathcal{L}(y) = \{C\}$.

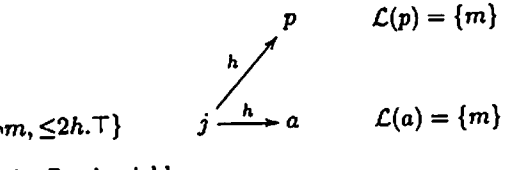
$\forall$-rule: If x is not indirectly blocked, $\forall R.C \in \mathcal{L}(x)$, and there is a node y with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

TBox-rule: If x is not indirectly blocked, C is a TBox statement, and $C \notin \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$.

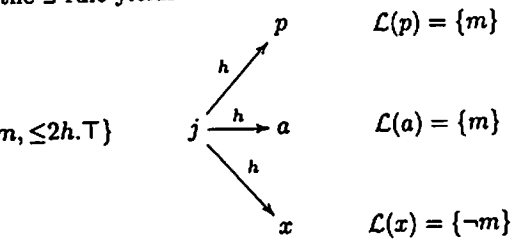
FIGURE 5.16: Expansion rules (part 1) for the $\mathcal{SHIQ}$ tableaux algorithm

which extends the example from Section 5.3.3.4.2 by adding the statement $\leq$2hasChild.$\top$(john). We will show that $\forall h.m(j)$ is still *not* a logical consequence of K.

As before, we add $\exists h.\neg m(j)$ to the knowledge base, resulting in K'. The initial tableau now looks as follows.



Now, application of the $\exists$-rule yields



and subsequent application of the $\leq$-root-rule allows us to identify p and a, resulting in the following.

trans-rule: If x is not indirectly blocked, $\forall S.C \in \mathcal{L}(x)$, S has a transitive subrole R, and x has an R-neighbor y with $\forall R.C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow \forall R.C$.

choose-rule: If x is not indirectly blocked, $\leq nS.C \in \mathcal{L}(x)$ or $\geq nS.C \in \mathcal{L}(x)$, and there is an S-neighbor y of x with $\{C, \text{NNF}(\neg C)\} \cap \mathcal{L}(y) = \emptyset$, then set $\mathcal{L}(y) \leftarrow C$ or $\mathcal{L}(y) \leftarrow \text{NNF}(\neg C)$.

$\geq$-rule: If x is not blocked, $\geq nS.C \in \mathcal{L}(x)$, and there are no n S-neighbors $y_1, \ldots, y_n$ of x with $C \in \mathcal{L}(y_i)$ and $y_i \not\approx y_j$ for $i, j \in \{1, \ldots, n\}$ and $i \neq j$, then

1. create n new nodes with labels $y_1, \ldots, y_n$ (where the labels are new),
2. set $\mathcal{L}(x, y_i) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \not\approx y_j$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

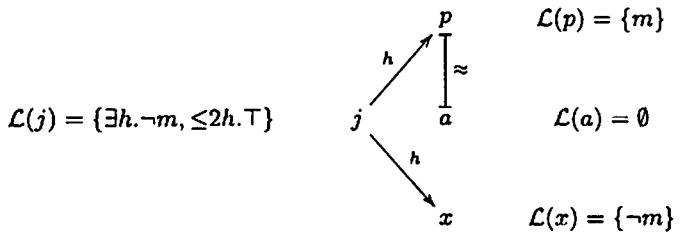
$\leq$-rule: If x is not indirectly blocked, $\leq nS.C \in \mathcal{L}(x)$, there are more than n S-neighbors y_i of x with $C \in \mathcal{L}(y_i)$, and x has two S-neighbors y, z such that y is neither a root node nor an ancestor of z, $y \not\approx z$ does not hold, and $C \in \mathcal{L}(y) \cap \mathcal{L}(z)$, then

1. set $\mathcal{L}(z) \leftarrow \mathcal{L}(y)$,
2. if z is an ancestor of x, then $\mathcal{L}(z, x) \leftarrow \{\text{Inv}(R) \mid R \in \mathcal{L}(x, y)\}$,
3. if z is not an ancestor of x, then $\mathcal{L}(x, z) \leftarrow \mathcal{L}(x, y)$,
4. set $\mathcal{L}(x, y) = \emptyset$, and
5. set $u \not\approx z$ for all u with $u \not\approx y$.

$\leq$-root-rule: If $\leq nS.C \in \mathcal{L}(x)$, there are more than n S-neighbors y_i of x with $C \in \mathcal{L}(y_i)$, and x has two S-neighbors y, z which are both root nodes, $y \not\approx z$ does not hold, and $C \in \mathcal{L}(y) \cap \mathcal{L}(z)$, then

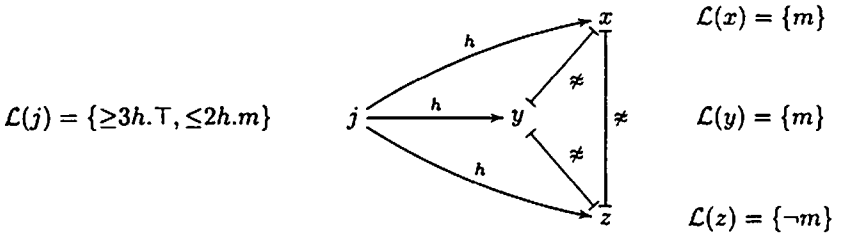
1. set $\mathcal{L}(z) \leftarrow \mathcal{L}(y)$,
2. for all directed edges from y to some w, set $\mathcal{L}(z, w) \leftarrow \mathcal{L}(y, w)$,
3. for all directed edges from some w to y, set $\mathcal{L}(w, z) \leftarrow \mathcal{L}(w, y)$,
4. set $\mathcal{L}(y) = \mathcal{L}(w, y) = \mathcal{L}(y, w) = \emptyset$ for all w,
5. set $u \not\approx z$ for all u with $u \not\approx y$, and
6. set $y \approx z$.

FIGURE 5.17: Expansion rules (part 2) for the $\mathcal{SHIQ}$ tableaux algorithm



None of the expansion rules is now applicable, so the tableau is complete and K' is satisfiable.

5.3.4.3.2 Choose We do a variant of the example just given in order to show how the choose-rule is applied. Consider the knowledge base $K = \{\geq 3h.\top(j), \leq 2h.m\}$. We want to find out if K is satisfiable. The initial tableau for K consists of a single node j with $\mathcal{L}(j) = \{\geq 3h.\top, \leq 2h.m\}$. Application of the $\geq$-rule yields three new nodes x, y and z with $\mathcal{L}(x) = \mathcal{L}(y) = \mathcal{L}(z) = \{\top\}$, $x \not\approx y$, $x \not\approx z$, and $y \not\approx z$. The choose-rule then allows us to assign classes to these new nodes, e.g., by setting $\mathcal{L}(x) \leftarrow m$, $\mathcal{L}(y) \leftarrow m$, and $\mathcal{L}(z) \leftarrow \neg m$. The resulting tableau, depicted below, is complete.



5.3.4.3.3 Inverse Roles The next example displays the handling of inverse roles. Consider the knowledge base $K = \{\exists C.h(j), \neg h \sqcup \forall P.h, C \sqsubseteq P^-\}$, which stands for

$$\exists \texttt{hasChild.Human(john)}$$
$$\texttt{Human} \sqsubseteq \forall \texttt{hasParent.Human}$$
$$\texttt{hasChild} \sqsubseteq \texttt{hasParent}^-$$

We show that Human(john) is a logical consequence from K, i.e. we start by adding $\neg h(j)$ to K, which is already in negation normal form.

In the initial tableau, we apply the $\exists$-rule to $\exists C.h \in \mathcal{L}(j)$, which yields the following.

$$\mathcal{L}(j) = \{\exists C.h, \neg h\} \qquad j \xrightarrow{C} x \qquad \mathcal{L}(x) = \{h\}$$

We now use the TBox rule and set $\mathcal{L}(x) \leftarrow \neg h \sqcup \forall P.h$. The $\sqcup$-rule on this yields $\mathcal{L}(x) \leftarrow \forall P.h$, since the addition of $\neg h$ would yield a contradiction.

$$\mathcal{L}(j) = \{\exists C.h, \neg h\} \qquad j \xrightarrow{C} x \qquad \mathcal{L}(x) = \{h, \neg h \sqcup \forall P.h, \forall P.h\}$$

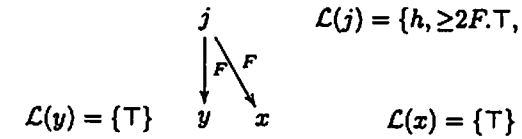
We now apply the $\forall$-rule to $\forall P.h \in \mathcal{L}(x)$: j is a C-predecessor of x, and hence a P^--predecessor of x due to $C \sqsubseteq P^-$. So j is a P^--neighbor of x, and the $\forall$-rule yields $\mathcal{L}(j) \leftarrow h$. Since we already have $\neg h \in \mathcal{L}(j)$, the algorithm terminates with the tableau containing a contradiction.

$$\mathcal{L}(j) = \{\exists C.h, \neg h, h\} \qquad j \xrightarrow{C} x \qquad \mathcal{L}(x) = \{h, \neg h \sqcup \forall P.h, \forall P.h\}$$

5.3.4.3.4 Transitivity and Blocking The next example displays blocking and the effect of the trans-rule. Consider the knowledge base $K = \{h \sqsubseteq \exists F.\top, F \sqsubseteq A, \forall A.h(j), h(j), \geq F.\top(j), A \circ A \sqsubseteq A\}$, which stands for the following.

$$\texttt{Human} \sqsubseteq \exists \texttt{hasFather}.\top$$
$$\texttt{hasFather} \sqsubseteq \texttt{hasAncestor}$$
$$\forall \texttt{hasAncestor}.\texttt{Human}(\texttt{john})$$
$$\texttt{Human}(\texttt{john})$$
$$\geq 2\texttt{hasFather}.\top(\texttt{john})$$
$$\texttt{hasAncestor} \circ \texttt{hasAncestor} \sqsubseteq \texttt{hasAncestor}$$

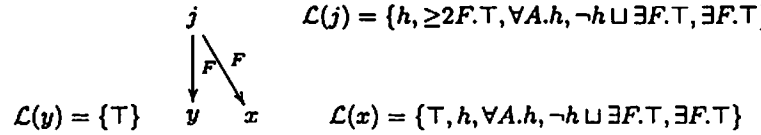
Since the knowledge base states that john has at least two fathers, we attempt to show unsatisfiability of K, which will not be possible.[11] We first get $\text{NNF}(K) = \{\neg h \sqcup \exists F.\top, F \sqsubseteq A, \forall A.h(j), h(j), \geq F.\top(j), A \circ A \sqsubseteq A\}$. From the initial tableau, we apply the $\geq$-rule to $\geq 2F.\top$, which results in the following tableau.



We now perform the following steps.

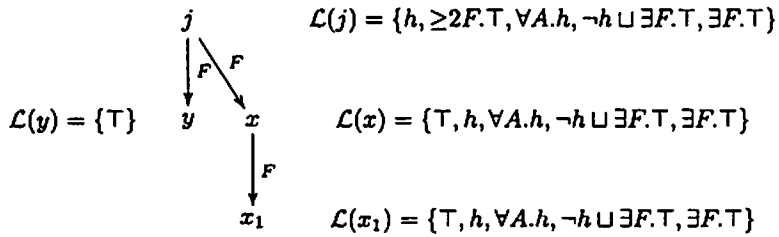
1. Apply the TBox-rule and set $\mathcal{L}(j) = \{\neg h \sqcup \exists F.\top\}$.
2. Apply the $\sqcup$-rule to the axiom just added, which yields $\mathcal{L}(j) \leftarrow \exists F.\top$ because adding $\neg h$ would result in a contradiction.
3. Apply the $\forall$-rule to $\forall A.h \in \mathcal{L}(j)$, which yields $\mathcal{L}(x) \leftarrow h$.
4. Apply the trans-rule to $\forall A.h \in \mathcal{L}(j)$, setting $\mathcal{L}(x) \leftarrow \forall A.h$.
5. Apply the TBox-rule and set $\mathcal{L}(x) \leftarrow \neg h \sqcup \exists F.\top$.
6. Apply the $\sqcup$-rule to the axiom just added, which yields $\mathcal{L}(x) \leftarrow \exists F.\top$ because adding $\neg h$ would result in a contradiction.

[11] There is no information in the knowledge base which forbids anybody having two fathers.

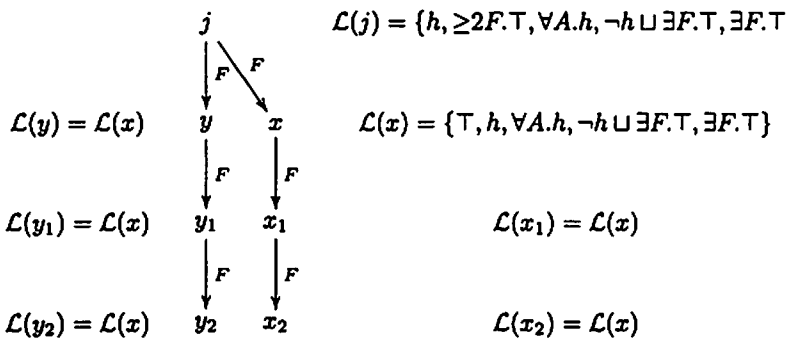


We can now perform the following steps.

7. Apply the $\exists$-rule to $\exists F.\top \in \mathcal{L}(x)$, creating a new node x_1 with $\mathcal{L}(x_1) = \top$.
8. Apply the $\forall$-rule to $\forall A.h \in \mathcal{L}(x)$, resulting in $\mathcal{L}(x_1) \leftarrow h$.
9. Apply the TBox rule and set $\mathcal{L}(x_1) \leftarrow \neg h \sqcup \exists F.\top$.
10. Apply the $\sqcup$-rule to the axiom just added, which yields $\mathcal{L}(x_1) \leftarrow \exists F.\top$ because adding $\neg h$ would result in a contradiction.



Note that $\mathcal{L}(x_1) = \mathcal{L}(x)$, so we can apply steps 7 to 10 to x_1 in place of x, creating a node x_2 in step 7. Likewise, we can do for y exactly what we have done for x, starting at step 1, creating two new nodes y_1 and y_2 in the process. The resulting tableau is as follows.



At this stage, x_2 is directly blocked by x_1 since the pair (x_1, x_2) repeats the pair (x, x_1). Likewise, y_2 is directly blocked by y_1 since the pair (y_1, y_2) repeats the pair (y, y_1). There is no expansion rule applicable, so the tableau is complete, showing that K is satisfiable.

5.3.4.3.5 Why We Need Pairwise Blocking The next example shows that the more complicated pairwise blocking is indeed needed for $\mathcal{SHIQ}$.

Consider the knowledge base K consisting of the statements $R \circ R \sqsubseteq R$, $F \sqsubseteq R$, and $\neg C \sqcap (\leq 1F) \sqcap \exists F^-.D \sqcap \forall R^-.(\exists F^-.D)(a)$, where D is short for the class expression $C \sqcap (\leq 1F) \sqcap \exists F.\neg C$.

K is unsatisfiable, which is not easy to see by simply inspecting the knowledge base. So let us construct the tableau, which will help us to understand the knowledge base. From the initial tableau, we repeatedly apply the $\sqcap$-rule to break down the class expression in $\mathcal{L}(a)$. Then we apply the $\exists$-rule to $\exists F^-.D$, creating a node y with $\mathcal{L}(y) = \{D\}$. $D \in \mathcal{L}(y)$ can be broken down by applying the $\sqcap$-rule repeatedly. Applying the $\forall$-rule to $\forall R^-.(\exists F^-.D) \in \mathcal{L}(a)$ yields $\mathcal{L}(y) \leftarrow \exists F^-.D$ due to $F \sqsubseteq R$, and the trans-rule applied to $\forall R^-.(\exists F^-.D) \in \mathcal{L}(a)$ yields $\mathcal{L}(y) \leftarrow \forall R^-.(\exists F^-.D)$. The following picture shows the current state; note that we have omitted some elements of $\mathcal{L}(a)$ and $\mathcal{L}(y)$.

a $\quad \mathcal{L}(a) \supseteq \{\neg C, \leq 1F, \exists F^-.D, \forall R^-.(\exists F^-.D)\}$

$\downarrow F^-$

y $\quad \mathcal{L}(y) \supseteq \{D, \exists F^-.D, \forall R^-.(\exists F^-.D), C, \leq 1F, \exists F.\neg C\}$

Similar arguments applied to y instead of a leave us with a new node z and the following situation.

a $\quad \mathcal{L}(a) \supseteq \{\neg C, \leq 1F, \exists F^-.D, \forall R^-.(\exists F^-.D)\}$

$\downarrow F^-$

y $\quad \mathcal{L}(y) \supseteq \{D, \exists F^-.D, \forall R^-.(\exists F^-.D), C, \leq 1F, \exists F.\neg C\}$

$\downarrow F^-$

z $\quad \mathcal{L}(z) = \mathcal{L}(y)$

Since the $\mathcal{SHIQ}$ tableau requires pairwise blocking, the node z is not blocked in this situation. If it were blocked, then the tableau would be complete, and K would be satisfiable. Since z is not blocked, however, we can expand $\exists F.\neg C \in \mathcal{L}(z)$ via the $\exists$-rule, creating a new node x with $\mathcal{L}(x) = \{\neg C\}$. Application of the $\leq$-rule to $\leq 1F \in \mathcal{L}(z)$ forces us to identify y and x, and yields the following.

a $\quad \mathcal{L}(a) \supseteq \{\neg C, \leq 1F, \exists F^-.D, \forall R^-.(\exists F^-.D)\}$

$\downarrow F^-$

$x \overset{\approx}{\longmapsto} y$ $\quad \mathcal{L}(y) \supseteq \{D, \exists F^-.D, \forall R^-.(\exists F^-.D), C, \leq 1F, \exists F.\neg C, \neg C\}$

$\downarrow F^-$

z $\quad \mathcal{L}(z) \supseteq \{D, \exists F^-.D, \forall R^-.(\exists F^-.D), C, \leq 1F, \exists F.\neg C\}$

Since $\{C, \neg C\} \subseteq \mathcal{L}(y)$, the tableau contains a contradiction and the algorithm terminates.

description logic	combined complexity	data complexity
$\mathcal{ALC}$	ExpTime-complete	NP-complete
$\mathcal{SHIQ}$	ExpTime-complete	NP-complete
$\mathcal{SHOIN}(\mathbf{D})$	NExpTime-complete	NP-hard
$\mathcal{SROIQ}$	N2ExpTime-complete	NP-hard
$\mathcal{EL}^{++}$	P-complete	P-complete
DLP	P-complete	P-complete
DL-Lite	In P	In LOGSPACE

FIGURE 5.18: Worst-case complexity classes of some description logics

5.3.5 Computational Complexities

Considerations of computational complexities of reasoning with various description logics have been a driving force in their development.[12] The rationale behind this is that understanding the computational complexity of a knowledge representation language aids avoiding language constructs which are too expensive to deal with in practice. This is an arguable position, and objections against the emphasis on computational complexity by description logic developers has been criticized from application perspectives. Nevertheless, it appears that the approach has been successful in the sense that it has indeed helped to produce paradigms with a favorable trade-off between expressivity and scalability. Complexities of description logics, more precisely of the underlying decision problems, are usually measured in terms of the size of the knowledge base. This is sometimes called the *combined complexity* of a description logic. If complexity is measured in terms of the size of the ABox only, then it is called the *data complexity* of the description logic. These notions are in analogy to database theory.

Figure 5.18 lists the complexity classes for the most important description logics mentioned in this chapter. It should be noted that despite the emphasis on complexity issues in developing description logics, their complexities are very high, usually exponential or beyond. This means that reasoning even with relatively small knowledge bases could prove to be highly intractable in the worst case. However, this is not a fault of the design of description logics: Dealing with complex logical knowledge is inherently difficult.

At the same time, it turns out that average-case complexity, at least for real existing knowledge bases, is not so bad, and state of the art reasoning systems, as discussed in Section 8.5, can deal with knowledge bases of considerable size. Such performance relies mainly on optimization techniques and intelligent

[12] Introducing complexity theory is beyond the scope of this book. See [Pap94] for a comprehensive overview.

heuristics which can be added to tableau reasoners in order to improve their performance on real data.

5.4 Summary

In this chapter we have presented the logical underpinnings of OWL. We have introduced description logics and explained their formal semantics. In particular, we have given two alternative but equivalent ways of describing the formal semantics of $\mathcal{SROIQ}$, and therefore of OWL DL and of OWL 2 DL, namely the direct extensional model-theoretic semantics, and the predicate logic semantics which is obtained by a translation to first-order predicate logic with equality.

We then moved on to discuss the major paradigm for automated reasoning in OWL, namely tableaux algorithms. We have formally specified the algorithms for $\mathcal{ALC}$ and $\mathcal{SHIQ}$. We have also given many examples explaining the algorithms, and briefly discussed issues of computational complexity for description logics.

5.5 Exercises

Exercise 5.1 Translate the ontology which you created as a solution for Exercise 4:1 into DL syntax.

Exercise 5.2 Translate the ontology which you created as a solution for Exercise 4.1 into predicate logic syntax.

Exercise 5.3 Express the following sentences in $\mathcal{SROIQ}$, using the individual names `bonnie` and `clyde`, the class names `Honest` and `Crime`, and the role names `reports`, `commits`, `suspects`, and `knows`.

1. Everybody who is honest and commits a crime reports himself.
2. Bonnie does not report Clyde.
3. Clyde has committed at least 10 crimes.
4. Bonnie and Clyde have committed at least one crime together.
5. Everybody who knows a suspect is also a suspect.

Exercise 5.4 Translate the knowledge base

$$\texttt{Human} \sqsubseteq \exists \texttt{hasMother.Human}$$
$$\exists \texttt{hasMother.}(\exists \texttt{hasMother.Human}) \sqsubseteq \texttt{Grandchild}$$
$$\texttt{Human(anupriyaAnkolekar)}$$

into RDFS syntax.

Exercise 5.5 Validate the logical inferences drawn in Fig. 4.11 by arguing with extensional semantics.

Exercise 5.6 Consider the two RDFS triples

`r rdfs:domain B .` and `A rdfs:subClassOf B .`

Understood as part of an OWL knowledge base, they can be expressed as $B \sqsubseteq \forall r.\top$ and $A \sqsubseteq B$.

Give a triple which is RDFS-entailed by the two given triples, but which cannot be derived from the OWL DL semantics.

Furthermore, give an OWL DL statement which is a logical consequence of the two OWL statements but cannot be derived using the RDFS semantics.

Exercise 5.7 Show using the $\mathcal{ALC}$ tableaux algorithm that the knowledge base

$$\texttt{Student} \sqsubseteq \exists \texttt{attends.Lecture}$$
$$\texttt{Lecture} \sqsubseteq \exists \texttt{attendedBy.}(\texttt{Student} \sqcap \texttt{Eager})$$
$$\texttt{Student(aStudent)}$$
$$\neg\texttt{Eager(aStudent)}$$

is satisfiable.

Exercise 5.8 Show using the $\mathcal{ALC}$ tableaux algorithm that $(\exists r.E)(a)$ is a logical consequence of the knowledge base $K = \{C(a), C \sqsubseteq \exists r.D, D \sqsubseteq E \sqcup F, F \sqsubseteq E\}$.

Exercise 5.9 Show using the $\mathcal{ALC}$ tableaux algorithm that the knowledge base $K = \{\neg H \sqcup \exists p.H, B(t), \neg H(t)\}$ is satisfiable.

Exercise 5.10 Validate the logical inferences drawn in Fig. 4.11 using the $\mathcal{ALC}$ tableaux algorithm.

Exercise 5.11 Show using the $\mathcal{ALC}$ tableaux algorithm that the following knowledge base is unsatisfiable.

$$\texttt{Bird} \sqsubseteq \texttt{Flies}$$
$$\texttt{Penguin} \sqsubseteq \texttt{Bird}$$
$$\texttt{Penguin} \sqcap \texttt{Flies} \sqsubseteq \bot$$
$$\texttt{Penguin(tweety)}$$

Exercise 5.12 Show using the $\mathcal{SHIQ}$ tableaux algorithm that the statement $\forall$hasChild.Male(john) is a logical consequence of the following knowledge base.

$$\texttt{hasChild(john, peter)}$$
$$\texttt{hasChild(john, alex)}$$
$$\texttt{Male(peter)}$$
$$\texttt{Male(alex)}$$
$$\leq 2\texttt{hasChild.Male(john)}$$
$$\texttt{peter} \neq \texttt{alex}$$

Exercise 5.13 Show using the $\mathcal{SHIQ}$ tableaux algorithm that the statement $\geq$2hasChild.$\top$(john) is a logical consequence of the following knowledge base.

$$\geq 2\texttt{hasSon.}\top\texttt{(john)}$$
$$\texttt{hasSon} \sqsubseteq \texttt{hasChild}$$

5.6 Further Reading

[HHPS04] is the normative document for the semantics of OWL 1, while [MPSCG09] is the current version describing the semantics of the forthcoming OWL 2 DL.

The Description Logic Handbook [BCM+07] is a comprehensive reference for description logics.

[HPSvH03] gives an overview of OWL 1 in relation to RDF and $\mathcal{SHIQ}$.

The $\mathcal{SHIQ}$ tableaux algorithms have been introduced in [HST00, HS99]. Our presentation differs slightly for didactic purposes, but there is no substantial difference.

A tableaux algorithm for $\mathcal{SHOIQ}$ can be found in [HS07]. Nominals basically add another element of nondeterminism which is very difficult to deal with efficiently in automated reasoning systems.

$\mathcal{SROIQ}$ as an extension of OWL DL was proposed in [HKS06]. The extensions are uncritical in terms of realization in tableaux algorithms; in this sense, $\mathcal{SROIQ}$ is only a minor extension of $\mathcal{SHOIQ}$.

$\mathcal{EL}^{++}$ was introduced in [BBL05] and has recently sparked a considerable interest in studying polynomial description logics.

DL-Lite is covered in [CGL+07].

For DLP, see [GHVD03].

Complexities for many description logics, including appropriate literature references, can be retrieved from `http://www.cs.man.ac.uk/~ezolin/dl/`.

Part III

Rules and Queries