Classical Algorithms for Reasoning and Explanation in Description Logics

Birte Glimm and Yevgeny Kazakov(✉)

The University of Ulm, Ulm, Germany
{birte.glimm,yevgeny.kazakov}@uni-ulm.de

Abstract. Description Logics (DLs) are a family of languages designed to represent conceptual knowledge in a formal way as a set of ontological axioms. DLs provide a formal foundation of the ontology language OWL, which is a W3C standardized language to represent information in Web applications. The main computational problem in DLs is finding relevant consequences of the information stored in ontologies, e.g., to answer user queries. Unlike related techniques based on keyword search or machine learning, the notion of a consequence is well-defined using a formal logic-based semantics. This course provides an in-depth description and analysis of the main reasoning and explanation methods for ontologies: tableau procedures and axiom pinpointing algorithms.

1 Introduction

It often happens that one needs to find some specific information on the Web. Information can be of different kinds: a local weather report, shop opening hours, a cooking recipe, or some encyclopedic information like the birth place of Albert Einstein. To search for this information, one usually enters some keywords into Web search engines and inspects Web pages that contain such keywords in the hope that the relevant information can be found there. The modern search engines are a little more advanced: they can also search for Web pages that use synonyms or related keywords, they use vocabularies of terms to structure information on the Web (e.g. schema.org[1]) or to disambiguate search results, they use some machine learning techniques to rank Web pages according to their likeliness of containing the relevant information, they use facts or knowledge (e.g., extracted from Wikipedia or from internal sources such as Google's Knowledge Graph[2]) to answer some queries, and they give direct answers for certain queries, e.g., Google.com directly answers the question "Who was US president in 2015?" with a knowledge panel for Barack Obama. In general, however, there is *no guarantee* that the searched piece of information can be found, even if a corresponding Web page exists.

Although an average Web user can live with such limitations, there are some critical applications in which incorrect or missed results cannot be tolerated. For

[1] https://schema.org/.
[2] https://developers.google.com/knowledge-graph/.

M. Krötzsch and D. Stepanova (Eds.): Reasoning Web 2019, LNCS 11810, pp. 1–64, 2019.
https://doi.org/10.1007/978-3-030-31423-1_1

example, if a medical patient is misdiagnosed, i.e., a correct diagnosis based on the description of the patient's symptoms is not found, the consequences can be severe. Several similar examples can be given for other domains ranging from banking to autonomous driving. Supporting such applications requires representing the knowledge in a *precise* and *unambiguous* way so that it can be correctly processed by *automated tools*.

Ontology languages, such as OWL [44], offer a solution to this problem by defining the formal syntax and semantics to describe and interpret expert knowledge. The basic principle of ontologies is similar to Wikipedia: instead of extracting the knowledge from general sources, such as Web pages, the knowledge is described and curated in one place by *domain experts*. The main difference between Wikipedia and ontologies is in the way how the knowledge is described. Wikipedia pages provide mostly textual (natural language) descriptions, which are easy to understand by humans, but difficult to process by computers. Today Wikipedia is also partly fed from Wikidata,[3] which provides a knowledge base of facts. The Wikidata project was founded in 2012 and contains, at the time of writing, facts about roughly 60 million entities. Ontologies go beyond a knowledge base of facts and provide often complex descriptions by means of *formulas*; each formula can use a limited set of constructors with well-defined meaning.

The main benefit of *formal* ontology languages such as OWL is that it is possible to answer questions by combining several sources of information. For example, if an ontology knows that 'Albert Einstein was a physicist who was born in Ulm,' and that 'Ulm is a city in Germany,' the ontology can answer questions like 'Which physicists were born in Germany?' by returning 'Albert Einstein' as one of the answers. That is, an answer to a question may be obtained not only from the *explicitly stated* information, but also from the (*implicit*) consequences.

This course is concerned with logic-based languages called *Description Logics* (short: DLs) that provide the formal foundation of the ontology language OWL. DLs are not just one language but a whole *family* of languages, designed to offer a great variety of choices for knowledge-based applications. Each language defines its own set of *constructors* that can be used to build the ontological formulas, called *axioms*. Each axiom describes a particular aspect of the real world; for example an axiom saying that 'Ulm is a city in Germany' describes a relation between the entities 'Ulm' and 'Germany'. The goal of the ontology is to provide axioms that describe (a part of) the real world as accurately as needed for an application. Since the real world is extremely complex, each of these descriptions are necessarily *incomplete*, which means that they can be also *satisfied* in situations that are different from the real world. The semantics of DL defines an abstract notion of a *model* to represent such situations. If an axiom holds in all such models, it is said to be a *logical consequence* of the ontology.

Of course, ontologies cannot answer questions on their own; they require special programs that can analyze and combine axioms to obtain the answers, called *ontology reasoners*. Ontology reasoners are usually able to solve several types of *reasoning problems*, such as checking if there are logical contradictions

[3] https://www.wikidata.org.

in the ontology or finding logical consequences of a certain form. To be practically useful, the *reasoning algorithms* implemented in ontology reasoners need to possess certain formal properties. An algorithm is *sound* if any answer that it returns is correct. If the algorithm returns every correct answer, it is *complete*. An algorithm might not return any answer if it does not *terminate*. If an algorithm always terminates, it is also useful to know how long one needs to wait for the answer. This can be measured using a notion of *algorithmic complexity*. Generally, algorithms with lower complexity should be preferred.

Modern ontology reasoners use very sophisticated and highly optimized algorithms for obtaining reasoning results, and often such results are counter-intuitive or hard to understand for humans. Reasoning results may also be incorrect, which indicates that some axioms in the ontology must have errors. In order to improve the user experience when working with ontologies, and, in particular, facilitate ontology *debugging*, ontology development tools include capabilities for *explaining* reasoning results.

Almost every year since its inception, the Reasoning Web Summer School offered lectures focused on different topics of reasoning in DLs ranging from overview courses [2,49,50,65] to more specialized topics such as lightweight DLs [64], query answering [10,36,43], and non-standard reasoning problems [9,12, 59]. The purpose of this course is to provide a deeper understanding of the key reasoning and explanation algorithms used in DLs. We provide a detailed account on *tableau procedures*, which are the most popular reasoning procedures for DLs, including questions such as soundness, completeness, termination, and complexity. For explaining the reasoning results we look into general-purpose *axiom pinpointing procedures* that can efficiently identify some or all subsets of axioms responsible for a reasoning result. Some of these procedures can be also used to *repair* unintended entailments by identifying possible subsets of axioms whose removal breaks the entailment.

This paper is (partly) based on the course "Algorithms for Knowledge Representation" given at the University of Ulm, and includes full proofs, detailed examples, and simple exercises. The material should be accessible to students of the general university (bachelor) level in technical subjects such as, but not limited to, computer science. All relevant background, such as the basics of the computational complexity theory is introduced as needed. The course should be of particular interest to those who are interested in developing and implementing (DL) reasoning procedures. Since the duration of this course is limited to two lectures, we mainly focus on the *basic description logic* $\mathcal{ALC}$, to nevertheless provide a detailed account on the topics of DL reasoning and explanation. For this reason, this course does not provide a comprehensive literature survey. For the latter, we refer the reader to previous overview courses [2,49,50,65], DL textbooks [4,6], PhD theses [26,60] and some recent overviews [45].

2 Description Logics

Description Logics (DLs) are specialized logic-based languages designed to represent conceptual knowledge in a machine-readable form so that this information

can be processed by automated tools. Most DLs correspond to decidable fragments of first-order logic, which is a very expressive general-purpose language, however, with undecidable standard reasoning problems. Decidability has been one of the key requirements for the development of DL languages; to achieve decidability, the languages are often restricted to contain the features most essential for knowledge representation. For example, in natural language, one rarely speaks about more than two objects at a time. For this reason, DLs usually restrict the syntax to only unary and binary relations and to constants. Unary relations usually specify types (or classes) of objects and are called *concepts* in DLs (and *classes* in OWL). Binary relations specify how objects are related to each other, and are called *roles* in DLs (and *properties* in OWL). Constants refer to particular objects by their names. In DLs (and OWL) they are called *individuals*. In this paper, we mainly focus on the *basic description logic* $\mathcal{ALC}$ [51], which is regarded by some as the smallest sensible language for knowledge representation. This language traditionally serves not only as the basis of more expressive languages, but also as a relatively simple example on which the main ideas about reasoning in DLs can be explained.

2.1 Syntax

The *vocabulary* of the DL $\mathcal{ALC}$ consists of countably-infinite sets N_C of *concept names* (or *atomic concepts*), N_R of *role names* (or *atomic roles*), N_I of *individual names* (or *individuals*), *logical symbols*: $\top$, $\bot$, $\neg$, $\sqcap$, $\sqcup$, $\forall$, $\exists$, $\sqsubseteq$, $\equiv$, and *structural symbols*: '(', ')', '.'. These symbols are used to construct formulas that are called DL *axioms*.

Intuitively, (atomic) concepts are used to describe sets of objects. For example, we may introduce the following atomic concepts representing the respective sets of objects:

Human – the set of all human beings,
Male – the set of all male (not necessarily human) beings,
Country – the set of all countries.

Likewise, (atomic) roles represent binary relations between objects:

hasChild – the parent-child relation between objects,
hasLocation – a relation between objects and their (physical) locations.

Individuals represent some concrete object, for example:

germany – the country of Germany,
john – the person John.

In our examples, we usually use a *convention* for writing (atomic) concepts, roles, and individuals so that one can easily tell them apart: concepts are written starting with capital letters, while role and individual names start with a lower case letter. In addition, we reserve single letters (possibly with decorations) A, B for atomic concepts, r and s for (atomic) roles, a, b, c for individuals, and C, D, E for complex concepts, which are introduced next.

The logical symbols $\top, \bot, \neg, \sqcap, \sqcup, \forall$, and $\exists$ are used for constructing *complex concepts* (or just *concepts*). Just like for atomic concepts, complex concepts represent sets of objects, but these sets are uniquely determined by the *sub-concepts* from which they are constructed. The set of $\mathcal{ALC}$ *concepts* can be defined using the grammar definition:

$$C, D ::= A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \exists r.C \mid \forall r.C. \tag{1}$$

This definition means that the set of concepts (which are named by C and D) is recursively constructed starting from atomic concepts A, *top concept* $\top$, *bottom concept* $\bot$, by applying *conjunction* $C \sqcap D$, *disjunction* $C \sqcup D$, *negation* $\neg C$, *existential restriction* $\exists r.C$, and *universal restriction* $\forall r.C$. Intuitively, $\top$ represents the set of all objects of the modeled domain, $\bot$ the empty set of objects, $C \sqcap D$ the set of common objects of C and D, $C \sqcup D$ the union of objects in C and D, $\neg C$ all objects that are not in C, $\exists r.C$ all object that are related by r to *some* object in C, $\forall r.C$ all objects that are related by r to *only* objects in C. For example, one can construct the following $\mathcal{ALC}$-concepts:

$\mathsf{Male} \sqcap \mathsf{Human}$	– the set of all male humans,
$\mathsf{Dead} \sqcup \mathsf{Alive}$	– the union of all dead and all alive things,
$\neg\mathsf{Male}$	– the set of all non-male things,
$(\neg\mathsf{Male}) \sqcap \mathsf{Human}$	– the set of all non-male humans,
$\exists\mathsf{hasChild}.\mathsf{Male}$	– all things that have a male child,
$\forall\mathsf{hasChild}.\mathsf{Female}$	– all things that have only female children,
$\mathsf{Male} \sqcap (\forall\mathsf{hasChild}.\neg\mathsf{Male})$	– all male things all of whose children are not male.

Once complex concepts are constructed, they can be used to describe various properties by writing *axioms*. In the DL $\mathcal{ALC}$ we consider four possible types of axioms: a *concept inclusion* $C \sqsubseteq D$ states that every object of the concept C must be an object of the concept D, a *concept equivalence* $C \equiv D$ states that the concepts C and D must contain exactly the same objects, a *concept assertion* $C(a)$ states that the object represented by the individual a is an object of the concept C, and a *role assertion* $r(a, b)$ states that the objects represented by the individuals a and b are connected by the relation represented by the role r. Here are some examples of these axioms:

$\mathsf{Human} \sqsubseteq \mathsf{Dead} \sqcup \mathsf{Alive}$	– every human is either dead or alive,
$\mathsf{Parent} \equiv \exists\mathsf{hasChild}.\top$	– parents are exactly those that have some child,

$\mathsf{Male}(\mathsf{john})$ – John is a male,

$\mathsf{bornIn}(\mathsf{einstein}, \mathsf{ulm})$ – Albert Einstein was born in Ulm.

Axioms are usually grouped together to form *knowledge bases* (or *ontologies*). An $\mathcal{ALC}$ ontology $\mathcal{O}$ is simply a (possibly empty) set of $\mathcal{ALC}$ axioms. The axioms of an ontology are usually split into two parts: the *terminological part* (short: *TBox*) contains only concept inclusion and concept equivalence axioms, the *assertional part* (short: *ABox*) contains only concept and role assertion axioms. This distinction is often used to simplify the analysis of algorithms. For example, to answer questions about concepts, in many cases it is not necessary to consider the ABox, which is usually the larger part of an ontology.

Example 1. Consider the ontology $\mathcal{O}$ consisting of the following axioms:

1. $\mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top,$
2. $\mathsf{GrandParent} \equiv \exists \mathsf{hasChild}.\mathsf{Parent},$
3. $\mathsf{hasChild}(\mathsf{john}, \mathsf{mary}).$

Then the TBox of $\mathcal{O}$ consists of the first two axioms, and the ABox of $\mathcal{O}$ consists of the last axiom.

The main application of ontologies is to extract new information from the information explicitly stated in the ontologies. For example, from the first two axioms of the ontology $\mathcal{O}$ from Example 1 it follows that each grandparent must be a parent because each grandparent has a child (who happens to be a parent). This new information can be formalized using a concept inclusion axiom $\mathsf{GrandParent} \sqsubseteq \mathsf{Parent}$. Likewise, from the first and the last axiom of $\mathcal{O}$ one can conclude that the object represented by the individual john must be a parent because he has a child (mary). This piece of information can be formalized using a concept assertion axiom $\mathsf{Parent}(\mathsf{john})$. The two new axioms are said to be *logical consequences* of the ontology $\mathcal{O}$.

2.2 Semantics

To be able to calculate (preferably automatically) which axioms are logical consequences of ontologies and which are not, we need to define the *semantics* of ontologies. So far we have defined the *syntax* of ontologies, which describes how axioms in the ontologies can be constructed from various symbols. This information is not enough to understand the *meaning* of concepts and axioms. In fact, in $\mathcal{ALC}$ the same information can be described in many different ways. For example, the concept $\mathsf{Male} \sqcap \mathsf{Human}$ describes exactly the same set of objects as the concept $\mathsf{Human} \sqcap \mathsf{Male}$. The axiom $\mathsf{Human} \sqsubseteq \mathsf{Dead} \sqcup \mathsf{Alive}$ describes exactly the same situation as the axiom $\mathsf{Human} \sqcap (\neg \mathsf{Dead}) \sqsubseteq \mathsf{Alive}$. The formal semantics describes how to determine the meaning of concepts and axioms, while abstracting from the particular syntactic ways in which they are written down.

Like in many other logic-based formalisms (including propositional and first-order logic), the semantics of description logics is defined using (Tarski-style

set-theoretic) interpretations. Intuitively, an interpretation describes a possible state of the world modeled by the ontology. Formally, an *interpretation* is a pair $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta^\mathcal{I}$ is a non-empty set called the *domain* of $\mathcal{I}$ and $\cdot^\mathcal{I}$ is an *interpretation function* that assigns to each atomic concept $A \in N_C$ a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, to each atomic role $r \in N_R$ a binary relation $r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and to each individual $a \in N_I$ an element $a^\mathcal{I} \in \Delta^\mathcal{I}$. Intuitively, the domain $\Delta^\mathcal{I}$ represents the objects that can be part of the modeled world; this can be an infinite (and even an uncountable) set, but it must contain at least one element because otherwise it is not possible to assign $a^\mathcal{I} \in \Delta^\mathcal{I}$ for $a \in N_I$. Although the interpretation function requires an assignment for every symbol of the vocabulary (and there are infinitely many available symbols in N_C, N_R, and N_I), when defining interpretations for ontologies, we usually provide the values only for the symbols present in the ontology, assuming that all other symbols are interpreted in an arbitrary way.

Example 2. We can define an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ of the symbols appearing in the ontology $\mathcal{O}$ from Example 1, for example, as follows:

- $\Delta^\mathcal{I} = \{a, b, c\}$,
- $\mathsf{Parent}^\mathcal{I} = \{a, b\}$, $\mathsf{GrandParent}^\mathcal{I} = \{a\}$,
- $\mathsf{hasChild}^\mathcal{I} = \{\langle a, b\rangle, \langle b, c\rangle\}$,
- $\mathsf{john}^\mathcal{I} = a$, $\mathsf{mary}^\mathcal{I} = b$.

Once the interpretation is fixed, it can be *recursively extended* to complex $\mathcal{ALC}$ concepts according to the following rules that match the respective cases of the grammar definition (1). Assuming that the values of $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$ and $D^\mathcal{I} \subseteq \Delta^\mathcal{I}$ for concepts C and D have already been determined, the interpretations of concepts build from C and D can be computed as follows:

- $\top^\mathcal{I} = \Delta^\mathcal{I}$,
- $\bot^\mathcal{I} = \emptyset$,
- $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$,
- $(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$,
- $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$,
- $(\exists r.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \exists y : \langle x, y\rangle \in r^\mathcal{I} \;\;\&\;\; y \in C^\mathcal{I}\}$,
- $(\forall r.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \forall y : \langle x, y\rangle \in r^\mathcal{I} \;\Rightarrow\; y \in C^\mathcal{I}\}$.

The last two cases of this definition probably require some further clarifications. The interpretation of $\exists r.C$ contains exactly those elements $x \in \Delta^\mathcal{I}$ that are connected to *some* element y by a binary relation $r^\mathcal{I}$ (i.e., $\langle x, y\rangle \in r^\mathcal{I}$) such that $y \in C^\mathcal{I}$. The interpretation of $\forall r.C$ contains exactly those elements $x \in \Delta^\mathcal{I}$ such that *every* element y connected from x by $r^\mathcal{I}$ (i.e., for which $\langle x, y\rangle \in r^\mathcal{I}$ holds), is a member of $C^\mathcal{I}$ (i.e., $y \in C^\mathcal{I}$). In other words, x is $r^\mathcal{I}$-connected to *only* elements of $C^\mathcal{I}$. Importantly, if an element x does not have any $r^\mathcal{I}$-successor, i.e., $\langle x, y\rangle \in r^\mathcal{I}$ holds for no $y \in \Delta^\mathcal{I}$, then $x \notin (\exists r.C)^\mathcal{I}$ but $x \in (\forall r.C)^\mathcal{I}$ for every concept C.

Example 3. Consider the interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ from Example 2. Then:

- $\top^{\mathcal{I}} = \{a, b, c\}$,
- $(\mathsf{Parent} \sqcap \mathsf{GrandParent})^{\mathcal{I}} = \{a\}$,
- $(\mathsf{Parent} \sqcup \mathsf{GrandParent})^{\mathcal{I}} = \{a, b\}$,
- $(\neg\mathsf{GrandParent})^{\mathcal{I}} = \{b, c\}$,
- $(\mathsf{Parent} \sqcap \neg\mathsf{GrandParent})^{\mathcal{I}} = \{b\}$,
- $(\exists\mathsf{hasChild}.\top)^{\mathcal{I}} = \{a, b\}$,
- $(\exists\mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}} = \{a\}$,
- $(\forall\mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}} = \{a, c\}$, (!!!)
- $(\forall\mathsf{hasChild}.\mathsf{GrandParent})^{\mathcal{I}} = \{c\}$, (!!!)
- $(\forall\mathsf{hasChild}.\forall\mathsf{hasChild}.\bot)^{\mathcal{I}} = \{b, c\}$.

We next define how to interpret $\mathcal{ALC}$ axioms. The purpose of axioms in an ontology is to describe the characteristics of concepts, roles, and individuals involved in these axioms. These properties hold in some interpretations and are violated in other interpretations. For an interpretation $\mathcal{I}$ and an axiom α we write $\mathcal{I} \models \alpha$ if α *holds* (or *is satisfied*) in $\mathcal{I}$, defined as follows:

- $\mathcal{I} \models C \sqsubseteq D$ if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $\mathcal{I} \models C \equiv D$ if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$,
- $\mathcal{I} \models C(a)$ if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$,
- $\mathcal{I} \models r(a, b)$ if and only if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$.

If it is not the case that $\mathcal{I} \models \alpha$, we write $\mathcal{I} \not\models \alpha$ and say that α is *violated* (or *not satisfied*) in $\mathcal{I}$. Table 1 summarizes the syntax and semantics of $\mathcal{ALC}$.

Table 1. The summary of syntax and semantics of the DL $\mathcal{ALC}$

	Syntax	Semantics	
Roles:			
atomic role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	(given)
Concepts:			
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	(given)
top	$\top$	$\Delta^{\mathcal{I}}$	
bottom	$\bot$	$\emptyset$	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
existential restriction	$\exists r.C$	$\{x \mid \exists y : \langle x, y \rangle \in r^{\mathcal{I}} \ \& \ y \in C^{\mathcal{I}}\}$	
universal restriction	$\forall r.C$	$\{x \mid \forall y : \langle x, y \rangle \in r^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$	
Individuals:			
individual	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$	(given)
Axioms:			
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	
concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$	
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	
role assertion	$r(a, b)$	$\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in r^{\mathcal{I}}$	

Example 4. Continuing Example 3, we can determine the interpretation of the following axioms in the defined $\mathcal{I}$:

- $\mathcal{I} \models \mathsf{GrandParent} \sqsubseteq \mathsf{Parent}$: $\mathsf{GrandParent}^{\mathcal{I}} = \{a\} \subseteq \mathsf{Parent}^{\mathcal{I}} = \{a, b\}$,
- $\mathcal{I} \not\models \mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$: $\mathsf{Parent}^{\mathcal{I}} = \{a, b\} \not\subseteq \mathsf{GrandParent}^{\mathcal{I}} = \{a\}$,
- $\mathcal{I} \models \exists \mathsf{hasChild}.\mathsf{GrandParent} \equiv \bot$: $(\exists \mathsf{hasChild}.\mathsf{GrandParent})^{\mathcal{I}} = \emptyset = \bot^{\mathcal{I}}$
- $\mathcal{I} \models (\exists \mathsf{hasChild}.\mathsf{Parent})(\mathsf{john})$: $\mathsf{john}^{\mathcal{I}} = a \in (\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}} = \{a\}$,
- $\mathcal{I} \not\models \mathsf{hasChild}(\mathsf{mary}, \mathsf{john})$: $\langle \mathsf{mary}^{\mathcal{I}}, \mathsf{john}^{\mathcal{I}} \rangle = \langle b, a \rangle \notin \mathsf{hasChild}^{\mathcal{I}} = \{\langle a, b \rangle\}$,
- $\mathcal{I} \models (\forall \mathsf{hasChild}.\neg \mathsf{Parent})(\mathsf{mary})$: $\mathsf{mary}^{\mathcal{I}} = b \in (\forall \mathsf{hasChild}.\neg \mathsf{Parent})^{\mathcal{I}} = \{b, c\}$.

As mentioned earlier, an axiom may hold in one interpretation, but may be violated in another interpretation. For example, the axiom $A \sqsubseteq B$ holds in $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = \{a\}$, $A^{\mathcal{I}} = B^{\mathcal{I}} = \emptyset$, but is violated in $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ with $\Delta^{\mathcal{J}} = \{a\}$, $A^{\mathcal{J}} = \{a\}$, and $B^{\mathcal{J}} = \emptyset$. There are, however, axioms that hold in *every* interpretation. We call such axioms *tautologies*.

Example 5. The following $\mathcal{ALC}$ axioms are tautologies because they hold in every interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$:

- $C \sqsubseteq C$: because $C^{\mathcal{I}} \subseteq C^{\mathcal{I}}$,
- $C \sqsubseteq \top$: because $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} = \top^{\mathcal{I}}$,
- $C \sqcap D \sqsubseteq C$: because $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$,
- $\forall r.\top \equiv \top$: because

$$(\forall r.\top)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \forall y : \langle x, y \rangle \in r^{\mathcal{I}} \;\Rightarrow\; y \in \top^{\mathcal{I}} = \Delta^{\mathcal{I}}\} = \Delta^{\mathcal{I}} = \top^{\mathcal{I}}.$$

- $\exists r.C \sqcap \forall r.D \sqsubseteq \exists r.(C \sqcap D)$: because

$$\begin{aligned}
(\exists r.C \sqcap \forall r.D)^{\mathcal{I}} &= (\exists r.C)^{\mathcal{I}} \cap (\forall r.D)^{\mathcal{I}} \\
&= \{x \in \Delta^{\mathcal{I}} \mid \exists y : \langle x, y \rangle \in r^{\mathcal{I}} \;\&\; y \in C^{\mathcal{I}}\} \cap \\
&\quad \{x \in \Delta^{\mathcal{I}} \mid \forall y : \langle x, y \rangle \in r^{\mathcal{I}} \Rightarrow y \in D^{\mathcal{I}}\} \\
&\subseteq \{x \in \Delta^{\mathcal{I}} \mid \exists y : \langle x, y \rangle \in r^{\mathcal{I}} \;\&\; y \in C^{\mathcal{I}} \;\&\; y \in D^{\mathcal{I}}\} \\
&= \{x \in \Delta^{\mathcal{I}} \mid \exists y : \langle x, y \rangle \in r^{\mathcal{I}} \;\&\; y \in C^{\mathcal{I}} \cap D^{\mathcal{I}} = (C \sqcap D)^{\mathcal{I}}\} \\
&= (\exists r.(C \sqcap D))^{\mathcal{I}}.
\end{aligned}$$

The following $\mathcal{ALC}$ axioms are not tautologies as they do not hold in at least one interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$:

- $C \sqsubseteq C \sqcap D$: Take $\Delta^{\mathcal{I}} = \{a\}$, $C^{\mathcal{I}} = \{a\}$, and $D^{\mathcal{I}} = \emptyset$. Then $C^{\mathcal{I}} = \{a\} \not\subseteq \emptyset = \{a\} \cap \emptyset = C^{\mathcal{I}} \cap D^{\mathcal{I}} = (C \sqcap D)^{\mathcal{I}}$.
- $\forall r.C \sqsubseteq \exists r.C$: Take $\Delta^{\mathcal{I}} = \{a\}$ and $C^{\mathcal{I}} = r^{\mathcal{I}} = \emptyset$. Then $(\forall r.C)^{\mathcal{I}} = \{a\} \not\subseteq \emptyset = (\exists r.C)^{\mathcal{I}}$.

– $\exists r.C \sqcap \exists r.D \sqsubseteq \exists r.(C \sqcap D)$: Take $\Delta^{\mathcal{I}} = \{a, c, d\}$, $C^{\mathcal{I}} = \{c\}$, $D^{\mathcal{I}} = \{d\}$, and $r^{\mathcal{I}} = \{\langle a, c\rangle, \langle a, d\rangle\}$. Then $a \in (\exists r.C)^{\mathcal{I}}$ because $\langle a, c\rangle \in r^{\mathcal{I}}$ and $c \in C^{\mathcal{I}}$. Similarly, $a \in (\exists r.D)^{\mathcal{I}}$ since $\langle a, d\rangle \in r^{\mathcal{I}}$ and $d \in D^{\mathcal{I}}$. But $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} = \{c\} \cap \{d\} = \emptyset$. Thus, $(\exists r.(C \sqcap D))^{\mathcal{I}} = \emptyset$. Hence, $(\exists r.C \sqcap \exists r.D)^{\mathcal{I}} = (\exists r.C)^{\mathcal{I}} \cap (\exists r.D)^{\mathcal{I}} = \{a\} \cap \{a\} = \{a\} \not\subseteq \emptyset = (\exists r.(C \sqcap D))^{\mathcal{I}}$.

Interpretations that satisfy the axioms in an ontology will be of special interest to us, because these interpretations agree with the requirements imposed by the axioms. These interpretations are called models. Formally, an interpretation $\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (in symbols: $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I} \models \alpha$ for every $\alpha \in \mathcal{O}$. We say that $\mathcal{O}$ is *satisfiable* if $\mathcal{O}$ has at least one model, i.e., if $\mathcal{I} \models \mathcal{O}$ holds for at least one interpretation $\mathcal{I}$. Otherwise, we say that $\mathcal{O}$ is *unsatisfiable*.

Example 6. Consider the ontology $\mathcal{O}$ containing the first two axioms from Example 1:

1. $\mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top,$
2. $\mathsf{GrandParent} \equiv \exists \mathsf{hasChild}.\mathsf{Parent}.$

We can prove that $\mathcal{O}$ is satisfiable by presenting a simple model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{O}$:

– $\Delta^{\mathcal{I}} = \{a\}$,
– $\mathsf{Parent}^{\mathcal{I}} = \mathsf{GrandParent}^{\mathcal{I}} = \mathsf{hasChild}^{\mathcal{I}} = \emptyset$,
– $\mathsf{john}^{\mathcal{I}} = \mathsf{mary}^{\mathcal{I}} = a$.

Note that $(\exists \mathsf{hasChild}.\top)^{\mathcal{I}} = \emptyset$ and $(\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}} = \emptyset$ since $\mathsf{hasChild}^{\mathcal{I}} = \emptyset$. Thus, $\mathsf{Parent}^{\mathcal{I}} = (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$ and $\mathsf{GrandParent}^{\mathcal{I}} = (\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}}$, which implies that $\mathcal{I}$ satisfies both axioms in $\mathcal{O}$.

Let us now extend $\mathcal{O}$ with the third axiom from Example 1:

3. $\mathsf{hasChild}(\mathsf{john}, \mathsf{mary}).$

The previous interpretation $\mathcal{I}$ is no longer a model of $\mathcal{O}$ since $\langle \mathsf{john}^{\mathcal{I}}, \mathsf{mary}^{\mathcal{I}}\rangle = \langle a, a\rangle \notin \emptyset = \mathsf{hasChild}^{\mathcal{I}}$. This does not, however, mean that the ontology $\mathcal{O}$ is unsatisfiable since we can find another interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ that satisfies all three axioms:

– $\Delta^{\mathcal{J}} = \{a\}$,
– $\mathsf{Parent}^{\mathcal{J}} = \mathsf{GrandParent}^{\mathcal{J}} = \{a\}, \quad \mathsf{hasChild}^{\mathcal{J}} = \{\langle a, a\rangle\}$,
– $\mathsf{john}^{\mathcal{J}} = \mathsf{mary}^{\mathcal{J}} = a$.

It is easy to verify that $(\exists \mathsf{hasChild}.\top)^{\mathcal{J}} = (\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{J}} = \{a\}$, which proves that $\mathcal{J}$ still satisfies the first two axioms. Since $\langle \mathsf{john}^{\mathcal{J}}, \mathsf{mary}^{\mathcal{J}}\rangle = \langle a, a\rangle \in \{\langle a, a\rangle\} = \mathsf{hasChild}^{\mathcal{J}}$, $\mathcal{J}$ now also satisfies the third axiom.

Besides the notion of satisfiability of ontologies, in description logics one also considers a notion of satisfiability of concepts. We say that a *concept C is*

satisfiable if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. For example, the concept $\forall r.\bot$ is satisfiable because $(\forall r.\bot)^{\mathcal{I}} = \Delta^{\mathcal{I}} \neq \emptyset$ for every interpretation $\mathcal{I}$ such that $r^{\mathcal{I}} = \emptyset$ (and there is certainly at least one such interpretation). On the other hand, the concept $\exists r.\bot$ is not satisfiable since $\bot^{\mathcal{I}} = \emptyset$ and, consequently, $(\exists r.\bot)^{\mathcal{I}} = \emptyset$ for every $\mathcal{I}$.

Sometimes the interpretation that should satisfy the concept is constrained to be a model of a given ontology. We say that a *concept C is satisfiable with respect to an ontology $\mathcal{O}$* if $C^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{O}$. Note that if the ontology $\mathcal{O}$ is not satisfiable, then no concept C is satisfiable with respect to this ontology.

Example 7. Let $\mathcal{O} = \{A \sqsubseteq \neg A\}$. Note that $\mathcal{O}$ is satisfiable in any interpretation $\mathcal{I}$ such that $A^{\mathcal{I}} = \emptyset$ since $A^{\mathcal{I}} = \emptyset \subseteq (\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus \emptyset = \Delta^{\mathcal{I}}$. However, the concept A is not satisfiable w.r.t. $\mathcal{O}$. Indeed, assume that $A^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I}$. Then $a \in A^{\mathcal{I}}$ for some domain element $a \in \Delta^{\mathcal{I}}$. But then $a \notin \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} = (\neg A)^{\mathcal{I}}$. Hence, $A^{\mathcal{I}} \not\subseteq (\neg A)^{\mathcal{I}}$. Thus, $\mathcal{I} \not\models A \sqsubseteq \neg A$. Hence, for each $\mathcal{I}$ such that $A^{\mathcal{I}} \neq \emptyset$, we have $\mathcal{I} \not\models \mathcal{O}$.

We are finally ready to formally define the notion of logical entailment. We say that $\mathcal{O}$ *entails* an axiom α (written $\mathcal{O} \models \alpha$), if every model of $\mathcal{O}$ satisfies α. Intuitively this means that the axiom α should hold in every situation that agrees with the restrictions imposed by $\mathcal{O}$. Note that according to this definition, if $\mathcal{O}$ is unsatisfiable then the entailment $\mathcal{O} \models \alpha$ holds for every axiom α.

Example 8. Consider $\mathcal{O} = \{C \sqsubseteq \exists r.D,\ D \sqsubseteq E\}$. We prove that $\mathcal{O} \models C \sqsubseteq \exists r.E$. Take any $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{O}$. We show that $\mathcal{I} \models C \sqsubseteq \exists r.E$ or, equivalently, $C^{\mathcal{I}} \subseteq (\exists r.E)^{\mathcal{I}}$. Suppose that $x \in C^{\mathcal{I}}$. Since $\mathcal{I} \models C \sqsubseteq \exists r.D$, we have $C^{\mathcal{I}} \subseteq (\exists r.D)^{\mathcal{I}}$, so $x \in (\exists r.D)^{\mathcal{I}}$. Then there exists some $y \in \Delta^{\mathcal{I}}$ such that $\langle x, y\rangle \in r^{\mathcal{I}}$ and $y \in D^{\mathcal{I}}$. Since $\mathcal{I} \models D \sqsubseteq E$, we have $y \in D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$. Therefore, since $\langle x, y\rangle \in r^{\mathcal{I}}$ and $y \in E^{\mathcal{I}}$ we obtain $x \in (\exists r.E)^{\mathcal{I}}$. Thus, for each $x \in C^{\mathcal{I}}$, we have $x \in (\exists r.E)^{\mathcal{I}}$, which means that $C^{\mathcal{I}} \subseteq (\exists r.E)^{\mathcal{I}}$ or, equivalently, $\mathcal{I} \models C \sqsubseteq \exists r.E$.

2.3 Reasoning Problems

There are many situations in which one is interested in performing logical operations with ontologies, such as checking consistency or verifying entailments. Just like computer software, ontologies are usually developed manually by humans, and humans tend to make mistakes. For programming languages, dedicated software tools, such as syntax checkers, compilers, debuggers, testing frameworks, and static analysis tools help preventing and finding errors. Similar tools also exist for ontology development.

Just like for programming languages, one usually distinguishes several types of errors in ontologies. *Syntax errors* usually happen when the syntax rules for constructing concepts and axioms described in Sect. 2.1 are not used correctly. For example $C\neg D$ is not a correct concept according to grammar (1) since the negation operation is unary. Syntax errors also include situations where an

atomic role is used in the position of an atomic concept or when the parentheses are not balanced. Syntax errors are relatively easy to find using *parsers*, which can verify that the ontology is well-formed.

It can be that the ontology is syntactically well-formed, but some of its axioms do not make sense. For example, an ontology may contain the axiom $\mathsf{Father} \equiv \mathsf{Male} \sqcup \mathsf{Parent}$, in which, clearly a disjunction was accidentally used instead of a conjunction. Although for a human the problem seems obvious, it is hard to detect such an error using automated tools since computers do not know the meaning of the words involved. From the computer point of view, this axiom looks like $C \equiv D \sqcup E$, which is a legitimate axiom. These kinds of errors are usually called *semantic* or *modeling errors*.

Although it is not possible to automatically detect modeling errors in general, there are some common *symptoms* for such errors. For example, an incorrectly formulated axiom may cause a logical contradiction with other axioms in the ontology, which makes the whole ontology unsatisfiable. Another common symptom is unsatisfiability of *atomic* concepts with respect to an ontology. Each atomic concept is usually introduced to capture a certain non-empty subset of objects in the modeled domain. For example, the concept Parent was introduced to capture the individuals who are parents in the real world. If an atomic concept is unsatisfiable, this indicates that the modeled domain cannot correspond to any model of the ontology. Note that, as shown in Example 7, an atomic concept can be unsatisfiable even with respect to a satisfiable ontology.

A modeling error may also result in incorrect entailments of the ontology, which are sometimes easier to detect than the error itself. For example, the erroneous axiom $\mathsf{Father} \equiv \mathsf{Male} \sqcup \mathsf{Parent}$ entails the simpler concept inclusions $\mathsf{Male} \sqsubseteq \mathsf{Father}$ and $\mathsf{Parent} \sqsubseteq \mathsf{Father}$, which are also incorrect from the modeling point of view. By observing the entailed concept inclusions $A \sqsubseteq B$ between atomic concepts appearing in the ontology, an ontology developer can usually quickly identify those incorrect entailments. When the entailment $\mathcal{O} \models C \sqsubseteq D$ holds, it is often said that the concept C is *subsumed by* the concept D (or the concept D *subsumes* the concept C) w.r.t. $\mathcal{O}$. For detecting problems involving individuals, one can similarly inspect the entailed concept assertions $A(a)$ between atomic concepts A and individuals a appearing in the ontology. When $\mathcal{O} \models C(a)$, it is often said that a is an *instance* of C (or C is a *type* of a) w.r.t. $\mathcal{O}$. Checking subsumptions and instances is not only useful for finding modeling errors, but also for answering *queries*, which is usually the main purpose of ontologies in applications. For example, given a (complex) concept C, it is possible to query for all atomic concepts A for which the subsumption $\mathcal{O} \models C \sqsubseteq A$ holds or to query for all individuals a which are instances of C.

Thus, one can distinguish several *standard reasoning problems* that are of interest in ontology-based applications:

1. *Ontology satisfiability checking*:
   - Given: an ontology $\mathcal{O}$,
   - Return: *yes* if $\mathcal{O}$ is satisfiable and *no* otherwise.
2. *Concept satisfiability checking*:

- Given: an ontology $\mathcal{O}$ and a concept C,
- Return: *yes* if C is satisfiable w.r.t. $\mathcal{O}$ and *no* otherwise.

3. *Concept subsumption checking*:
   - Given: an ontology $\mathcal{O}$ and a concept inclusion $C \sqsubseteq D$,
   - Return: *yes* if $\mathcal{O} \models C \sqsubseteq D$ and *no* otherwise.
4. *Instance checking*:
   - Given: an ontology $\mathcal{O}$ and a concept assertion $C(a)$,
   - Return: *yes* if $\mathcal{O} \models C(a)$ and *no* otherwise.

Example 9. Consider the ontology $\mathcal{O}$ from Example 1:

1. $\mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top,$
2. $\mathsf{GrandParent} \equiv \exists \mathsf{hasChild}.\mathsf{Parent},$
3. $\mathsf{hasChild}(\mathsf{john}, \mathsf{mary}).$

As was shown in Example 6, this ontology has a model $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ with

- $\Delta^{\mathcal{J}} = \{a\}$,
- $\mathsf{Parent}^{\mathcal{J}} = \mathsf{GrandParent}^{\mathcal{J}} = \{a\}, \quad \mathsf{hasChild}^{\mathcal{I}} = \{\langle a, a\rangle\}$,
- $\mathsf{john}^{\mathcal{J}} = \mathsf{mary}^{\mathcal{J}} = a$.

Therefore, the answer to the ontology satisfiability checking problem for $\mathcal{O}$ is *yes*.

The answer to the concept satisfiability checking problem for $\mathcal{O}$ and concept Parent is also *yes* because $\mathcal{J} \models \mathcal{O}$ and $\mathsf{Parent}^{\mathcal{J}} = \{a\} \neq \emptyset$. The same answer is also obtained for the inputs $\mathcal{O}$ and $\mathsf{GrandParent}$.

We next check which subsumptions hold between these concepts. The subsumption $\mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$ holds in $\mathcal{J}$ since $\mathsf{Parent}^{\mathcal{J}} = \{a\} \subseteq \{a\} = \mathsf{GrandParent}^{\mathcal{J}}$, but there is another model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{O}$ in which this subsumption does not hold:

- $\Delta^{\mathcal{I}} = \{a, b\}$,
- $\mathsf{Parent}^{\mathcal{I}} = \{a\}, \quad \mathsf{GrandParent}^{\mathcal{I}} = \emptyset, \quad \mathsf{hasChild}^{\mathcal{I}} = \{\langle a, b\rangle\}$,
- $\mathsf{john}^{\mathcal{I}} = a, \quad \mathsf{mary}^{\mathcal{I}} = b$.

Indeed, $\mathsf{Parent}^{\mathcal{I}} = \{a\} = (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$. Therefore, $\mathcal{I} \models \mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top$. We further have $\mathcal{I} \models \mathsf{GrandParent} \equiv \exists \mathsf{hasChild}.\mathsf{Parent}$ since $\mathsf{GrandParent}^{\mathcal{I}} = \emptyset = (\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}}$. Since $\langle \mathsf{john}^{\mathcal{I}}, \mathsf{mary}^{\mathcal{I}}\rangle = \langle a, b\rangle \subseteq \{\langle a, b\rangle\} = \mathsf{hasChild}^{\mathcal{I}}$, we have $\mathcal{I} \models \mathsf{hasChild}(\mathsf{john}, \mathsf{mary})$. Therefore, $\mathcal{I} \models \mathcal{O}$. However, $\mathsf{Parent}^{\mathcal{I}} = \{a\} \not\subseteq \emptyset = \mathsf{GrandParent}^{\mathcal{I}}$. Therefore, $\mathcal{I} \not\models \mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$. Since we have found a model $\mathcal{I}$ of $\mathcal{O}$ for which the subsumption $\mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$ does not hold, we have proved that $\mathcal{O} \not\models \mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$. Therefore, the answer to the concept subsumption checking problem for $\mathcal{O}$ and $\mathsf{Parent} \sqsubseteq \mathsf{GrandParent}$ is *no*.

The subsumption $\mathsf{GrandParent} \sqsubseteq \mathsf{Parent}$ holds in both $\mathcal{J}$ and $\mathcal{I}$, and in fact, in all models of $\mathcal{O}$. Indeed, assume that $\mathcal{I} \models \mathcal{O}$. We will show that $\mathsf{GrandParent}^{\mathcal{I}} \subseteq \mathsf{Parent}^{\mathcal{I}}$. To do this, take any $x \in \mathsf{GrandParent}^{\mathcal{I}}$. If there is no such x then, trivially, $\mathsf{GrandParent}^{\mathcal{I}} = \emptyset \subseteq \mathsf{Parent}^{\mathcal{I}}$. Since $\mathcal{I} \models \mathsf{GrandParent} \equiv$

$\exists \mathsf{hasChild}.\mathsf{Parent}$, we have $x \in \mathsf{GrandParent}^{\mathcal{I}} = (\exists \mathsf{hasChild}.\mathsf{Parent})^{\mathcal{I}}$. Hence, there exists some y such that $\langle x, y\rangle \in \mathsf{hasChild}^{\mathcal{I}}$ and $y \in \mathsf{Parent}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} = \top^{\mathcal{I}}$. Hence $x \in (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$. Since $\mathcal{I} \models \mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top$, we have $\mathsf{Parent}^{\mathcal{I}} = (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$. Hence, $x \in \mathsf{Parent}^{\mathcal{I}}$. Since $x \in \mathsf{GrandParent}^{\mathcal{I}}$ was arbitrary, we proved that $\mathsf{GrandParent}^{\mathcal{I}} \subseteq \mathsf{Parent}^{\mathcal{I}}$, that is, $\mathcal{I} \models \mathsf{GrandParent} \sqsubseteq \mathsf{Parent}$ and so, $\mathcal{O} \models \mathsf{GrandParent} \sqsubseteq \mathsf{Parent}$.

Finally, we check which of the individuals john and mary appearing in $\mathcal{O}$ are instances of the atomic concepts Parent and $\mathsf{GrandParent}$. For the model $\mathcal{I}$ defined above, $\mathsf{GrandParent}^{\mathcal{I}} = \emptyset$, hence, $\mathcal{I} \not\models \mathsf{GandParent}(\mathsf{john})$ and $\mathcal{I} \not\models \mathsf{GandParent}(\mathsf{mary})$ since $\mathsf{john}^{\mathcal{I}} = a \notin \emptyset = \mathsf{GandParent}^{\mathcal{I}}$ and $\mathsf{mary}^{\mathcal{I}} = b \notin \emptyset = \mathsf{GandParent}^{\mathcal{I}}$. Also $\mathcal{I} \not\models \mathsf{Parent}(\mathsf{mary})$ since $\mathsf{mary} = b \notin \{a\} = \mathsf{Parent}^{\mathcal{I}}$. Hence, the answer to the instance checking problem for $\mathcal{O}$ and each concept assertion $\mathsf{GandParent}(\mathsf{john})$, $\mathsf{GandParent}(\mathsf{mary})$, and $\mathsf{Parent}(\mathsf{mary})$ is *no*.

The answer to the instance checking problem for $\mathcal{O}$ and the fourth concept assertion $\mathsf{Parent}(\mathsf{john})$ is *yes* since $\mathcal{I} \models \mathsf{Parent}(\mathsf{john})$ for each $\mathcal{I} \models \mathcal{O}$. Indeed, let $x = \mathsf{john}^{\mathcal{I}}$ and $y = \mathsf{mary}^{\mathcal{I}}$. Since $\mathcal{I} \models \mathsf{hasChild}(\mathsf{john}, \mathsf{mary})$, we have $\langle x, y\rangle \in \mathsf{hasChild}^{\mathcal{I}}$. Trivially, $y \in \Delta^{\mathcal{I}} = \top^{\mathcal{I}}$. Hence $x \in (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$. Since $\mathcal{I} \models \mathsf{Parent} \equiv \exists \mathsf{hasChild}.\top$, we have $\mathsf{Parent}^{\mathcal{I}} = (\exists \mathsf{hasChild}.\top)^{\mathcal{I}}$. Therefore, $x \in \mathsf{Parent}^{\mathcal{I}}$. Since $x = \mathsf{john}^{\mathcal{I}}$, we proved that $\mathcal{I} \models \mathsf{Parent}(\mathsf{john})$ and, since $\mathcal{I}$ was an arbitrary model of $\mathcal{O}$, we proved that $\mathcal{O} \models \mathsf{Parent}(\mathsf{john})$.

Exercise 1. Determine which of the individuals john and mary are instances of the *negated* concepts $\neg\mathsf{Parent}$ and $\neg\mathsf{GrandParent}$ for the ontology $\mathcal{O}$ in Example 9. Are there any surprises? Can you explain the unexpected answers you obtained?

In Example 9 we have solved all reasoning problems "by hand" by either providing counter-models for entailments or proving that entailments hold for all models. Of course, in practice, it is not expected that the ontology developers or anybody else is going to solve these tasks manually. It is expected that these tasks are solved by computers *automatically* and, preferably, *quickly*. The main focus of the research in DLs, therefore, was development and analysis of algorithms for solving reasoning problems.

We have listed four standard reasoning tasks for ontologies: (1) ontology satisfiability checking, (2) concept satisfiability checking, (3) concept subsumption checking, and (4) instance checking. Developing *separate* algorithms for solving each of these problems would be too time consuming. Fortunately, it turns out, as soon as we find an algorithm for solving one of these problems, we can solve the remaining three too using simple modifications of this algorithm.

2.4 Reductions Between Reasoning Problems

In the remainder of this course, we are interested in measuring the *computational complexity* of problems and algorithms. We also develop *polynomial reductions* between the four standard reasoning problems mentioned above. Appendix A.1 provides additional material for readers who first want to refresh their knowledge about these notions.

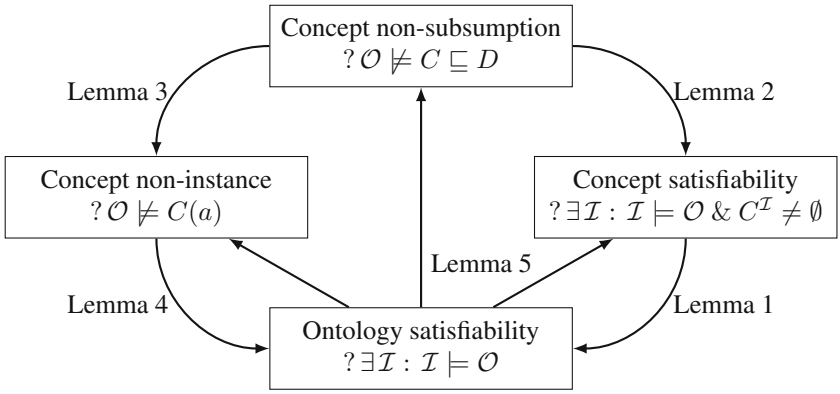


Fig. 1. An overview of reductions between the standard DL reasoning problems

Notice the similarities in the formulations of the reasoning problems 1–4 given earlier. All problems are decision problems, i.e., they get some objects as inputs and are expected to produce either *yes* or *no* as the output. We next apply the common approach of formulating polynomial reductions between these reasoning problems: We first prove that all problems can be reduced to the ontology satisfiability problem and then show how to reduce the ontology satisfiability problem to all other problems. The overview of the reductions is shown in Fig. 1. Note that for the concept subsumption and instance checking problems, we provide reductions for their *complementary* problems, i.e., where the answers *yes* and *no* are swapped.

The following lemma shows how to reduce the concept satisfiability problem to the ontology satisfiability problem. Intuitively, to check if a concept C is satisfiable w.r.t. $\mathcal{O}$, it is sufficient to extend $\mathcal{O}$ with a new concept assertion $C(a)$ and check satisfiability of the resulting ontology. Clearly, the reduction $R(\langle \mathcal{O}, C \rangle) = \{\mathcal{O} \cup C(a)\}$ can be computed in polynomial time in the size of $\mathcal{O}$ plus C.

Lemma 1. *Let $\mathcal{O}$ be an ontology, C a concept, and a an individual not appearing in $\mathcal{O}$. Then C is satisfiable w.r.t. $\mathcal{O}$ if and only if $\mathcal{O} \cup \{C(a)\}$ is satisfiable.*

Proof. ($\Rightarrow$): To prove the "only if" direction, assume that C is satisfiable w.r.t. $\mathcal{O}$. Then there exists a model $\mathcal{I} \models \mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$. That is, there exists some $x \in C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. Let $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be a new interpretation defined as follows:

- $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$,
- $A^{\mathcal{J}} = A^{\mathcal{I}}$ and $r^{\mathcal{J}} = r^{\mathcal{I}}$ for each atomic concept A and atomic role r,
- $b^{\mathcal{J}} = b^{\mathcal{I}}$ for every individual $b \neq a$,
- $a^{\mathcal{J}} = x \in C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$.

Clearly, $\mathcal{J} \models \mathcal{O}$ since the interpretation of every symbol in $\mathcal{O}$ remained unchanged, and $\mathcal{J} \models C(a)$ since $a^{\mathcal{J}} = x \in C^{\mathcal{I}} = C^{\mathcal{J}}$. Hence, $\mathcal{O} \cup \{C(a)\}$ is satisfiable.

($\Leftarrow$): To prove the "if" directly, assume that $\mathcal{O} \cup \{C(a)\}$ is satisfiable. Then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I} \models C(a)$. The last implies $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Hence $C^{\mathcal{I}} \neq \emptyset$. Consequently, C is satisfiable w.r.t. $\mathcal{O}$. □

Note that in the proof of the "only if" direction of Lemma 1, it is essential that the individual a is *fresh*, i.e., it does not appear in $\mathcal{O}$. If this assumption is dropped, the lemma does not hold any longer.

Exercise 2. Give an example of an $\mathcal{ALC}$ ontology $\mathcal{O}$ and a concept assertion $C(a)$, with individual a appearing in $\mathcal{O}$ such that C is satisfiable w.r.t. $\mathcal{O}$ but $\mathcal{O} \cup \{C(a)\}$ is not satisfiable.

Exercise 3. The reduction described in Lemma 1 introduces a new individual to the ontology, even if the original ontology did not have any individuals. This may be undesirable if the algorithm for checking ontology satisfiability can work only with TBoxes. Formulate a different reduction from concept satisfiability to ontology satisfiability that does not introduce any individuals. Prove that this reduction is correct like in Lemma 1.

Hint 1: Using a concept assertion $C(a)$ one forces the interpretation of C to be nonempty. Using which other axioms one can force non-emptiness of concepts? Which concepts are always interpreted by nonempty sets? Hint 2: Similar to fresh individuals, the reduction can use fresh atomic concepts and roles.

We next show how to reduce the problem of checking concept non-subsumption to the problem of concept satisfiability, which, in turn, as shown by Lemma 1, can be reduced to checking ontology satisfiability.

Lemma 2. *Let $\mathcal{O}$ be an ontology and C, D concepts. Then $\mathcal{O} \not\models C \sqsubseteq D$ if and only if $C \sqcap \neg D$ is satisfiable w.r.t. $\mathcal{O}$.*

Proof. It is easy to see that the following statements are equivalent:

1. $\mathcal{O} \not\models C \sqsubseteq D$,
2. There exists a model $\mathcal{I} \models \mathcal{O}$ such that $C^{\mathcal{I}} \not\subseteq D^{\mathcal{I}}$,
3. There exists a model $\mathcal{I} \models \mathcal{O}$ such $C^{\mathcal{I}} \setminus D^{\mathcal{I}} \neq \emptyset$,
4. There exists a model $\mathcal{I} \models \mathcal{O}$ such $(C \sqcap \neg D)^{\mathcal{I}} \neq \emptyset$,
5. $C \sqcap \neg D$ is satisfiable w.r.t. $\mathcal{O}$.

In particular, 3 and 4 are equivalent because:
$(C \sqcap \neg D)^{\mathcal{I}} = C^{\mathcal{I}} \cap (\neg D)^{\mathcal{I}} = C^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \setminus D^{\mathcal{I}}) = C^{\mathcal{I}} \setminus D^{\mathcal{I}}$. □

The problem of checking concept (non-)subsumption can alternatively be reduced to checking (non-)entailment of concept instances. The following lemma proves that to check $\mathcal{O} \models C \sqsubseteq D$, one can extend $\mathcal{O}$ with a concept assertion $C(a)$ for a fresh individual a, and check if the resulting ontology entails the concept instance $D(a)$.

Lemma 3. *Let $\mathcal{O}$ be an ontology, C, D concepts, and a an individual not appearing in $\mathcal{O}$. Then $\mathcal{O} \models C \sqsubseteq D$ if and only if $\mathcal{O} \cup \{C(a)\} \models D(a)$.*

Proof. ($\Rightarrow$): To prove the "only if" direction, assume that $\mathcal{O} \models C \sqsubseteq D$. To show that $\mathcal{O} \cup \{C(a)\} \models D(a)$, take any model $\mathcal{I}$ of $\mathcal{O} \cup \{C(a)\}$. Since $\mathcal{I} \models \mathcal{O}$ and $\mathcal{O} \models C \sqsubseteq D$, we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Then, since $\mathcal{I} \models C(a)$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Hence, $\mathcal{I} \models D(a)$. Since $\mathcal{I}$ was an arbitrary model such that $\mathcal{I} \models \mathcal{O} \cup \{C(a)\}$, we have shown that $\mathcal{O} \cup \{C(a)\} \models D(a)$.

($\Leftarrow$): We prove the "if" direction by showing the contrapositive: if $\mathcal{O} \not\models C \sqsubseteq D$ then $\mathcal{O} \cup \{C(a)\} \not\models D(a)$. Assume that $\mathcal{O} \not\models C \sqsubseteq D$. Then there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $\mathcal{I} \not\models C \sqsubseteq D$, or, equivalently, $C^{\mathcal{I}} \not\subseteq D^{\mathcal{I}}$. This means that there exists $x \in C^{\mathcal{I}}$ such that $x \notin D^{\mathcal{I}}$. Define another interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$, which is identical to $\mathcal{I}$ on all symbols, except for the interpretation of the individual a:

- $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}}$,
- $A^{\mathcal{J}} = A^{\mathcal{I}}$ and $r^{\mathcal{J}} = r^{\mathcal{I}}$ for each atomic concept A and atomic role r,
- $b^{\mathcal{J}} = b^{\mathcal{I}}$ for every individual $b \neq a$,
- $a^{\mathcal{J}} = x \in C^{\mathcal{I}} \setminus D^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$.

Clearly, $\mathcal{J} \models \mathcal{O}$ since the interpretation of every symbol in $\mathcal{O}$ remained unchanged. Since $a^{\mathcal{J}} \in C^{\mathcal{I}} \setminus D^{\mathcal{I}} = C^{\mathcal{J}} \setminus D^{\mathcal{J}}$, we have $\mathcal{J} \models C(a)$ and $\mathcal{J} \not\models D(a)$. Thus, we found $\mathcal{J} \models \mathcal{O} \cup \{C(a)\}$ such that $\mathcal{J} \not\models D(a)$, which proves that $\mathcal{O} \cup \{C(a)\} \not\models D(a)$, as required. □

Exercise 4. Similarly to Exercise 2, show that the condition that the individual a used in Lemma 3 is fresh cannot be dropped. Give an example where the statement of the lemma is not true without this condition.

As the next lemma shows, the concept instance checking problem can easily be reduced to checking ontology satisfiability.

Lemma 4. *Let $\mathcal{O}$ be an ontology, C a concept, and a an individual. Then $\mathcal{O} \not\models C(a)$ if and only if $\mathcal{O} \cup \{(\neg C)(a)\}$ is satisfiable.*

Proof. It is easy to see that the following statements are equivalent:

1. $\mathcal{O} \not\models C(a)$,
2. There exists a model $\mathcal{I} \models \mathcal{O}$ such that $a^{\mathcal{I}} \notin C^{\mathcal{I}}$,
3. There exists a model $\mathcal{I} \models \mathcal{O}$ such that $a^{\mathcal{I}} \in (\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
4. $\mathcal{O} \cup \{(\neg C)(a)\}$ is satisfiable. □

Finally, we show that the ontology satisfiability problem can easily be reduced to the other three problems. Specifically, to check if an ontology $\mathcal{O}$ is satisfiable, one can check if the concept $\top$ is satisfiable with respect to $\mathcal{O}$ (thus, reducing to the concept satisfiability problem), or check if $\mathcal{O}$ does not entail the subsumption $\top \sqsubseteq \bot$ (thus reducing to the concept non-subsumption problem), or check if $\mathcal{O}$ does not entail an instance $\top(a)$ for some individual a (thus, reducing to the concept non-instance problem).

Lemma 5. *Let $\mathcal{O}$ be an ontology. Then the following conditions are equivalent:*

1. $\mathcal{O}$ *is satisfiable,*
2. $\top$ *is satisfiable with respect to* $\mathcal{O}$*,*
3. $\mathcal{O} \not\models \top \sqsubseteq \bot$*,*
4. $\mathcal{O} \not\models \bot(a)$ *for every individual* a*,*
5. $\mathcal{O} \not\models \bot(a)$ *for some individual* a*.*

Proof. *Case* $1 \Rightarrow 2$*:* If $\mathcal{O}$ is satisfiable then $\mathcal{I} \models \mathcal{O}$ for some $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, then $\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \neq \emptyset$ for some $\mathcal{I} \models \mathcal{O}$, then $\top$ is satisfiable with respect to $\mathcal{O}$.

Case $2 \Rightarrow 3$*:* If $\top$ is satisfiable with respect to $\mathcal{O}$ then there exists $\mathcal{I} \models \mathcal{O}$ such that $\top^{\mathcal{I}} \neq \emptyset$, then $\top^{\mathcal{I}} \not\subseteq \emptyset = \bot^{\mathcal{I}}$, then $\mathcal{I} \not\models \top \sqsubseteq \bot$, then $\mathcal{O} \not\models \top \sqsubseteq \bot$ since $\mathcal{I} \models \mathcal{O}$.

Case $3 \Rightarrow 4$*:* If $\mathcal{O} \not\models \top \sqsubseteq \bot$, then there exists $\mathcal{I} \models \mathcal{O}$ (such that $\top^{\mathcal{I}} \not\subseteq \bot^{\mathcal{I}}$) then, for every individual a: $a^{\mathcal{I}} \notin \emptyset = \bot^{\mathcal{I}}$, hence $\mathcal{I} \not\models \bot(a)$, hence $\mathcal{O} \not\models \bot(a)$ since $\mathcal{I} \models \mathcal{O}$.

Case $4 \Rightarrow 5$*:* If $\mathcal{O} \not\models \bot(a)$ for every individual a then, trivially, $\mathcal{O} \not\models \bot(a)$ for some individual a since the set of individuals N_I is nonempty.

Case $5 \Rightarrow 1$*:* If $\mathcal{O} \not\models \bot(a)$ for some individual a, then there exists $\mathcal{I} \models \mathcal{O}$ (such that $a^{\mathcal{I}} \notin \bot^{\mathcal{I}}$), then $\mathcal{O}$ is satisfiable. □

3 Tableau Procedures

In this section, we introduce the so-called *tableau procedures*, which are the most popular procedures for reasoning in DLs, particularly, for very expressive languages. Tableau procedures or variants thereof have been implemented in many *ontology reasoners*, such as HermiT [42], FacT++ [62], Konclude [58], and Pellet [56]. Intuitively, tableau procedures work by trying to construct ontology models of a particular shape, called the *tree* models. To simplify our exposition, in this section we mainly focus on tableau procedures for TBox reasoning, i.e., we assume that our ontologies do not contain concept and role assertions.

The construction of a model is governed by a number of rules that incrementally expand the model by adding new domain elements and requirements that they need to satisfy (e.g., be instances of particular concepts). To describe this process in a convenient way, in tableau procedures one works with a different representation of interpretations, which is called a tableau.

Definition 1. *A* tableau *is a tuple* $T = (V, L)$*, where*

- V *is a nonempty set of* tableau nodes *of* T*,*
- L *is a* labeling function *that assigns:*
  - *to every node* $v \in V$ *a subset* $L(v)$ *of concepts,*
  - *to every pair of nodes* $\langle v, w \rangle \in V \times V$ *a subset* $L(v, w)$ *of roles.*

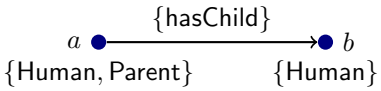
A tableau T is usually drawn as a *labeled graph* with the set of vertices V and the set of (directed) edges $E = \{\langle v, w \rangle \in V \times V \mid L(v, w) \neq \emptyset\}$, in which every node $v \in V$ is labeled with $L(v)$ and every edge $\langle v, w \rangle \in E$ is labeled with $L(v, w)$. In what follows we assume that if $L(v)$ or $L(v, w)$ were not explicitly assigned for some nodes $\{v, w\} \subseteq V$, then $L(v) = L(v, w) = \emptyset$.

Example 10. Consider the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ from Example 2 extended with $\mathsf{Human}^{\mathcal{I}} = \{a, b\}$:

- $\Delta^{\mathcal{I}} = \{a, b\}$,
- $\mathsf{Human}^{\mathcal{I}} = \{a, b\}$, $\mathsf{Parent}^{\mathcal{I}} = \{a\}$, $\mathsf{GrandParent}^{\mathcal{I}} = \emptyset$,
- $\mathsf{hasChild}^{\mathcal{I}} = \{\langle a, b\rangle\}$,
- $\mathsf{john}^{\mathcal{I}} = a$, $\mathsf{mary}^{\mathcal{I}} = b$.

This interpretation can be equivalently represented by a tableau $T = (V, L)$ with:

- $V = \{a, b\}$,
- $L(a) = \{\mathsf{Human}, \mathsf{Parent}\}$,
- $L(b) = \{\mathsf{Human}\}$,
- $L(a, b) = \{\mathsf{hasChild}\}$.



This tableau is graphically illustrated on the right.

Note that according to Definition 1, tableau nodes can be labeled with arbitrary concepts, not necessarily with atomic ones like in Example 10. This is in contrast to interpretations, which define only the values for atomic concepts and roles. For interpretations, it is not necessary to define how complex concepts are interpreted, since these values can always be *calculated* according to the rules given in Sect. 2.2. For the tableau procedures, the information $C \in L(v)$ represents a *requirement* that $v \in C^{\mathcal{I}}$ should hold for the constructed model $\mathcal{I}$. The tableau should subsequently be expanded to satisfy all such requirements. For example, if $C \sqcap D \in L(v)$ then $L(v)$ should be expanded by adding the concepts C and D, since $v \in (C \sqcap D)^{\mathcal{I}}$ implies $v \in C^{\mathcal{I}}$ and $v \in D^{\mathcal{I}}$. This expansion process is governed using dedicated *tableau expansion rules*.

As we have seen in Sect. 2.3, all standard reasoning problems can be reduced to each other in polynomial time. Therefore, an algorithm for solving any of these problems can easily be modified to solve the other problems. In the next sections, therefore, we use tableau procedures for solving one of these problems: concept satisfiability.

3.1 Deciding Concept Satisfiability

In this section, we formulate a simplified version of the procedure for checking concept satisfiability that works without considering the axioms in the ontology. That is, given an $\mathcal{ALC}$ concept C we need to check if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$. Although this problem is not of much use in ontology-based applications, it allows us to illustrate the main principles of tableau procedures.

To check satisfiability of a given concept C, we first transform it into a special *normal form*, which is easier to work with.

Definition 2. *An $\mathcal{ALC}$ concept C is in* negation normal form *(short:* NNF*) if negation can appear in C only in the form of $\neg A$, where A is an atomic concept.*

Table 2. Tableau expansion rules for checking satisfiability of $\mathcal{ALC}$ concepts

Rule	Conditions	Expansions
$\sqcap$-Rule	$D \sqcap E \in L(x)$, $\{D, E\} \not\subseteq L(x)$	Set $L(x) := L(x) \cup \{D, E\}$
$\sqcup$-Rule	$D \sqcup E \in L(x)$, $\{D, E\} \cap L(x) = \emptyset$	Set $L(x) := L(x) \cup \{D\}$ or $L(x) := L(x) \cup \{E\}$
$\exists$-Rule	$\exists r.D \in L(x)$ and there is no $y \in V$ such that $r \in L(x, y)$ and $D \in L(y)$	Extend $V := V \cup \{y\}$ for a new y, set $L(x, y) := \{r\}$ and $L(y) := \{D\}$
$\forall$-Rule	$\forall r.D \in L(x)$, $r \in L(x, y)$, $D \notin L(y)$	Set $L(y) := L(y) \cup \{D\}$
$\bot$-Rule	$\{A, \neg A\} \subseteq L(x)$, $\bot \notin L(x)$	Set $L(x) := L(x) \cup \{\bot\}$

In other words, to construct a concept in NNF, it is permitted to apply negation only to atomic concepts. Thus, $\mathcal{ALC}$ concepts in NNF can be defined by the grammar:

$$C, D ::= A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg A \mid \exists r.C \mid \forall r.C. \tag{6}$$

Example 11. The concept $\forall r.(\neg A \sqcup \exists S.\neg B)$ is in NNF; the concepts $\neg\exists r.A$, $\forall r.\neg(A \sqcap B)$, and $A \sqcap \exists r.\neg\top$ on the other hand are *not* in NNF.

Each $\mathcal{ALC}$ concept C can be converted to an equivalent concept in NNF by applying simple rules to "push negation inwards" that are reminiscent of De Morgan's Laws:

$$\begin{aligned}
\neg(C \sqcap D) &\Rightarrow (\neg C) \sqcup (\neg D), & \neg\neg C &\Rightarrow C,\\
\neg(C \sqcup D) &\Rightarrow (\neg C) \sqcap (\neg D), & \neg\top &\Rightarrow \bot,\\
\neg(\exists r.C) &\Rightarrow \forall r.(\neg C), & \neg\bot &\Rightarrow \top.\\
\neg(\forall r.C) &\Rightarrow \exists r.(\neg C),
\end{aligned}$$

Example 12. Consider the $\mathcal{ALC}$ concept $(\exists r.A) \sqcap \neg((\exists r.A) \sqcap \neg B)$. This concept can be converted to NNF as follows:

$$\begin{aligned}
(\exists r.A) \sqcap \neg((\exists r.A) \sqcap \neg B) &\Rightarrow (\exists r.A) \sqcap (\neg(\exists r.A) \sqcup \neg\neg B)\\
&\Rightarrow (\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B).
\end{aligned}$$

Exercise 5. Show that the transformation of concepts to NNF described above preserves satisfiability of concepts. That is, the input concept is satisfiable if and only if its NNF is satisfiable. Hint: show for each transformation step $C \Rightarrow D$ that $C^{\mathcal{I}} = D^{\mathcal{I}}$ holds for every interpretation $\mathcal{I}$.

To check satisfiability of an $\mathcal{ALC}$ concept C in NNF, we create a new Tableau $T = (V, L)$ with $V = \{v_0\}$ and $L(v_0) = \{C\}$, and apply the *tableau expansion* rules from Table 2. A rule is *applicable* if all *conditions* of the rule are satisfied in the current tableau T for certain choices of *rule parameters*, such as the values of x, y or the matching concepts and roles in the labels. For example, the $\sqcap$-Rule

Fig. 2. Two possible tableau expansions for the concept $C = (\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B)$ due to the non-deterministic $\sqcup$-Rule

is applicable to a node $x \in V$ if some conjunction $D \sqcap E$ belongs to the label $L(x)$ of this node, but at least one of the conjuncts D or E does not belong to $L(x)$. In this case, T is *expanded* by adding new nodes or labels as specified in the *expansions* part of the rules. In this case we say that the rule is *applied* (for the specific choice of the rule parameters). For example, the $\sqcap$-Rule is applied by adding the conjuncts D and E to the label $L(x)$ of the node x. Note that after applying each rule in Table 2, the rule is no longer applicable for the same choices of rule parameters. The tableau expansion rules are applied until no rule is applicable any longer. In this case we say that the T is *fully expanded*.

Example 13. Consider the concept $C = (\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B)$ obtained by the conversion to NNF in Example 12. We check satisfiability of C by applying the tableau expansion rules from Table 2. Consider first the left-hand side of Fig. 2. We initialize $T = (V, L)$ by setting $V = \{v_0\}$ and $L(v_0) = \{C\}$. Since C is a conjunction, the conditions of the $\sqcap$-Rule are satisfied for $x = v_0$, $D = \exists r.A$, and $E = \forall r.(\neg A) \sqcup B$. Applying this rule adds the conjuncts $\exists r.A$ and $\forall r.(\neg A) \sqcup B$ to $L(v_0)$. Now the conditions of the $\exists$-Rule are satisfied for $x = v_0$ and $\exists r.A \in L(v_0)$: note that there is no $v \in V$ such that $r \in L(v_0, v)$. Applying this rule creates a new node v_1, and sets $L(v_0, v_1) = \{r\}$ and $L(v_1) = \{A\}$. Similarly, since $\forall r.(\neg A) \sqcup B \in L(v_0)$, but neither $\forall r.(\neg A) \in L(v_0)$ nor $B \in L(v_0)$, the $\sqcup$-Rule is applicable. There are two ways this rule can be applied to T: either we add the first disjunct $\forall r.(\neg A)$ to $L(v_0)$, or we add the second disjunct B to $L(v_0)$. It is not necessary to add both of them. Let us chose the first disjunct and see what happens. After we apply the $\sqcup$-Rule in this way, we obtain $\forall r.(\neg A) \in L(v_0)$. Since $r \in L(v_0, v_1)$ and $\neg A \notin L(v_1)$, the $\forall$-Rule is now applicable for $x = v_0$, $y = v_1$, and $\forall r.(\neg A) \in L(v_0)$. The application of this rule adds $\neg A$ to $L(v_1)$. Now we have both A and $\neg A$ in $L(v_1)$, which satisfies the conditions of the $\bot$-Rule since $\bot \notin L(v_1)$. The application of the $\bot$-Rule, therefore, adds $\bot$ to $L(v_1)$. After applying this rule, no further rule is applicable, so the tableau is fully expanded.

If during the application of the $\sqcup$-Rule to $\forall r.(\neg A) \sqcup B \in L(v_0)$ we, alternatively, choose to add the second disjunct B to $L(v_0)$, we obtain another fully expanded tableau without $\bot \in L(v_1)$ shown in the right-hand side of Fig. 2.

Remark 1. Note that the tableau edges in Example 13 were labeled by just a single role r. Although Definition 1 allows for arbitrary *sets* of roles in edge labels, the tableau rules for $\mathcal{ALC}$, can only create *singleton* sets of roles. Indeed, it is easy

to see from Table 2, that the $\exists$-Rule is the only rule that can modify edge labels, and can only set them to singleton role sets $\{r\}$. More expressive languages, such as the DL $\mathcal{ALCH}$ to be considered in Exercise 10, can have additional rules that can *extend* edge labels similarly to node labels, thus resulting in edge labels that contain multiple roles.

As seen from Example 13, the result of applying the tableau expansion rules is not uniquely determined. If we choose to apply the $\sqcup$-Rule by adding the first disjunct to the label of v_0, we eventually obtain a *clash* $\bot \in L(v_1)$. We say that a tableau $T = (V, L)$ *contains a clash* if $\bot \in L(v)$ for some $v \in V$. Otherwise, we say that T is *clash-free*. A clash means that the tableau cannot correspond to an interpretation since $\bot \in L(v)$ corresponds to the requirement $v \in \bot^{\mathcal{I}} = \emptyset$, which cannot be fulfilled. In our example, the clash was obtained as a result of the "wrong choice" in the application of the $\sqcup$-Rule. When, instead, we choose the second disjunct, a clash-free tableau can be produced. We show next how to construct an interpretation from such a tableau.

Remark 2. Note that a clash $\bot \in L(v)$ may be produced by other rules than the $\bot$-Rule. For example, if $C \sqcap \bot \in L(v)$, then $\bot \in L(v)$ can be produced by the $\sqcap$-Rule. Similarly, if $\exists r.\bot \in L(v)$, then the clash is produced by the $\exists$-Rule.

Definition 3. *A tableau* $T = (V, L)$ defines an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *where:*

- $\Delta^{\mathcal{I}} = V$,
- $A^{\mathcal{I}} = \{x \in V \mid A \in L(x)\}$ *for each atomic concept* $A \in N_C$,
- $r^{\mathcal{I}} = \{\langle x, y\rangle \in V \times V \mid r \in L(x, y)\}$ *for each atomic role* $r \in N_R$.

Example 14. Consider the first tableau expansion from Example 13 (see the left of Fig. 2). This tableau defines an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = \{v_0, v_1\}$, $A^{\mathcal{I}} = \{v_1\}$, $B^{\mathcal{I}} = \emptyset$, and $r^{\mathcal{I}} = \{\langle v_0, v_1\rangle\}$. Let us calculate the values of the other concepts appearing in the label of the tableau under this interpretation:

- $(\neg A)^{\mathcal{I}} = \{v_0\}$,
- $(\exists r.A)^{\mathcal{I}} = \{v_0\}$,
- $(\forall r.(\neg A))^{\mathcal{I}} = \{v_1\}$,
- $(\forall r.(\neg A) \sqcup B)^{\mathcal{I}} = \{v_1\}$,
- $((\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B))^{\mathcal{I}} = \emptyset$.

As we can see, the interpretation $\mathcal{I}$ does not prove the satisfiability of the concept $C = (\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B)$, for which the tableau is constructed, since $C^{\mathcal{I}} = \emptyset$.

Let us now consider the interpretation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ defined by the second tableau expansion from Example 13 (see the right-hand side of Fig. 2): $\Delta^{\mathcal{J}} = \{v_0, v_1\}$, $A^{\mathcal{J}} = \{v_1\}$, $B^{\mathcal{J}} = \{v_0\}$, and $r^{\mathcal{J}} = \{\langle v_0, v_1\rangle\}$. It is easy to see that for this interpretation we have:

- $(\neg A)^{\mathcal{J}} = \{v_0\}$,
- $(\exists r.A)^{\mathcal{J}} = \{v_0\}$,
- $\forall r.(\neg A)^{\mathcal{J}} = \{v_1\}$,
- $(\forall r.(\neg A) \sqcup B)^{\mathcal{J}} = \{v_0, v_1\}$,
- $((\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B))^{\mathcal{J}} = \{v_0\}$.

Since $C^{\mathcal{J}} = \{v_0\} \neq \emptyset$, the interpretation $\mathcal{J}$ proves that C is satisfiable.

The satisfiability of the concept C proved in Example 14 using the interpretation for the second tableau expansion is not a coincidence. As we show next, in general, if the tableau rules can be applied without obtaining a clash, then each concept appearing in the label of each tableau node is satisfiable in the corresponding model.

Remark 3. Note that each rule in Table 2, with the exception of the $\bot$-Rule, can only add concepts to the labels if they are sub-concepts of some existing concept in the labels (to which the rule applies). Hence, every concept appearing in the labels of tableau nodes is a sub-concept of the original concept C for which the tableau is constructed or $\bot$. In particular, each such concept is in NNF. Similarly, only roles appearing in C can be added to the labels of the tableau edges.

Lemma 6. *Let $T = (V, L)$ be a clash-free, fully expanded tableau and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ an interpretation defined by T. Then, for every $v \in V$ and $C \in L(v)$, we have $v \in C^{\mathcal{I}}$.*

Proof. By Remark 3, the concept C is in NNF. We prove the lemma by induction on the construction of C according to the grammar definition (6):

Case $C = A \in L(v)$*:* In this case $v \in C^{\mathcal{I}} = A^{\mathcal{I}} = \{x \mid A \in L(x)\}$ by definition of $\mathcal{I}$.

Case $C = \top$*:* Then, trivially $v \in V = \Delta^{\mathcal{I}} = \top^{\mathcal{I}} = C^{\mathcal{I}}$.

Case $C = \bot \in L(v)$*:* Then T has a clash, which is not possible according to the assumption of the lemma.

Case $C = \neg A \in L(v)$*:* Then $A \notin L(v)$ since otherwise $\{A, \neg A\} \subseteq L(v)$ and $\bot \in L(v)$ since the $\bot$-Rule is not applicable to T, which would again mean that T has a clash. Since $A \notin L(v)$, we have $v \notin A^{\mathcal{I}}$ by definition of $\mathcal{I}$. Hence, $v \in \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} = (\neg A)^{\mathcal{I}}$.

Case $C = D \sqcap E \in L(v)$*:* Since the $\sqcap$-Rule is not applicable to $D \sqcap E \in L(v)$, we have $D \in L(v)$ and $E \in L(v)$. Then, by induction hypothesis, $v \in D^{\mathcal{I}}$ and $v \in E^{\mathcal{I}}$. Hence, $v \in D^{\mathcal{I}} \cap E^{\mathcal{I}} = (D \sqcap E)^{\mathcal{I}} = C^{\mathcal{I}}$.

Case $C = D \sqcup E \in L(v)$*:* Since the $\sqcup$-Rule is not applicable to $D \sqcup E \in L(v)$, we have $D \in L(v)$ or $E \in L(v)$. Then, by induction hypothesis, $v \in D^{\mathcal{I}}$ or $v \in E^{\mathcal{I}}$. Hence, $v \in D^{\mathcal{I}} \cup E^{\mathcal{I}} = (D \sqcup E)^{\mathcal{I}} = C^{\mathcal{I}}$.

Case $C = \exists r.D$*:* Since the $\exists$-Rule is not applicable to $\exists r.D \in L(v)$, there exists some $w \in V$ such that $r \in L(v, w)$ and $D \in L(w)$. From $r \in L(v, w)$, by definition of $\mathcal{I}$, we obtain $\langle v, w \rangle \in r^{\mathcal{I}}$. From $D \in L(w)$, by induction hypothesis, we obtain $w \in D^{\mathcal{I}}$. Hence, from $\langle v, w \rangle \in r^{\mathcal{I}}$ and $w \in D^{\mathcal{I}}$ we obtain $v \in (\exists r.D)^{\mathcal{I}}$.

Case $C = \forall r.D$*:* In order to prove that $v \in C^{\mathcal{I}} = (\forall r.D)^{\mathcal{I}}$, take any $w \in \Delta^{\mathcal{I}} = V$ such that $\langle v, w\rangle \in r^{\mathcal{I}}$. We need to show that $w \in D^{\mathcal{I}}$. Since $\langle v, w\rangle \in r^{\mathcal{I}}$, by definition of $\mathcal{I}$, we have $r \in L(v, w)$. Since the $\forall$-Rule is not applicable to $x = v$, $y = w$ and $\forall r.D \in L(v) = L(x)$, we must have $D \in L(y) = L(w)$. Hence, by induction hypothesis, $w \in D^{\mathcal{I}}$, as required. □

Corollary 1. *Let C be an $\mathcal{ALC}$ concept in NNF, and $T = (V, L)$ a clash-free, fully expanded tableau obtained by the tableau procedure for checking satisfiability for C. Then C is satisfiable.*

Proof. Due to the tableau initialization, and since the tableau rules in Table 2 never remove nodes or labels, we must have $C \in L(v)$ for some $v \in V$. Hence, by Lemma 6 $v \in C^{\mathcal{I}}$ for the interpretation $\mathcal{I}$ defined by T. Thus, $C^{\mathcal{I}} \neq \emptyset$ and C is satisfiable. □

Corollary 1 means that if we have managed to apply all tableau rules without producing a clash for a concept C in NNF, then we have proved that C is satisfiable. Does converse of this property also hold? Specifically, if C is satisfiable, is it always possible to apply the tableau rules without producing a clash? We prove that it is indeed the case by showing that if $C^{\mathcal{I}} \neq \emptyset$ for some interpretation $\mathcal{I}$, then we can always construct a tableau that *mimics* this interpretation in a certain way.

Definition 4. *We say say that a tableau $T = (V, L)$* mimics *an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ if there exists a mapping $\tau\colon V \to \Delta^{\mathcal{I}}$ such that:*

(1) for each $v \in V$ and each $C \in L(v)$, we have $\tau(v) \in C^{\mathcal{I}}$, and
(2) for each $\langle v, w\rangle \in V \times V$ and each $r \in L(v, w)$, we have $\langle \tau(v), \tau(w)\rangle \in r^{\mathcal{I}}$.

The mapping τ is called a mimic *of T in $\mathcal{I}$.*

For example, if $T = (V, L)$ is a clash-free fully expanded tableau, then T mimics the interpretation $\mathcal{I}$ defined by T (cf. Definition 3) since the identity mapping $\tau(v) = v \in V = \Delta^{\mathcal{I}}$ satisfies the requirements of Definition 4. Indeed, by Lemma 6, for every $C \in L(v)$, we have $\tau(v) = v \in C^{\mathcal{I}}$, and, by Definition 3, for every $\langle v, w\rangle \in V \times V$ and $r \in L(v, w)$, we have $\langle \tau(v), \tau(w)\rangle = \langle v, w\rangle \in r^{\mathcal{I}}$. However, a tableau T can also mimic other interpretations.

Example 15. Consider the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\Delta^{\mathcal{I}} = \{a\}$, $A^{\mathcal{I}} = B^{\mathcal{I}} = \{a\}$ and $r^{\mathcal{I}} = \{\langle a, a\rangle\}$ and the tableau $T = (V, L)$ obtained after the second expansion in Example 13 (see the right-hand side of Fig. 2). Then the mapping $\tau\colon V \to \Delta^{\mathcal{I}}$ defined by $\tau(v_0) = \tau(v_1) = a$ is a mimic of T in $\mathcal{I}$. Indeed, it is easy to see that $A^{\mathcal{I}} = B^{\mathcal{I}} = (\exists r.A)^{\mathcal{I}} = (\forall r.(\neg A) \sqcup B)^{\mathcal{I}} = ((\exists r.A) \sqcap (\forall r.(\neg A) \sqcup B))^{\mathcal{I}} = \{a\}$, hence, $\tau(v) = a \in C^{\mathcal{I}}$ for every $v \in V$ and every $C \in L(v)$. Also, since $\langle \tau(v_0), \tau(v_1)\rangle = \langle a, a\rangle \in r^{\mathcal{I}}$, Condition (2) of Definition 4 holds for $r \in L(v_0, v_1)$.

Note that if a tableau $T = (V, L)$ contains a clash $\bot \in L(v)$ for some $v \in V$, then T cannot mimic any interpretation $\mathcal{I}$, since, otherwise $\tau(v) \in \bot^{\mathcal{I}} = \emptyset$. Note

also that T can have a mimic even if it is not fully expanded. For example, if $C^{\mathcal{I}} \neq \emptyset$ for some concept C and interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, then the initial tableau $T = (V, L)$ with $V = \{v_0\}$ and $L(v_0) = \{C\}$ mimics $\mathcal{I}$ since for each $a \in C^{\mathcal{I}} \neq \emptyset$ and $\tau\colon V \to \Delta^{\mathcal{I}}$ defined by $\tau(v_0) = a$, we have $\tau(v_0) \in C^{\mathcal{I}}$. We will next show that in such a case T can always be expanded so that it still mimics $\mathcal{I}$.

Lemma 7. *Let $T = (V, L)$ be a tableau that mimics an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ and R be some tableau rule from Table 2 that is applicable to T. Then R can be applied in such a way that the resulting tableau also mimics $\mathcal{I}$.*

Proof. Suppose that $\tau\colon V \to \Delta^{\mathcal{I}}$ is a mimic of T in $\mathcal{I}$. We show how to apply R and extend τ to a mimic of the expanded tableau by considering all possible cases for R:

Case $\sqcap$-Rule: If the $\sqcap$-Rule is applicable to T, then $D \sqcap E \in L(v)$ for some $v \in V$. Since τ is a mimic of T in $\mathcal{I}$, we have $\tau(v) \in (D \sqcap E)^{\mathcal{I}}$. The application of the $\sqcap$-Rule only adds D and E to $L(v)$. To show that T still mimics $\mathcal{I}$ after this rule application, it is sufficient to prove that $\tau(v) \in D^{\mathcal{I}}$ and $\tau(v) \in E^{\mathcal{I}}$. This, clearly, follows from $\tau(v) \in (D \sqcap E)^{\mathcal{I}} = D^{\mathcal{I}} \cap E^{\mathcal{I}}$.

Case $\sqcup$-Rule: If the $\sqcup$-Rule is applicable to T, then $D \sqcup E \in L(v)$ for some $v \in V$. Since τ is a mimic of T in $\mathcal{I}$, we have $\tau(v) \in (D \sqcup E)^{\mathcal{I}} = D^{\mathcal{I}} \cup E^{\mathcal{I}}$. Hence, $\tau(v) \in D^{\mathcal{I}}$ or $\tau(v) \in E^{\mathcal{I}}$. We show that in each of these two cases one can apply the $\sqcup$-Rule so that the resulting tableau still mimics $\mathcal{I}$. Indeed, if $\tau(v) \in D^{\mathcal{I}}$, we can apply the $\sqcup$-Rule by adding D to $L(v)$. Since $\tau(v) \in D^{\mathcal{I}}$, τ is still a mimic of T in $\mathcal{I}$ after this rule application. Similarly, if $\tau(v) \in E^{\mathcal{I}}$, we can apply the $\sqcup$-Rule by adding E to $L(v)$. Since $\tau(v) \in E^{\mathcal{I}}$, τ remains a mimic of T in $\mathcal{I}$.

Case $\exists$-Rule: If the $\exists$-Rule is applicable to T, then $\exists r.D \in L(v)$ for some $v \in V$. Since τ is a mimic of T in $\mathcal{I}$, we have $\tau(v) \in (\exists r.D)^{\mathcal{I}}$. The application of the $\exists$-Rule adds a new node w to V with $L(v, w) = \{r\}$ and $L(w) = \{D\}$. To show that T still mimics $\mathcal{I}$ after this rule application, we define $\tau(w)$ such that Conditions (1) and (2) of Definition 4 hold for the two added labels. Specifically, since $\tau(v) \in (\exists r.D)^{\mathcal{I}}$, there exists some $d \in D^{\mathcal{I}}$ such that $\langle \tau(v), d \rangle \in r^{\mathcal{I}}$. Define $\tau(w) := d$. Then, since $\tau(w) = d \in D^{\mathcal{I}}$, Condition (1) of Definition 4 holds for $D \in L(v)$. Since $\langle \tau(v), \tau(w) \rangle = \langle \tau(v), d \rangle \in r^{\mathcal{I}}$, Condition (2) of Definition 4 holds for $r \in L(v, w)$.

Case $\forall$-Rule: If the $\forall$-Rule is applicable to T, then $\forall r.D \in L(v)$ for some $v \in V$ and $r \in L(v, w)$ for some $w \in V$. Since τ is a mimic of T in $\mathcal{I}$, we have $\tau(v) \in (\forall r.D)^{\mathcal{I}}$ and $\langle \tau(v), \tau(w) \rangle \in r^{\mathcal{I}}$. The application of the $\forall$-Rule adds D to $L(w)$. To show that T still mimics $\mathcal{I}$ after the rule application, it is sufficient to prove that $\tau(w) \in D^{\mathcal{I}}$, which clearly follows from $\tau(v) \in (\forall r.D)^{\mathcal{I}}$ and $\langle \tau(v), \tau(w) \rangle \in r^{\mathcal{I}}$.

Case $\bot$-Rule: If the $\bot$-Rule is applicable to T, then $\{A, \neg A\} \subseteq L(v)$ for some $v \in V$. Since τ is a mimic of T in $\mathcal{I}$, we have $\tau(v) \in A^{\mathcal{I}}$ and $\tau(v) \in (\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$. Clearly, this is not possible, which means that this case cannot occur. □

Algorithm 1. A tableau algorithm for checking satisfiability $\mathcal{ALC}$ concepts

CSat(C): Checking satisfiability of a concept C
input : an $\mathcal{ALC}$ concept C
output : yes if $C^{\mathcal{I}} \neq \emptyset$ for some interpretation $\mathcal{I}$ and **no** otherwise

```
1  C ← NNF(C);
2  V ← {v0}, L ← {v0 ↦ {C}};
3  T ← (V, L);
4  while not FullyExpanded(T) do
5  |  R ← ChooseApplicableRule(T);
6  |  T ← ApplyRule(T, R);
7  if ⊥ ∈ ⋃_{v∈V} L(v) then
8  |  return no;
9  else
10 |  return yes;
```

Algorithm 1 summarizes our tableau procedure for checking satisfiability of $\mathcal{ALC}$ concepts. After converting the input concept to NNF (Lines 1) and initializing the tableau (Line 2–3), the algorithm continuously applies the tableau rules from Table 2 until the tableau is fully expanded (Lines 4–6). If the resulting tableau contains a clash (Line 7) the algorithm returns *no*; if not, the algorithm returns *yes*.

Note that due to the $\sqcup$-Rule, the result of applying a rule (Line 6) is not uniquely determined. Hence, Algorithm 1 is *non-deterministic*. We next show that this algorithm solves the concept satisfiability problem for $\mathcal{ALC}$, i.e., it is *correct*. Recall (see Appendix A.1) that a non-deterministic algorithm A *solves* a problem $P\colon X \to \{yes, no\}$ if, for each $x \in X$ such that $P(x) = no$, each run of A terminates with the result *no*, and for each $x \in X$ such that $P(x) = yes$, there exists *at least one run* for which the algorithm terminates and produces *yes*. Proving correctness of a (non-deterministic) algorithm is usually accomplished by proving several properties: (1) *Soundness*: if for an input $x \in X$, A returns *yes* then $P(x) = yes$, (2) *Completeness*: if $P(x) = yes$ then A returns *yes* for at least one run, and (3) *Termination*: A terminates for every input. Soundness of Algorithm 1 follows from Lemma 6 since the algorithm returns *yes* only if a clash-free fully expanded tableau is computed. Completeness follows from Lemma 7 since, for a satisfiable concept, one can always apply the rules such that a clash is avoided. It is thus remains to prove that Algorithm 1 always terminates.

Remark 4. It may seem that a run of Algorithm 1 is determined not only by the choice of a possible expansion of the non-deterministic $\sqcup$-Rule (Line 6), but also by the choice of which next rule to apply in case there are several applicable rules (Line 5). However, since Lemma 7 holds for any applicable rule, the latter choice does not have any impact on the completeness of the algorithm. In other words, we may assume that the function **ChooseApplicableRule**(T)

is *deterministic*, i.e., it always returns the same value for the same input (unlike function **ApplyRule**(T, R) for which different values need to be considered in different runs). The choice of *which* next rule to apply is usually referred to as *don't care non-determinism* of the algorithm, whereas the choice of *how* a rule is applied (the $\sqcup$-Rule in our case) is referred to as a *don't know non-determinism*.

Exercise 6. The function **ChooseApplicableRule**(T) of Algorithm 1 can be defined in many different ways. If several rules are applicable to a node, e.g., the $\sqcap$-Rule and the $\sqcup$-Rule, the function may determine which of these rules should be applied first by specifying a *rule precedence*. If the same rule is applicable to different nodes, the function, likewise, can choose to which node the rule should be applied first. Discuss, which of these choices are more likely to result in fewer and/or shorter runs of the tableau procedure?

In order to prove termination of Algorithm 1, we show that the size of the tableau T that is constructed for a concept C at each step of the algorithm is bounded by an exponential function in the size of C (the number of symbols in C). This implies that every run of Algorithm 1 terminates after at most exponentially many rule applications since each rule application increases the size of the tableau. By Remark 3, each node label can contain only concepts that are sub-concepts of C or $\bot$, and each edge label can contain only roles that appear in C. Therefore, the maximal size of the label for each node and edge for each pair of nodes is bounded by a linear function in the size of C. We next show that the number of different nodes of a tableau is at most exponential in the size of C.

Definition 5. *For each node $v \in V$ of a tableau $T = (V, L)$, we define its* level *$\ell(v)$ by induction on the rule application of the tableau procedure:*

- *For the node v_0 created during tableau initialization we set $\ell(v_0) = 0$;*
- *For a node w created by an application of the $\exists$-Rule to a node $v \in V$, we set $\ell(w) = \ell(v) + 1$.*

The following lemma gives a bound on the number of nodes at each level of a tableau:

Lemma 8. *Let $T = (V, L)$ be a (possibly not fully expanded) tableau obtained for a concept C of size n. Then for each $k \geq 0$, the number of nodes v with $\ell(v) = k$ is bounded by n^k.*

Proof. The proof is by induction over $k \geq 0$.

Case $k = 0$: There exists only one node $v = v_0 \in V$ with $\ell(v) = 0$ since all other nodes are constructed by the $\exists$-Rule.

Case $k > 0$: Take any node $w \in V$ with $\ell(w) = k$. Since $k > 0$, w can only be constructed by an application of the $\exists$-Rule to some node v with $\ell(v) = k - 1$. This rule has been applied to some concept $\exists r.D \in L(v)$ and, after this rule application, the $\exists$-Rule can no longer be applied to $\exists r.D \in L(v)$. Hence, each

w with $\ell(v) = k$ is uniquely associated with a pair $\langle \exists r.D, v\rangle$ where $\exists r.D$ is a sub-concept of the original concept C and $\ell(v) = k - 1$. Since the number of sub-concepts of C is bounded by n and, by the induction hypothesis, the number of nodes v with $\ell(v) = k - 1$ is bounded by n^{k-1}, the number of nodes w with $\ell(v) = k$ is bounded by $n \cdot n^{k-1} = n^k$. □

Finally, we prove that the level of a tableau node cannot exceed the quantifier depth of the input concept.

Definition 6. *The* quantifier depth *of an $\mathcal{ALC}$ concept C is a number $qd(C)$ that is defined inductively over (1) as follows:*

- $qd(\top) = qd(\bot) = qd(A) = 0$ *for each* $A \in N_C$,
- $qd(C \sqcap D) = qd(C \sqcup D) = \max(qd(C), qd(D))$,
- $qd(\neg C) = qd(C)$,
- $qd(\exists r.C) = qd(\forall r.C) = qd(C) + 1$.

Example 16.

$$\begin{aligned} qd(\exists r.((\neg A) \sqcap \forall r.B)) &= qd((\neg A) \sqcap \forall r.B) + 1 \\ &= \max(qd(\neg A),\, qd(\forall r.B)) + 1 \\ &= \max(qd(A),\, qd(B) + 1) + 1 \\ &= \max(0,\, 0 + 1) + 1 = 2. \end{aligned}$$

Note that $qd(C)$ is not greater than the number of quantifier symbols ($\exists$ or $\forall$) in C, which is not greater than the length of C.

Lemma 9. *Let $T = (V, L)$ be a (possibly not fully expanded) tableau obtained for a concept C with $qd(C) = q$, and $v \in V$ a node with $\ell(v) = k$. Then for each $D \in L(v)$, we have $qd(D) \leq q - k$.*

Proof. We prove the lemma by induction on the size (i.e., on the construction) of T.

If T is created during the tableau initialization, then $D = C$, $v = v_0$, and $k = 0$. Hence $qd(D) = qd(C) = q = q - k$ as required.

Otherwise, T was created by applying one of the tableau expansion rules in Table 2. If $D \in L(v)$ was not added by this rule, we can apply the induction hypothesis to the (smaller) tableau before the rule application. Otherwise, we consider all possible cases of such a rule that can add $D \in L(v)$:

Case $\sqcap$-Rule: If D was added by the $\sqcap$-Rule, then, before this rule application, $D \sqcap E \in L(v)$ or $E \sqcap D \in L(v)$ for some E. By applying the induction hypothesis for this concept, we obtain $qd(D) \leq qd(D \sqcap E) \leq q - k$ or $qd(D) \leq qd(E \sqcap D) \leq q - k$.

Case $\sqcup$-Rule: If D was added by the $\sqcup$-Rule, then, before this rule application, $D \sqcup E \in L(v)$ or $E \sqcup D \in L(v)$ for some E. By applying the induction hypothesis for this concept, we obtain $qd(D) \leq qd(D \sqcup E) \leq q - k$ or $qd(D) \leq qd(E \sqcup D) \leq q - k$.

Case $\exists$*-Rule:* If D was added by the $\exists$-Rule, then the node v was also created by this rule and the rule was applied to some $w \in V$ with $\exists r.D \in L(w)$ for some role r. Then $\ell(v) = \ell(w) + 1$ and, by induction hypothesis, $qd(\exists r.D) \leq q - \ell(w) = q - (\ell(v) - 1) = q - k + 1$. Since $qd(\exists r.D) = qd(D) + 1$, we obtain $qd(D) = qd(\exists r.D) - 1 \leq q - k$.

Case $\forall$*-Rule:* If D was added by the $\forall$-Rule, then this rule was applied to some $w \in V$ with $\forall r.D \in L(w)$ and $r \in L(w, v)$. Since $r \in L(w, v)$ could be only added by the $\exists$-Rule, $\ell(v) = \ell(w) + 1$. By induction hypothesis for $\forall r.D \in L(w)$, we obtain $qd(\forall r.D) \leq q - \ell(w) = q - (\ell(v) - 1) = q - k + 1$. Since $qd(\forall r.D) = qd(D) + 1$, we obtain $qd(D) = qd(\forall r.D) - 1 \leq q - k$.

Case $\bot$*-Rule:* If $D = \bot$ was added by the $\bot$-Rule, then, before this rule application, we have $\{A, \neg A\} \subseteq L(v)$ for some atomic concept A. By induction hypothesis, $qd(A) \leq q - k$. Hence $qd(\bot) = 0 = qd(A) \leq q - k$. □

Let $T = (V, L)$ be a tableau constructed during a run of the tableau procedure for a concept C with size n and $qd(C) = q \leq n$. Since every node $v \in V$ of a tableau $T = (V, L)$ always contains at least one concept D in the label, by Lemma 9 it follows that $0 \leq qd(D) \leq q - \ell(v)$. Hence $\ell(v) \leq q$ holds for every $v \in V$. Since, by Lemma 8 the number of nodes at level k is bounded by n^k, the total number of nodes in the tableau is bounded by $\sum_{0 \leq k \leq q} n^k \leq n^{q+1} = 2^{(\log_2 n) \cdot (q+1)} \leq 2^{n^2}$.

Essentially we have shown that the tableau procedure constructs a special kind of interpretation (represented by the tableau). The interpretations have a *tree shape*: each node except for the initial node (the *root* of the tree) is connected by an edge to exactly one *predecessor* node from which this node was created (by an application of the $\exists$-Rule). The depth of the tree is bounded by the quantifier depth of the concept C for which the tableau was constructed. The branching degree of the tree (the maximal number of successor nodes of each node) is bounded by the number of existential concepts occurring in C. Hence the size of the tree is bounded *exponentially* in the size of C.

Exercise 7. Show that the exponential upper bound on the size of the tableau cannot be improved. Specifically, for each $n \geq 0$ construct a concept C_n of polynomial size in n (i.e., the number of symbols in C_n is bounded by $p(n)$ for some polynomial function p) such that the fully expanded tableau for C_n contains at least 2^n nodes. Hint: the tableau rules should create a binary tree of depth n. Hence by Lemma 9, $qd(C_n) \geq n$. The label of each non-leaf node should contain two different concepts of the form $\exists R.D$.

The exponential bound on the tableau size implies the following complexity result:

Theorem 1. *Algorithm 1 solves the concept satisfiability problem in* $\mathcal{ALC}$ *in non-deterministic exponential time.*

Remark 5. It is possible to make some further improvements to Algorithm 1 to prove better complexity bounds. First note that to check satisfiability of a concept C, it is not necessary to keep the whole tableau in memory. Once all tableau expansion rules are applied to a node (and the node is checked for the presence of a clash), the node can be completely deleted. By processing nodes in a depth-first manner it is, therefore, possible to store at most linearly many nodes in memory at any given time (because the tableau has a linear depth). This gives a non-deterministic *polynomial space* algorithm solving this problem. A well-known result from complexity theory called Savitch's theorem then implies that there is a *deterministic* polynomial-space algorithm solving this problem, which is now an optimal complexity bound for checking concept satisfiability in $\mathcal{ALC}$ (see, e.g., [6, Sect. 5.1.1]).

3.2 TBox Reasoning

In this section, we extend the tableau procedure presented earlier to also take into account TBox axioms of the ontology. Given an $\mathcal{ALC}$ concept C and an $\mathcal{ALC}$ TBox $\mathcal{O}$, our goal now is to check the satisfiability of C w.r.t. $\mathcal{O}$, i.e., to check if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$.

As in the case of the previous procedure, before applying the tableau rules, we first need to convert the input into a suitable normal form. We say that a TBox axiom is in *normal form* (or is *normalized*) if it is a concept inclusion axiom of the form $\top \sqsubseteq D$ where D is a concept in NNF. Every TBox axiom can be converted into the normal form by applying the following simple rewriting rules:

$$
\begin{array}{lcll}
C \equiv D & \Rightarrow & C \sqsubseteq D, \quad D \sqsubseteq C, & \\
C \sqsubseteq D & \Rightarrow & \top \sqsubseteq \neg C \sqcup D & \text{if } C \neq \top, \\
\top \sqsubseteq D & \Rightarrow & \top \sqsubseteq \mathrm{NNF}(D) & \text{if } D \text{ is not in NNF.}
\end{array}
$$

Exercise 8. Similarly to Exercise 5, show that TBox normalization preserves concept satisfiability w.r.t. the TBox. That is, a concept C is satisfiable w.r.t. a TBox $\mathcal{O}$ if and only if C is satisfiable w.r.t. the normalization of $\mathcal{O}$. Hint: show that for each rewrite step $\alpha \Rightarrow \beta$ above and each interpretation $\mathcal{I}$ we have $\mathcal{I} \models \alpha$ if and only if $\mathcal{I} \models \beta$.

To take the resulting axioms into account in our tableau procedure, we need to add an additional tableau expansion rule shown in Table 3. We can now use a simple modification of Algorithm 1, where, in addition to a (normalized) concept C, the input also contains a (normalized) ontology $\mathcal{O}$ and, in addition to the rules in Table 2, a new rule from Table 3 can be chosen and applied at Steps 5 and 6.

Example 17. Consider $C = A$ and $\mathcal{O} = \{A \sqcap \forall r.B \sqsubseteq \exists r.A\}$. We check satisfiability of C w.r.t. $\mathcal{O}$ using the tableau procedure. The concept C is already in NNF. The axiom in $\mathcal{O}$ is normalized as follows:

Table 3. The additional tableau expansion rule for handling (normalized) TBox axioms

Rule	Conditions	Expansions
$\top$-Rule	$\top \sqsubseteq D \in \mathcal{O}$, $D \notin L(x)$	Set $L(x) := L(x) \cup \{D\}$

$$A \sqcap \forall r.B \sqsubseteq \exists r.A \quad \Rightarrow \quad \top \sqsubseteq \neg(A \sqcap \forall r.B) \sqcup \exists r.A,$$
$$\Rightarrow \quad \top \sqsubseteq ((\neg A) \sqcup \exists r.\neg B) \sqcup \exists r.A.$$

The tableau is initialized to $T = (V, L)$ with $V = \{v_0\}$ and $L(v_0) = \{A\}$, and expanded by the following rule applications:

1. $\top$-Rule: $L(v_0) := L(v_0) \cup \{((\neg A) \sqcup \exists r.\neg B) \sqcup \exists r.A\}$,
2. $\sqcup$-Rule: $L(v_0) := L(v_0) \cup \{(\neg A) \sqcup \exists r.\neg B\}$,
3. $\sqcup$-Rule: $L(v_0) := L(v_0) \cup \{\exists r.\neg B\}$,
4. $\exists$-Rule: $L(v_0, v_1) := \{r\}$, $L(v_1) := \{\neg B\}$,
5. $\top$-Rule: $L(v_1) := L(v_1) \cup \{((\neg A) \sqcup \exists r.\neg B) \sqcup \exists r.A\}$,
6. $\sqcup$-Rule: $L(v_1) := L(v_1) \cup \{(\neg A) \sqcup \exists r.\neg B\}$,
7. $\sqcup$-Rule: $L(v_1) := L(v_1) \cup \{\neg A\}$.

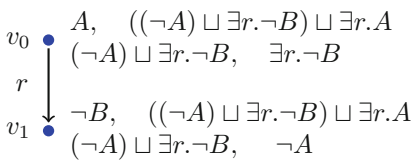


Fig. 3. A possible tableau expansion for $C = A$ and $\mathcal{O} = \{\top \sqsubseteq (\neg A \sqcup \exists r.\neg B) \sqcup \exists r.A\}$

Figure 3 shows the resulting tableau expansion. Note that the new $\top$-Rule is applied for both nodes v_0 and v_1 (Steps 1 and 5). Without applying this rule, no other rule would be applicable. Note that at Step 3 we have applied the $\sqcup$-Rule by adding the second disjunct $\exists r.\neg B$ to $L(v_0)$ because adding the first disjunct $\neg A$ would result in a clash since $A \in L(v_0)$. The same disjunction also appears in $L(v_1)$, but since $A \notin L(v_1)$, we could apply the $\sqcup$-Rule by adding the first disjunct $\neg A$ to $L(v_1)$ (Step 7).

Since after Step 7 the tableau is fully expanded and does not contain a clash, we conclude that C is satisfiable w.r.t. $\mathcal{O}$.

Intuitively, the new $\top$-Rule ensures that the interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ defined by $T = (V, L)$ (see Definition 3) satisfies all axioms in the (normalized) ontology $\mathcal{O}$ once all tableau rules are applied. Indeed, Lemma 6 still holds for the extended tableau procedure, since the proof of the lemma is by induction on the construction of a concept $C \in L(v)$ and we did not add any new concept

constructors. Now, since T is fully expanded, for every normalized axiom $\top \sqsubseteq D \in \mathcal{O}$ and every $v \in V$, we have $D \in L(v)$ due to the $\top$-Rule. Hence, by Lemma 6, $v \in D^{\mathcal{I}}$ for every $v \in V = \Delta^{\mathcal{I}}$. Consequently, $\mathcal{I} \models \top \sqsubseteq D$ for each $\top \sqsubseteq D \in \mathcal{O}$. This implies that the extended Algorithm 1 remains sound.

Completeness of the extension of Algorithm 1 can also easily be shown. If C is satisfiable w.r.t. $\mathcal{O}$, then there exists a model $\mathcal{I} \models \mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$. We extend the proof of Lemma 7 to show that in this case, one can apply the tableau rules in such a way that T always mimics $\mathcal{I}$, thus, avoiding the production of a clash. For this we just need to update the proof with the case for the newly added rule:

Case $\top - Rule$: If the $\top$-Rule is applicable to T, then $D \notin L(v)$ for some $v \in V$ and some $\top \sqsubseteq D \in \mathcal{O}$. The application of the $\top$-Rule adds only D to $L(v)$. To show that T still mimics $\mathcal{I}$ after this rule application, it is sufficient to prove that $\tau(v) \in D^{\mathcal{I}}$. Since $\top \sqsubseteq D \in \mathcal{O}$ and $\mathcal{I} \models \mathcal{O}$ we have $\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Hence $\tau(v) \in \Delta^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

Table 4. An additional expansion rule to handle TBox axioms of the form $C \sqsubseteq D$

Rule	Conditions	Expansions
$\sqsubseteq$-Rule	$C \sqsubseteq D \in \mathcal{O}$, $C \in L(x)$, $D \notin L(x)$	Set $L(x) := L(x) \cup \{D\}$

Exercise 9. In order to understand why the TBox axioms require a transformation to the form $\top \sqsubseteq D$, suppose we generalize the normal form to also permit axioms $C \sqsubseteq D$ where both C and D are in NNF, and formulate a new $\sqsubseteq$-Rule to handle axioms of this form as given in Table 4. Does the modified tableau algorithm remain sound and complete? Which of the lemmas cannot be proved any longer?

What happens if we only allow normalized axioms of the form $\top \sqsubseteq D$ and $A \sqsubseteq D$ where A is an atomic concept and D is a concept in NNF. Is the tableau algorithm with the $\top$-Rule and the $\sqsubseteq$-Rule sound and complete for this case?

Exercise 10. Description logic $\mathcal{ALCH}$ is an extension of the description logic $\mathcal{ALC}$, in which ontologies can contain *role inclusion axioms* of the form $r \sqsubseteq s$, where r and s are roles. An interpretation $\mathcal{I}$ satisfies $r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$.

Extend the tableau procedure by adding a new rule to handle role inclusion axioms. Prove that this procedure is sound and complete. Use this procedure to show that the concept $C = A \sqcap \neg\exists s.(A \sqcap B) \sqcap \forall r.B$ is unsatisfiable w.r.t. $\mathcal{O} = \{A \sqsubseteq \exists r.A, r \sqsubseteq s\}$.

We have shown that the tableau algorithm extended with the $\top$-Rule remains sound and complete. In order to show that it solves the concept satisfiability problem w.r.t. TBoxes, it remains to show that it terminates for every input. Unfortunately, the latter is not the case as shown in the next example. Intuitively,

since the $\top$-Rule adds a concept to every node label, this new concept can, in particular, trigger an application of the $\exists$-Rule, which, in turn, creates new nodes, for which the $\top$-Rule is applicable again.

Example 18. Consider $C = A$ and $\mathcal{O} = \{A \sqsubseteq \exists r.A\}$. We check the satisfiability of C w.r.t. $\mathcal{O}$ using the extended tableau procedure. The concept C is already in NNF, so we just need to normalize the axiom in $\mathcal{O}$:

$$A \sqsubseteq \exists r.A \quad \Rightarrow \quad \top \sqsubseteq (\neg A) \sqcup \exists r.A.$$

The tableau is initialized to $T = (V, L)$ with $V = \{v_0\}$ and $L(v_0) = \{A\}$, and expanded as shown in Fig. 4. Notice that unlike in Example 17, if we were to apply the $\sqcup$-Rule for $(\neg A) \sqcup \exists r.A \in L(v_1)$ by choosing the first disjunct $\neg A$, we would trigger a clash since $L(v_1)$ also contains A (added by $\exists$-Rule). Hence, the creation of infinitely many tableau nodes cannot be avoided.

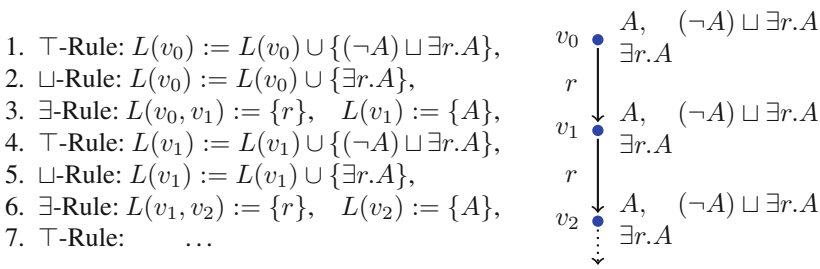


Fig. 4. The only clash-free tableau expansion for $C = A$ and $\mathcal{O} = \{\top \sqsubseteq (\neg A) \sqcup \exists r.A\}$

As discussed in Remark 5, to check satisfiability of a concept (also, with respect to an ontology), it is not necessary to keep the whole tableau in memory. We just need to verify that a clash-free tableau *exists*. This idea can be developed further to regain termination of the tableau algorithm with TBoxes. Notice that in Example 18, all nodes contain identical concepts in the labels. This means that if a rule is applicable to one node then it can be applied in exactly the same way to any other node with the same content. Hence, if a clash is obtained in this node, it is also obtained in the other node. Consequently, it is not necessary to apply the tableau rules to all nodes in order to verify that there exists a clash-free tableau expansion. Some rule applications can be *blocked*.

Definition 7. *A* blocking condition *is a function that assigns to every tableau* $T = (V, L)$ *a nonempty subset* $W \subsetneq V$ *of* active *nodes such that for every node* $v_1 \in W$ *and every node* $v_2 \notin W$ *such that* $L(v_1, v_2) \neq \emptyset$*, there exists a node* $w \in W$ *with* $L(v_2) \subseteq L(w)$*. In this case we say that a node* v_2 *is* (directly) blocked *by node* w*. Each node in* $V \setminus W$ *is called a* blocked *node.*

Algorithm 2. A tableau algorithm for checking satisfiability of $\mathcal{ALC}$ concepts with respect to $\mathcal{ALC}$ ontologies

COSat(C, $\mathcal{O}$): Checking satisfiability of a concept C w.r.t. an ontology $\mathcal{O}$
input : an $\mathcal{ALC}$ concept C, an $\mathcal{ALC}$ ontology $\mathcal{O}$
output : **yes** if $C^{\mathcal{I}} \neq \emptyset$ for some model $\mathcal{I}$ of $\mathcal{O}$ and **no** otherwise

```
1  C ← NNF(C);
2  O ← Normalize(O);
3  V ← {v0}, L ← {v0 ↦ {C}};
4  T ← (V, L);
5  while not FullyExpandedUpToBlocking(T) do
6  |   R ← ChooseApplicableRule(T);
7  |   T ← ApplyRule(T, R);
8  if ⊥ ∈ ⋃_{v∈V} L(v) then
9  |   return no;
10 else
11 |   return yes;
```

Example 19. Let $T = (V, L)$ be the tableau obtained after Step 6 in Example 18, i.e., $V = \{v_0, v_1, v_2\}$, $L(v_0) = L(v_1) = \{A, (\neg A) \sqcup \exists r.A, \exists r.A\}$, $L(v_2) = \{A\}$, and $L(v_0, v_1) = L(v_1, v_2) = \{r\}$. Then a blocking condition for T can be defined by setting $W = \{v_0\}$ since for $\langle v_0, v_1\rangle \in E$ we have $L(v_1) \subseteq L(v_0)$.

We leave it open, how exactly the set of active nodes of a tableau is determined, so that different blocking strategies can be used in different algorithms. We show next that one can restrict the tableau algorithm to apply rules only to active nodes.

Definition 8. *Let $T = (V, L)$ be a tableau with a subset $W \subseteq V$ of active nodes. We say that a tableau rule from Tables 2 or 3 is* applicable to a node $v \in V$ *if the conditions of this rule are satisfied for a mapping $x \mapsto v$. We say that a tableau T is* fully expanded up to blocking *if no tableau rule is applicable to any active node $w \in W$.*

Example 20. It is easy to see that the tableau T from Example 19 with $W = \{v_0\}$ is fully expanded up to blocking since no tableau rule (for the ontology $\mathcal{O}$ from Example 18) is applicable to v_0.

Algorithm 2 is a modification of Algorithm 1 for checking satisfiability of concepts w.r.t. ontologies. Apart from a new step for normalization of the input ontology (Line 1), the algorithm uses a blocking condition to verify if the tableau is fully expanded up to blocking according to Definition 8 (Line 5), and to select a rule applicable to an active node (Line 6). We assume that the initial node v_0 always remains active.

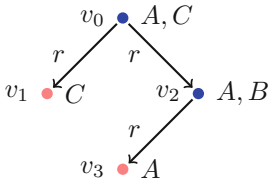
We next show that the updated algorithm remains sound and complete. The introduction of a blocking condition does not have any impact on completeness:

the proof of Lemma 7 (including the new case for the $\top$-Rule) remains as before. Soundness of the new algorithm is, however, nontrivial: if the tableau is fully expanded with blocking, it does not mean that it is fully expanded without blocking. To prove soundness, we modify Definition 3 by taking the blocking condition into account:

Definition 9. *A tableau $T = (V, L)$ and a subset of active nodes $W \subseteq V$* define an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *such that:*

- $\Delta^{\mathcal{I}} = W$,
- $A^{\mathcal{I}} = \{x \in W \mid A \in L(x)\}$ *for each atomic concept* $A \in N_C$,
- $r^{\mathcal{I}} = \{\langle x, y\rangle \in W \times W \mid \exists z \in V : r \in L(x, z)$ *and* $z = y$ *or* z *is blocked by* $y\}$ *for each atomic role* $r \in N_R$.

A few comments about Definition 9 are in order. First note that this definition coincides with Definition 3 when $W = V$. The definition of $r^{\mathcal{I}}$ can be explained as follows: if $r \in L(x, z)$ and both nodes x and z are active, then $\langle x, z\rangle \in r^{\mathcal{I}}$. This corresponds to the case '$z = y$' of the definition for $r^{\mathcal{I}}$. If x active but z is not, then, since $L(x, z) \neq \emptyset$, z should be blocked by some $y \in W$. In this case, $r^{\mathcal{I}}$ contains all such pairs $\langle x, y\rangle$. This corresponds to the case 'z is blocked by y' of the definition for $r^{\mathcal{I}}$. If x not active then the label $r \in L(x, z)$ is ignored.



$\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where:

- $\Delta^{\mathcal{I}} = \{v_0, v_2\}$,
- $A^{\mathcal{I}} = \{v_0, v_2\}$, $B^{\mathcal{I}} = \{v_2\}$, $C^{\mathcal{I}} = \{v_0\}$,
- $r^{\mathcal{I}} = \{\langle v_0, v_0\rangle, \langle v_0, v_2\rangle, \langle v_2, v_2\rangle, \langle v_2, v_0\rangle\}$.

Fig. 5. A tableau with blocking (v_0 and v_2 are are active nodes, v_1 and v_3 are blocked nodes) and the interpretation defined by this tableau

Example 21. Consider the tableau $T = (V, L)$ illustrated on the left-hand side of Fig. 5:

- $V = \{v_0, v_1, v_2, v_3\}$,
- $L(v_0) = \{A, C\}$, $L(v_1) = \{C\}$, $L(v_2) = \{A, B\}$, $L(v_3) = \{A\}$,
- $L(v_0, v_1) = L(v_0, v_2) = L(v_2, v_3) = \{r\}$.

Suppose that the set of active nodes is $W = \{v_0, v_2\}$. Note that v_1 is blocked by v_0 since $L(v_1) = \{C\} \subseteq \{A, C\} = L(v_0)$, v_3 is blocked by v_2 since $L(v_3) = \{A\} \subseteq \{A, B\} = L(v_2)$, and v_3 is blocked by v_0 since $L(v_3) = \{A\} \subseteq \{A, C\} = L(v_0)$. Then T and W define an interpretation shown in the right of Fig. 5.

Let $T = (V, L)$ be a tableau obtained by applying the tableau expansion rules for a concept C and an ontology $\mathcal{O}$. Suppose that T is fully expanded up to blocking for $W \subseteq V$ and does not contain a clash. Let $\mathcal{I}$ be the interpretation defined by T and W according to Definition 9. Our goal is to show that $\mathcal{I} \models \mathcal{O}$ and $C^{\mathcal{I}} \neq \emptyset$, which implies that C is satisfiable w.r.t. $\mathcal{O}$. To do this, we prove an analog of Lemma 6 for our new interpretation $\mathcal{I}$.

Lemma 10. *Let $T = (V, L)$ be a clash-free, fully expanded tableau up to blocking for $W \subseteq V$ and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ an interpretation defined by T and W according to Definition 9. Then for every $v \in W$ and every $C \in L(v)$, we have $v \in C^{\mathcal{I}}$.*

Proof. Just like for Lemma 6, we prove this lemma by induction on the construction of C according to the grammar definition (6). The only changes compared to the previous proof are those cases where the definition of $r^{\mathcal{I}}$ was used:

Case $C = \exists r.D$: Since the $\exists$-Rule is not applicable to $\exists r.D \in L(v)$, there exists some $w \in V$ such that $r \in L(v, w)$ and $D \in L(w)$. Define an active node $w' \in W$ as follows. If $w \in W$, we set $w' = w$. Otherwise, w must be blocked by some $w' \in W$. Then, by the definition of $r^{\mathcal{I}}$, we have $\langle v, w' \rangle \in r^{\mathcal{I}}$. Note also that $D \in L(w) \subseteq L(w')$. Since $w' \in W$, by induction hypothesis $w' \in D^{\mathcal{I}}$. From $\langle v, w' \rangle \in r^{\mathcal{I}}$ and $w' \in D^{\mathcal{I}}$ we obtain $v \in (\exists r.D)^{\mathcal{I}}$.

Case $C = \forall r.D$: In order to prove that $v \in C^{\mathcal{I}} = (\forall r.D)^{\mathcal{I}}$, take any $w' \in \Delta^{\mathcal{I}} = W$ such that $\langle v, w' \rangle \in r^{\mathcal{I}}$. We prove that $w' \in D^{\mathcal{I}}$. Since $\langle v, w' \rangle \in r^{\mathcal{I}}$, by definition of $r^{\mathcal{I}}$, there exists some $w \in V$ such that $r \in L(v, w)$ and either $w = w'$ or w is blocked by w'. In both cases $L(w) \subseteq L(w')$. Since the $\forall$-Rule is not applicable to $\forall r.D \in L(v)$ and $r \in L(v, w)$, we must have $D \in L(w) \subseteq L(w')$. Since $w' \in W$, from $D \in L(w')$ by induction hypothesis, we obtain $w' \in D^{\mathcal{I}}$, as required. □

Exercise 11. Identify the places where the properties of a blocking condition (Definition 7) have been used in the proof of Lemma 10. Can the blocking condition be relaxed in such a way that more nodes of a tableau can potentially be blocked, but the proof of Lemma 10 can be still repeated? For example, do we really need that *all concepts* of $L(v_2)$ are contained in $L(w)$?

Finally, we consider the question of termination of Algorithm 2. Clearly, the algorithm does not terminate for every blocking condition. For example, as shown in Example 18, the algorithm does not terminate if all nodes are active, i.e., $W = V$. Hence, we need to make some further assumptions about the blocking condition in order to show termination.

Definition 10. *The* eager blocking *condition (for a root node w_0) assigns to every tableau $T = (V, L)$ a minimal (w.r.t. set inclusion) set of active nodes $W \subseteq V$ containing w_0 that satisfies the condition of Definition 7.*

Intuitively, the eager blocking condition for $T = (V, L)$ can be implemented as follows. We first set $W = \{w_0\}$. Then, repeatedly for every $v_1 \in W$ and

$v_2 \in V \setminus W$ such that $L(v_1, v_2) \neq \emptyset$, check if there exists $w \in W$ such that $L(v_2) \subseteq L(w)$. If there is no such element, we add v_2 to W. This process continues until no further nodes can be added. Note that the resulting set W depends on the order in which the nodes v_2 are processed. In practice, the set W can dynamically be updated when applying tableau rules. The eager blocking condition is related to the notion of *anywhere blocking* [42].

Lemma 11. *Let C and $\mathcal{O}$ be inputs of Algorithm 2 with the combined size n (i.e., the total number of symbols). Then each run of Algorithm 2 terminates in at most doubly exponential time in the size of n provided an eager blocking condition is used.*

Proof. Without loss of generality, we may assume that C and $\mathcal{O}$ are normalized since this step can be performed in linear time. Let $T = (V, L)$ be a tableau obtained during the run of the algorithm. For each node $v \in V$ of the tableau, we define its level $\ell(n)$ as in Definition 5. We will prove that $\ell(v) \leq 2^n$ for each node $v \in V$.

Assume to the contrary that there exists $w \in V$ with $\ell(w) = 2^n + 1$. Then w must have been created by the $\exists$-Rule from some $v \in W$ with $\ell(v) = \ell(w) - 1 = 2^n$, where $W \subseteq V$ is the set of active nodes of the tableau at the moment of this rule application. Since $\ell(v) = 2^n$, there must exist nodes $v_0, v_1, \ldots, v_{2^n} = v$ such that $L(v_{i-1}, v_i) \neq \emptyset$ $(1 \leq i \leq 2^n)$. It is easy to see that $v_i \in W$ for all i with $0 \leq i \leq 2^n$. Indeed, otherwise there exists a maximal such i such that $v_i \notin W$ $(0 \leq i \leq 2^n)$. Since $v_{2^n} = v \in W$, then $i < 2^n$ and $v_0 \neq v_{i+1} \in W$. But then one can remove v_{i+1} from W without violating the conditions of Definition 7 since $L(w, v_{i+1}) = \emptyset$ for all $w \in W$. This contradicts our assumption that W is a minimal set of active nodes containing v_0.

Now consider the sets of concepts $S_i = L(v_i)$ in the labels of v_i $(0 \leq i \leq 2^n)$. Since each node label can contain only concepts that appear either in C or in $\mathcal{O}$ (possibly as sub-concepts) and the number of such concepts is bounded by the total combined length n of the input, there can be at most 2^n different subsets among S_i $(0 \leq i \leq 2^n)$. By the pigeonhole principle, there exist some indexes i and j $(0 \leq i < j \leq 2^n)$ such that $S_i = S_j$. But then node $v_j \neq v_0$ can be removed from W without violating the conditions of Definition 7 since for each $v \in W$ with $L(v, v_j) \neq \emptyset$, there exists $w = v_i \in W$ such that $L(v_i) \subseteq L(w)$ because $L(w) = L(v_i) = L(v_j)$. This again contradicts our assumption that W is a minimal set of active nodes. The obtained contradiction, therefore, proves that $\ell(v) \leq 2^n$ for every $v \in V$.

Now, by Lemma 8, the total number of tableau nodes is bounded by $\sum_{0 \leq k \leq 2^n} n^k \leq n^{2^n+1} = 2^{(\log_2 n) \cdot (2^n+1)} \leq 2^{2^{n^2}}$. Since each node contains at most n concept labels and every application of a tableau rule introduces at least one of them, each run of Algorithm 2 terminates after at most double exponentially many steps. □

Theorem 2. *Algorithm 2 solves the concept satisfiability problem with respect to ontologies expressed in $\mathcal{ALC}$ in non-deterministic doubly exponential time.*

As with Algorithm 1, the complexity bound provided by Algorithm 2 is not optimal and can be improved to *deterministic exponential* time (see, e.g., [6, Sect. 5.1.2]).

Note that the requirements about the blocking condition used in Lemma 11 can be relaxed. Indeed, in the proof of the lemma we only used that a node v_j is directly blocked by an *ancestor* node v_i, i.e., a node from which v_j was created by a sequence of $\exists$-Rule applications.

Exercise 12 (Advanced). Is it possible to improve the upper bound shown in the proof of Lemma 11 to *single exponential* time? Which additional assumptions about the blocking condition are necessary for this? Hint: for every tableau $T = (V, L)$ used in the computation, consider the set of all subsets of the labels of the nodes: $P = \{S \subseteq L(v) \mid v \in V\}$. How can this set change after a tableau rule application? How many times can this set change during the tableau run? What is the maximal possible number of consequent rule applications that do not change this set?

4 Axiom Pinpointing

In Sect. 2.3, we have discussed several ontology reasoning problems and how they can help in detecting modeling errors in ontologies. For example, inconsistency of an ontology indicates that the modeled domain cannot match any model of the ontology since the ontology does not have models. In Sect. 3 we have shown how to check ontologies for consistency and solve other reasoning problems using tableau procedures. Knowing that an ontology is inconsistent, however, does not tell much about what exactly *causes* the inconsistency let alone how to *repair* it.

Recall from Sect. 2.3, that all reasoning problems can be reduced to concept subsumption checking. For example, by Lemma 5, an ontology $\mathcal{O}$ is unsatisfiable if and only if $\mathcal{O} \models \top \sqsubseteq \bot$. *Axiom pinpointing* methods can help the user to identify the exact axioms that are responsible for this or any other entailment.

Definition 11. *A* justification *for an entailment $\mathcal{O} \models \alpha$ is a subset of axioms $J \subseteq \mathcal{O}$ such that $J \models \alpha$ and for every $J' \subsetneq J$, we have $J' \not\models \alpha$.*

In other words, a justification for an entailment $\mathcal{O} \models \alpha$ is a minimal set of axioms of the ontology that entails α. Note that since $\mathcal{O} \models \alpha$, at least one justification for the entailment exists. Indeed, either $J_0 = \mathcal{O}$ satisfies the condition of Definition 11, or there exits $J_1 \subsetneq J_0$ such that $J_1 \models \alpha$. Similarly, either J_1 is a justification or there exists some $J_2 \subsetneq J_1$ such that $J_2 \models \alpha$, etc. At some point this process stops since $\mathcal{O}$ contains only finitely many axioms and J_i $(i \geq 0)$ gets smaller with every step. Therefore, the last set J_k will be a justification for $\mathcal{O} \models \alpha$.

Note that we say *a* justification instead of *the* justification. Indeed, Definition 11 does not imply that a justification must be unique as the following example shows.

Example 22. Consider the following entailment:

$$\mathcal{O} = \{A \sqsubseteq B,\ B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\} \models \alpha = A \sqsubseteq C.$$

This entailment has three different justifications:

- $J_1 = \{A \sqsubseteq B,\ B \sqsubseteq C\}$,
- $J_2 = \{A \sqsubseteq C\}$,
- $J_3 = \{A \sqsubseteq B,\ A \sqcap B \sqsubseteq \bot\}$.

Indeed, it is easy to see that $J_i \models \alpha$ for $1 \leq i \leq 3$. For example, $J_3 \models A \sqsubseteq C$ because for every model $\mathcal{I} \models J_3$ we have $A^{\mathcal{I}} \subseteq A^{\mathcal{I}} \cap B^{\mathcal{I}} \subseteq \bot^{\mathcal{I}} = \emptyset \subseteq C^{\mathcal{I}}$. We can show that each J_i satisfies the remaining condition of Definition 11 by enumerating all proper subsets of J_i $(1 \leq i \leq 3)$:

- J_1 has only the proper subsets $M_0 = \emptyset$, $M_1 = \{A \sqsubseteq B\}$ and $M_2 = \{B \sqsubseteq C\}$,
- J_2 has only the proper subset $M_0 = \emptyset$,
- J_3 has only the proper subsets $M_0 = \emptyset$, $M_1 = \{A \sqsubseteq B\}$, and $M_3 = \{A \sqcap B \sqsubseteq \bot\}$.

We can show that none of these subsets M_i entails α by presenting the corresponding counter-models $\mathcal{I}_i = (\Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i})$ such that $\mathcal{I}_i \models M_i$ but $\mathcal{I}_i \not\models \alpha$ $(0 \leq i \leq 3)$:

- For $M_0 = \emptyset$ take $\mathcal{I}_0 = (\Delta^{\mathcal{I}_0}, \cdot^{\mathcal{I}_0})$ with $\Delta^{\mathcal{I}_0} = \{a\}$, $A^{\mathcal{I}_0} = \{a\}$ and $C^{\mathcal{I}_0} = \emptyset$. Clearly, $\mathcal{I}_0 \models M_0$ but $\mathcal{I}_0 \not\models A \sqsubseteq C$ since $A^{\mathcal{I}_0} = \{a\} \not\subseteq \emptyset = C^{\mathcal{I}_0}$.
- For $M_1 = \{A \sqsubseteq B\}$ take $\mathcal{I}_1 = (\Delta^{\mathcal{I}_1}, \cdot^{\mathcal{I}_1})$ with $\Delta^{\mathcal{I}_1} = \{a\}$, $A^{\mathcal{I}_1} = B^{\mathcal{I}_1} = \{a\}$, and $C^{\mathcal{I}_1} = \emptyset$. Clearly, $\mathcal{I}_1 \models M_1$ because $A^{\mathcal{I}_1} = \{a\} \subseteq \{a\} = B^{\mathcal{I}_1}$ but $\mathcal{I}_1 \not\models A \sqsubseteq C$ similarly as for $\mathcal{I}_0$.
- For $M_2 = \{B \sqsubseteq C\}$ take $\mathcal{I}_2 = (\Delta^{\mathcal{I}_2}, \cdot^{\mathcal{I}_2})$ with $\Delta^{\mathcal{I}_2} = \{a\}$, $A^{\mathcal{I}_2} = \{a\}$, and $B^{\mathcal{I}_2} = C^{\mathcal{I}_2} = \emptyset$. Clearly, $\mathcal{I}_2 \models M_2$ because $B^{\mathcal{I}_2} = \emptyset \subseteq \emptyset = C^{\mathcal{I}_2}$ but $\mathcal{I}_2 \not\models A \sqsubseteq C$ similarly as for $\mathcal{I}_0$ and $\mathcal{I}_1$.
- For $M_3 = \{A \sqcap B \sqsubseteq \bot\}$ take $\mathcal{I}_3 = \mathcal{I}_2$ from the previous case. Clearly, $\mathcal{I}_2 \models M_3$ because $(A \sqcap B)^{\mathcal{I}_2} = A^{\mathcal{I}_2} \cap B^{\mathcal{I}_2} = \{a\} \cap \emptyset = \emptyset \subseteq \bot^{\mathcal{I}_2} = \emptyset$ but $\mathcal{I}_2 \not\models A \sqsubseteq C$.

How many justifications may an entailment have? The following example shows that the number of justifications can be exponential in the size of the ontology.

Example 23. Consider the following ontology $\mathcal{O}$ and $\alpha = A_0 \sqsubseteq A_n$:

$$\mathcal{O} = \{A_{i-1} \sqsubseteq B \sqcap A_i,\ A_{i-1} \sqsubseteq C \sqcap A_i \mid 1 \leq i \leq n\},$$

where A_i $(0 \leq i \leq n)$, B, and C are atomic concepts. Note that, for each i with $1 \leq i \leq n$, we have $\mathcal{O} \models A_{i-1} \sqsubseteq A_i$ because $A_{i-1} \sqsubseteq B \sqcap A_i \models A_{i-1} \sqsubseteq A_i$ (or $A_{i-1} \sqsubseteq C \sqcap A_i \models A_{i-1} \sqsubseteq A_i$). Consequently, $\mathcal{O} \models \alpha = A_0 \sqsubseteq A_n$. However, there are 2^n minimal subsets $J \subseteq \mathcal{O}$ such that $J \models \alpha$. Indeed, since each axiom $A_{i-1} \sqsubseteq A_i$ follows from two different axioms, J must include one of them for each i $(1 \leq i \leq n)$. Hence, there are 2^n possible variants for each J.

Algorithm 3. Minimizing entailment

Minimize($\mathcal{O}, \alpha$): compute a justification for $\mathcal{O} \models \alpha$
input : an ontology $\mathcal{O}$ and an axiom α such that $\mathcal{O} \models \alpha$
output : a minimal subset $J \subseteq \mathcal{O}$ such that $J \models \alpha$ (cf. Definition 11)

```
1 J ← O;
2 for β ∈ O do
3     if J \ {β} ⊨ α then
4         J ← J \ {β};
5 return J;
```

Specifically, let $S \subseteq \{i \mid 1 \leq i \leq n\}$ be any subset of indexes between 1 and n. There are in total 2^n such subsets. For each such subset S, define

$$J_S = \{A_{i-1} \sqsubseteq B \sqcap A_i \mid i \in S\} \cup \{A_{i-1} \sqsubseteq C \sqcap A_i \mid i \notin S\} \subseteq \mathcal{O}.$$

Clearly, $J_{S_1} \neq J_{S_2}$ for each $S_1 \neq S_2$. Furthermore, note that for each i with $1 \leq i \leq n$, we have $J_S \models A_{i-1} \sqsubseteq A_i$: if $i \in S$ then $J \ni A_{i-1} \sqsubseteq B \sqcap A_i \models A_{i-1} \sqsubseteq A_i$; if $i \notin S$ then $J \ni A_{i-1} \sqsubseteq C \sqcap A_i \models A_{i-1} \sqsubseteq A_i$. Hence $J_S \models \alpha$.

To show that each J_S is a justification for $\mathcal{O} \models \alpha$, it remains to show that $J' \not\models \alpha$ for every $J' \subsetneq J_S$. Indeed, if $J' \subsetneq J_S$ then for some k with $1 \leq k \leq n$, we have $A_{k-1} \sqsubseteq B \sqcap A_k \notin J_S$ and $A_{k-1} \sqsubseteq C \sqcap A_k \notin J_S$. Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation with $\Delta^{\mathcal{I}} = \{a\}$, $A_i^{\mathcal{I}} = B^{\mathcal{I}} = C^{\mathcal{I}} = \{a\}$ for $0 \leq i < k$, and $A_i^{\mathcal{I}} = \emptyset$ for $k \leq i \leq n$. It is easy to see that $\mathcal{I} \models A_{i-1} \sqsubseteq B \sqcap A_i$ and $\mathcal{I} \models A_{i-1} \sqsubseteq C \sqcap A_i$ for each i with $1 \leq i < k$ or with $k < i \leq n$. Indeed, if $1 \leq i < k$, then $A_{i-1}^{\mathcal{I}} = \{a\} \subseteq \{a\} \cap \{a\} = (B \sqcap A_i)^{\mathcal{I}} = (C \sqcap A_i)^{\mathcal{I}}$. If $k < i \leq n$, then $A_{i-1}^{\mathcal{I}} = \emptyset \subseteq \{a\} \cap \emptyset = (B \sqcap A_i)^{\mathcal{I}} = (C \sqcap A_i)^{\mathcal{I}}$. Hence, $\mathcal{I} \models J'$. Since $A_0^{\mathcal{I}} = \{a\} \not\subseteq \emptyset = A_n^{\mathcal{I}}$, we have $\mathcal{I} \not\models \alpha$. Consequently, $J' \not\models \alpha$.

Assuming we have an algorithm for testing entailment of axioms, e.g., the tableau procedure described in Sect. 3, we are now concerned with the question of how to *compute* justifications in particular for the entailment of concept inclusions.

4.1 Computing One Justification

Computing one justification for $\mathcal{O} \models \alpha$ is relatively easy. Starting from $J = \mathcal{O}$, we repeatedly remove axioms from J if this does not break the entailment $J \models \alpha$. At a certain point, no axioms can be removed without breaking the entailment, which implies that J is justification for $\mathcal{O} \models \alpha$. Algorithm 3 summarizes this idea.

Example 24. The following table shows a run of Algorithm 3 on the input $\mathcal{O}$ and α from Example 22. Each row of the table shows the value of the variables J and β in the beginning of each for-loop iteration (Lines 2–4). The last column

shows the result of the evaluation of the if-statement in Line 3. The last line shows the (returned) value of J after the last iteration of the loop.

J	β	$J \setminus \{\beta\} \models^? \alpha = A \sqsubseteq C$
$\{A \sqsubseteq B,\ B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\}$	$A \sqsubseteq B$	*yes*
$\{B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\}$	$B \sqsubseteq C$	*yes*
$\{A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\}$	$A \sqsubseteq C$	*no*
$\{A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\}$	$A \sqcap B \sqsubseteq \bot$	*yes*
$\{A \sqsubseteq C\}$	-	-

As we can see, the algorithm returns the justification $J_2 = \{A \sqsubseteq C\}$.

The correctness of Algorithm 3 relies on the fact that the entailment relation $\mathcal{O} \models \alpha$ between an ontology $\mathcal{O}$ and an axioms α is *monotonic* over axiom additions to $\mathcal{O}$:

Lemma 12. *Let J_1 and J_2 be two sets of $\mathcal{ALC}$ axioms such that $J_1 \subseteq J_2$, and α an $\mathcal{ALC}$ axiom. Then $J_1 \models \alpha$ implies $J_2 \models \alpha$.*

Proof. It is equivalent to show that $J_2 \not\models \alpha$ implies $J_1 \not\models \alpha$. If $J_2 \not\models \alpha$ then there exists a model $\mathcal{I} \models J_2$ such that $\mathcal{I} \not\models \alpha$. Since $J_1 \subseteq J_2$ and $\mathcal{I} \models J_2$, we have $\mathcal{I} \models J_1$. Since $\mathcal{I} \models J_1$ and $\mathcal{I} \not\models \alpha$, we obtain $J_1 \not\models \alpha$, as required. □

Note that in the proof of Lemma 12 we did not rely on any specific constructors of $\mathcal{ALC}$. We have only used that the entailment relation $J \models \alpha$ is defined by means of interpretations, i.e., $J \models \alpha$ iff $\mathcal{I} \models \alpha$ for every $\mathcal{I} \models J$ and $\mathcal{I} \models J$ iff $\mathcal{I} \models \beta$ for every $\beta \in J$. Although most standard DLs (including those that underpin the OWL standard) have such a *classical semantics*, there are some *non-monotonic DLs* in which the entailment relation is defined in other ways, e.g., as a result of performing certain operations [11,14,18]. From now on we assume that we deal only with monotonic (classical) entailment relations. We show that Algorithm 3 is correct in such cases.

Theorem 3. *Let J be the output of Algorithm 3 for the input $\mathcal{O}$ and α such that $\mathcal{O} \models \alpha$. Then J is a justification for $\mathcal{O} \models \alpha$.*

Proof. Clearly, $J \models \alpha$ since we only assign subsets that entail α to the variable J (in Lines 1 and 4). If J is not a justification for $\mathcal{O} \models \alpha$, by Definition 11 there exists some $J' \subsetneq J$ such that $J' \models \alpha$. Since $J' \subsetneq J$, there exists some $\beta \in J \setminus J' \subseteq \mathcal{O}$. Let J'' be the value of the variable J of Algorithm 3 at the beginning of the for-loop (Line 2) when $\beta \in \mathcal{O}$ was processed. Since $\beta \notin J' \subseteq J \subseteq J''$, we have $J' \subseteq J \setminus \{\beta\} \subseteq J'' \setminus \{\beta\}$. Since $J' \models \alpha$, by Lemma 12, $J'' \setminus \{\beta\} \models \alpha$. Hence β must have been removed from J'' at Line 4, and consequently, $\beta \notin J$. This contradicts $\beta \in J \setminus J'$, which proves that there is no $J' \subsetneq J$ such that $J' \models \alpha$. Hence J is a justification for $\mathcal{O} \models \alpha$. □

Finally, observe that a run of Algorithm 3 requires exactly n subsumption tests. Hence, the complexity of computing one justification is bounded by a linear function over the complexity of entailment checking. In particular, one justification for concept subsumptions in $\mathcal{ALC}$ can be computed in exponential time.

4.2 Computing All Justifications

A justification for $\mathcal{O} \models \alpha$ contains axioms that are responsible for *one* reason for the entailment. As we have seen in Examples 22 and 23, there can be several different justifications. To repair an unwanted entailment $\mathcal{O} \models \alpha$, it is, therefore, necessary to change an axiom in every justification of $\mathcal{O} \models \alpha$. How do we compute *all* justifications?

Note that the output of Algorithm 3 depends on the *order* in which the axioms in $\mathcal{O}$ are enumerated in the for-loop (Line 2). Different orders of the axioms can result in different removals and, consequently, different justifications.

Example 25. Consider the run of Algorithm 3 on the input $\mathcal{O}$ and α from Example 22, where the axioms in $\mathcal{O}$ are enumerated in the reverse order as in Example 24.

J	β	$J \setminus \{\beta\} \models^? \alpha = A \sqsubseteq C$
$\{A \sqcap B \sqsubseteq \bot,\ A \sqsubseteq C,\ B \sqsubseteq C,\ A \sqsubseteq B\}$	$A \sqcap B \sqsubseteq \bot$	*yes*
$\{A \sqsubseteq C,\ B \sqsubseteq C,\ A \sqsubseteq B\}$	$A \sqsubseteq C$	*yes*
$\{B \sqsubseteq C,\ A \sqsubseteq B\}$	$B \sqsubseteq C$	*no*
$\{B \sqsubseteq C,\ A \sqsubseteq B\}$	$A \sqsubseteq B$	*no*
$\{B \sqsubseteq C,\ A \sqsubseteq B\}$	-	-

In this case, the algorithm returns the justification $J_1 = \{A \sqsubseteq B,\ B \sqsubseteq C\}$.

Exercise 13. For which order of axioms in $\mathcal{O}$ does Algorithm 3 return the justification $J_3 = \{A \sqsubseteq B,\ A \sqcap B \sqsubseteq \bot\}$ from Example 22?

Exercise 14. Prove that for each justification J of an entailment $\mathcal{O} \models \alpha$ there exists some order of axioms in $\mathcal{O}$ for which Algorithm 3 with the input $\mathcal{O}$ and α returns J.

The property stated in Exercise 14 means that for computing all justifications of $\mathcal{O} \models \alpha$, it is sufficient to run Algorithm 3 for all possible orders of axioms in $\mathcal{O}$. Since the number of permutations of elements in an n-element set is $n!$,[4] the

[4] $n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$, there are n possibilities to choose the first element, $n-1$ to choose the second element from the remaining ones, $n-2$ to choose the third one, etc.

algorithm terminates after exactly $n \cdot n!$ entailment tests; since $n \cdot n! \leq n^{n+1} = 2^{(log_2 n)\cdot(n+1)} \leq 2^{n^2}$, this value is bounded by an exponential function in n. As shown in Example 23, the number of justifications can be exponential in n, so the exponential behavior of an algorithm for computing *all* justifications cannot be avoided, in general. Unfortunately, the described algorithm is not very practical since it performs exponentially many subsumption tests for *all* inputs, even if, e.g., $\mathcal{O} \models \alpha$ has just one justification, which is $\mathcal{O}$ itself. This is because this algorithm is *not goal-directed*: the computation of each next justification does not depend on the justifications computed before.

How can we find a more goal-directed algorithm? Suppose that we have computed a justification J_1 using Algorithm 3. The next justification J_2 must be different from J_1, so J_2 should miss at least one axiom from J_1. Hence the next justification J_2 can be found by finding $\beta_1 \in J_1$ such that $\mathcal{O} \setminus \{\beta_1\} \models \alpha$ and calling Algorithm 3 for the input $\mathcal{O} \setminus \{\beta_1\}$ and α. The next justification J_3, similarly, should miss something from J_1 and something from J_2, so it can be found by finding some $\beta_1 \in J_1$ and $\beta_2 \in J_2$ such that $\mathcal{O} \setminus \{\beta_1, \beta_2\} \models \alpha$ and calling Algorithm 3 for the input $\mathcal{O} \setminus \{\beta_1, \beta_2\}$ and α. In general, when justifications J_i $(1 \leq i \leq k)$ are computed, the next justification can be found by calling Algorithm 3 for the input $\mathcal{O} \setminus \{\beta_i \mid 1 \leq i \leq k\}$ and α such that $\beta_i \in J_i$ $(1 \leq i \leq k)$ and $\mathcal{O} \setminus \{\beta_i \mid 1 \leq i \leq k\} \models \alpha$. Enumeration of subsets $\mathcal{O} \setminus \{\beta_i \mid 1 \leq i \leq k\}$ can be organized using a data structure called a hitting set tree.

Definition 12. *A* hitting set tree *(short:* HS-tree*) for the entailment $\mathcal{O} \models \alpha$ is a labeled tree $T = (V, E, L)$ with $V \neq \emptyset$ such that:*

1. *each non-leaf node $v \in V$ is labeled with a justification $L(v) = J$ for $\mathcal{O} \models \alpha$ and, for each $\beta \in J$, v has an outgoing edge $\langle v, w\rangle \in E$ with label $L(v, w) = \beta$*
2. *each leaf node $v \in V$ is labeled by a special symbol $L(v) = \bot$.*

For each $v \in V$ let $H(v)$ be the set of edge labels appearing on the path from v to the root node of H. Then the following properties should additionally hold:

3. *for each non-leaf node $v \in V$ we have $L(v) \cap H(v) = \emptyset$,*
4. *for each leaf node $v \in V$ we have $\mathcal{O} \setminus H(v) \not\models \alpha$.*

Figure 6 shows an example of two different HS-trees for the entailment from Example 22. Note that the justification J_2 labels two different nodes of the left tree. We next prove that every HS-tree must contain every justification at least once.

Lemma 13. *Let $T = (V, E, L)$ be an HS-tree for the entailment $\mathcal{O} \models \alpha$. Then, for each justification J for $\mathcal{O} \models \alpha$, there exists a node $v \in V$ such that $L(v) = J$.*

Proof. Let $v \in V$ be a node with a maximal (w.r.t. set inclusion) set $H(v)$ (see Definition 12) such that $H(v) \cap J = \emptyset$, i.e., for every other node $w \in V$ either $H(w) \subseteq H(v)$ or $H(w) \cap J \neq \emptyset$. We prove that $L(v) = J$.

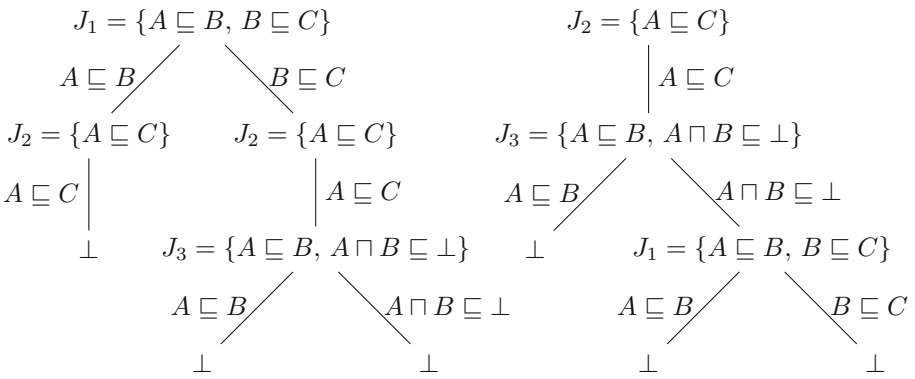


Fig. 6. Two HS-trees for $\mathcal{O} = \{A \sqsubseteq B,\ B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\} \models \alpha = A \sqsubseteq C$

Observe that since $H(v) \cap J = \emptyset$ and $J \subseteq \mathcal{O}$, we have $J \subseteq \mathcal{O} \setminus H(v)$. Since $J \models \alpha$, by monotonicity of entailment, we obtain $\mathcal{O} \setminus H(v) \models \alpha$. Therefore, by Condition 4 of Definition 12, v cannot be a leaf node. Hence, $L(n) = J'$ for some justification J' of $\mathcal{O} \models \alpha$. If $J = J'$ we have proved what is required. Otherwise, since J is a justification for $\mathcal{O} \models \alpha$ and $J' \models \alpha$, we have $J' \not\subseteq J$. Hence, there exists some $\beta \in J' \setminus J$. By Condition 1 of Definition 12, there exists $\langle v, w\rangle \in E$ with $L(v, w) = \beta$. Furthermore, by Condition 3 of Definition 12, $J' \cap H(v) = \emptyset$. Hence, $\beta \notin H(v)$ since $\beta \in J'$. Hence, $H(w) = H(v) \cup \{\beta\} \not\subseteq H(v)$ and, since $\beta \notin J$ and $H(v) \cap J = \emptyset$, we have $H(w) \cap J = \emptyset$. This contradicts our assumption that $H(v)$ is a maximal set such that $H(v) \cap J = \emptyset$. This contradiction proves that $L(n) = J$ is the only possible case. □

We next show that each HS-tree $T = (V, E, L)$ for an entailment $\mathcal{O} \models \alpha$ has at most exponentially many nodes in the number of axioms in $\mathcal{O}$. Take any $\langle v, w\rangle \in E$. Then v is not a leaf node. Hence, by Condition 1 of Definition 12, $L(v) = J$ for some justification J for $\mathcal{O} \models \alpha$ and $L(v, w) \in J$. By Condition 3, $J \cap H(v) = \emptyset$. Hence $L(v, w) \notin H(v)$. This implies that for each node $v \in V$ each axiom $\beta \in H(v)$ appears on the path from v to the root node exactly once. Hence the depth of H is bounded by the maximal number of axioms in $H(v)$, which is bounded by the number of axioms in $\mathcal{O}$. Similarly, since each non-leaf node v has exactly one successor for every $\beta \in L(v) \subseteq \mathcal{O}$, the branching factor of H is also bounded by the number of axioms in $\mathcal{O}$. This analysis gives us the following bound on the size of T:

Lemma 14. *Every HS-tree T for $\mathcal{O} \models \alpha$ has at most $\sum_{0 \le k \le n} n^k$ nodes where n is the number of axioms in $\mathcal{O}$.*

Exercise 15. Prove that T has at most $(n+1)!$ nodes. Hint: show that every path of T from the root to the leaf has a unique sequence of axioms on the labels of edges. Is this bound better than the bound from Lemma 14?

An HS-tree $T = (V, E, L)$ for an entailment $\mathcal{O} \models \alpha$ can be constructed as follows. We start by creating the root node $v_0 \in V$. Then we repeatedly assign labels of nodes and edges as follows. For each $v \in V$, if $L(v)$ was not yet assigned, we calculate $H(v)$. If $\mathcal{O} \setminus H(v) \not\models \alpha$, we label $L(v) = \bot$ according to Condition 4 of Definition 12. Otherwise, we compute a justification J for $\mathcal{O} \setminus H(v) \models \alpha$ using Algorithm 3 and set $L(v) = J$. Note that J satisfies Condition 3 of Definition 12 since $J \subseteq \mathcal{O} \setminus H(v)$. Next, for each $\beta \in J$, we create a successor node w of v and label $L(v, w) = \beta$. This ensures that Condition 1 of Definition 12 is satisfied for v. Since, by Lemma 14, H has a bounded number of nodes, this process eventually terminates. The described algorithm is known as Reiter's *Hitting Set Tree algorithm* (or short: *HST-algorithm*) [22,46].

Exercise 16. Construct an HS-tree for the entailment $\mathcal{O} \models \alpha$ from Example 23 for the parameter $n = 2$ using the HST-algorithm.

Note that we call Algorithm 3 exactly once per node. Then Lemma 14 gives us the following bound on the number of entailment tests performed by the HST-algorithm:

Lemma 15. *An HS-tree for an entailment $\mathcal{O} \models \alpha$ can be constructed using at most $\sum_{1 \leq k \leq n+1} n^k$ entailment tests, where n is the number of axioms in $\mathcal{O}$.*

Note that unlike the algorithm sketched in Exercise 14, the input for each call of Algorithm 3 depends on the results returned by the previous calls.

Exercise 17. Suppose that an entailment $\mathcal{O} \models \alpha$ has a single justification $J = \mathcal{O}$. How many entailment tests will be performed by the HST-algorithm if $\mathcal{O}$ contains n axioms?

The HST-algorithm can further be optimized in several ways. First, it is not necessary to store the complete HS-tree in memory. For computing a justification J at each node v, it is sufficient to know just the set $H(v)$. For each successor w of v associated with some $\beta \in H(v)$, the set $H(w)$ can be computed as $H(w) = H(v) \cup \{\beta\}$. Hence, it is possible to compute all justifications by recursively processing and creating the sets $H(v)$ as shown in Algorithm 4. The algorithm saves all justifications in a set S, which is initially empty (Line 1). The justifications are computed by processing the sets $H(v)$; the sets that are not yet processed are stored in the queue Q, which initially contains $H(v_0) = \emptyset$ for the root node v_0 (Line 2). The elements of Q are then repeatedly processed in a loop (Lines 3–10) until Q becomes empty. First, we choose any $H \in Q$ (Line 4) and remove it from Q (Line 5). Then, we test whether $\mathcal{O} \setminus H \models \alpha$ (Line 6). If the entailment holds, this means that the corresponding node v of the HS-tree with $H(v) = H$ is not a leaf node. We then compute a justification J using Algorithm 3 and add it to S (Lines 7–8). Further, for each $\beta \in J$, we create the set $H(w) = H(v) \cup \{\beta\}$ for the corresponding successor node w of v and add $H(w)$ to Q for later processing (Lines 9–10). If the entailment $\mathcal{O} \setminus H \models \alpha$ does not hold, this means that we have reached a leaf of the HS-tree and no further children of this node should be created.

Algorithm 4. Computing all justifications by the Hitting Set Tree algorithm

ComputeJustificationsHST($\mathcal{O}, \alpha$): compute all justifications for $\mathcal{O} \models \alpha$
input : an ontology $\mathcal{O}$ and an axiom α such that $\mathcal{O} \models \alpha$
output : the set of all minimal subsets $J \subseteq \mathcal{O}$ such that $J \models \alpha$

```
1  S ← ∅;
2  Q ← {∅};
3  while Q ≠ ∅ do
4      H ← choose H ∈ Q;
5      Q ← Q \ {H};
6      if O \ H ⊨ α then
7          J ← Minimize(O \ H, α);
8          S ← S ∪ {J};
9          for β ∈ J do
10             Q ← Q ∪ {H ∪ {β}};
11 return S;
```

Exercise 18. Prove directly that Algorithm 4 returns *all* justifications for the given entailment $\mathcal{O} \models \alpha$. For this, show that the following invariant always holds in the main loop (Lines 3–10): if J is a justification for $\mathcal{O} \models \alpha$, then either $J \in S$ or there exists $H \in Q$ such that $J \subseteq \mathcal{O} \setminus H$.

Note that it is not specified in which *order* the elements should be taken from Q in Line 4 of Algorithm 4. Indeed, *correctness* of the algorithm does not depend on the order in which the sets $H \in Q$ are processed. However, *performance* of the algorithm may depend on this order. If the elements of Q are processed according to the *first-in-first-out (short: FIFO)* strategy, i.e., we take elements in the order in which they were inserted to the queue, this means that the nodes of the HS-tree are processed in a *breadth-first* way, so the queue may contain *exponentially many* unprocessed sets H at some point in time. If the elements of Q are processed according to the *last-in-first-out (short: LIFO)* strategy, i.e., we remove elements in the reversed order in which they were inserted, this means that the nodes of the HS-tree are processed in a *depth-first* way, so the queue always contains at most n^2 sets (at most n sets for nodes of some tree path plus at most $n - 1$ other successors for each of these nodes). Consequently, the LIFO strategy should be more memory efficient. This optimization is related to the optimization discussed in Remark 5, which allows for executing a tableau procedure in polynomial space.

A few further optimizations can be used to improve the *running time* of Algorithm 4 in certain cases. Note that some justifications can be computed multiple times as shown for the example on the left-hand side of Fig. 6. It is possible to *detect* such repetitions by checking if any justification $J \in S$ computed so far is a subset of $\mathcal{O} \setminus H$ for the currently processed set H. In this case, a (potentially

expensive) call of Algorithm 3 in Line 7 of Algorithm 4 can be avoided by reusing J. Of course, testing $J \subseteq \mathcal{O} \setminus H$ for *all* $J \in S$ can also be expensive since S may contain exponentially many justifications. In practice, it makes sense to perform this test only for small J, for which the test is more likely to succeed. Another possible repetition is when some $H \in Q$, which was already processed before, is processed again. In this case, not only the previously computed justification $J \subseteq \mathcal{O} \setminus H$ can be reused, but it is also not necessary to create the successor sets $H \cup \{\beta\}$ for $\beta \in J$ since also those sets should have been created before. Of course, to check if a set H was processed before, we need to save all previously processed sets H, which is not done in the base version of Algorithm 4 since this information can increase the memory consumption of the algorithm. Hence, this is an example of an optimization that trades memory consumption for potentially improving the running time. Another optimization is to test if H is a superset of some set H' for which the test $\mathcal{O} \setminus H' \models \alpha$ was *negative*. Clearly, in this case, the test $\mathcal{O} \setminus H \models \alpha$ is negative as well by monotonicity of the entailment, so such H can immediately be disregarded by the algorithm without performing this test.

4.3 Computing All Repairs

The main idea of the HST-algorithm is to systematically compute two kinds of sets: (1) justifications J for the entailment $\mathcal{O} \models \alpha$ and (2) sets H that contain one element from each justification J on a branch. The name of the algorithm comes from the notion of a hitting set, which characterizes the latter sets.

Definition 13. *Let P be a set of sets of some elements. A set H is a* hitting set *for P if $H \cap S \neq \emptyset$ for each $S \in P$. A hitting set H for P is* minimal *if every $H' \subsetneq H$ is not a hitting set for P.*

Intuitively, a hitting set for P is a set H that contains at least one element from every set $S \in P$. An HS-tree is then a tree $T = (V, E, L)$ such that for each $v \in V$, $H(v)$ is a hitting set of the set of justifications on the path from v to the root of T. The leaf nodes v of T are labeled by hitting sets $H(v)$ such $\mathcal{O} \setminus H(v) \not\models \alpha$. Intuitively, the set $H(v)$ represents a set such that the removal of $H(v)$ from $\mathcal{O}$ breaks the entailment $\mathcal{O} \models \alpha$.

Definition 14. *A set R is a* repair *for the entailment $\mathcal{O} \models \alpha$ if $\mathcal{O} \setminus R \not\models \alpha$. A* minimal repair *for $\mathcal{O} \models \alpha$ is a repair R such that for every $R' \subsetneq R$ we have $\mathcal{O} \setminus R \models \alpha$.*

Notice some similarities between the notion of a minimal repair and the notion of a justification (cf. Definition 11): justifications are minimal subsets of $\mathcal{O}$ which entail the conclusion α, whereas minimal repairs are complements of the maximal subsets of $\mathcal{O}$ which do not entail α. Notice that each repair R for $\mathcal{O} \models \alpha$ should contain one axiom from every justification J for $\mathcal{O} \models \alpha$. Indeed, if $J \cap R = \emptyset$ for some justification J and repair R, then $J \subseteq \mathcal{O} \setminus R$, which violates

the conditions $J \models \alpha$ and $J \subseteq \mathcal{O} \setminus R \not\models \alpha$ due to monotonicity of the entailment. This means that every justification must be a minimal hitting set of the set of all minimal repairs and, likewise, every minimal repair is a minimal hitting set of the set of all justifications. This property is known as the *hitting set duality* (between justifications and minimal repairs).

The HTS-algorithm can easily be extended to compute repairs in addition to justifications. Indeed, as mentioned above, if $v \in V$ is a leaf node, then $H(v)$ is a repair for $\mathcal{O} \models \alpha$ since $\mathcal{O} \setminus H(v) \not\models \alpha$ by Condition 4 of Definition 12.

Example 26. The leaf nodes of the HS-tree on the left-hand side of Fig. 6 correspond to the repairs:

$$R_1 = \{A \sqsubseteq B,\ A \sqsubseteq C\},$$
$$R_2 = \{B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqsubseteq B\},$$
$$R_3 = \{B \sqsubseteq C,\ A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\}.$$

Notice that the repair R_2 is not minimal since $R_1 \subsetneq R_2$.

A natural question is whether *all* repairs for the entailment will be computed by the described extension of the HST-algorithm. This is not true for arbitrary repairs: indeed, the whole ontology $\mathcal{O}$ from Example 22 is clearly a repair for the entailment $\mathcal{O} \models \alpha = A \sqsubseteq C$, since $\mathcal{O} \setminus \mathcal{O} = \emptyset \not\models \alpha$ because $A \sqsubseteq C$ is not a tautology. However, as shown in Example 26, the repair $\mathcal{O}$ was not computed. It turns out, however, that the extended HST-algorithm computes all *minimal* repairs.

Exercise 19. Prove an analogy of Lemma 13 showing that if R is a minimal repair for $\mathcal{O} \models \alpha$ then each HS-tree $T = (V, E, L)$ for $\mathcal{O} \models \alpha$ contains a leaf node $v \in V$ such that $H(v) = R$. Hint: take a node $v \in V$ with a *maximal* $H(v)$ such that $H(v) \subseteq R$ and prove that $H(v) = R$ for this node. Use the property that $J \cap R \neq \emptyset$ for every justification J for $\mathcal{O} \models \alpha$.

To compute repairs using Algorithm 4, it is sufficient to add an else-block for the test in Line 6, in which the set H for which this test fails, is added to the set of repairs.

4.4 Computing Justifications and Repairs Using SAT Solvers

In the previous section, we have discussed the *hitting set duality* property between justifications and repairs: every justification is a hitting set for the set of all minimal repairs and every minimal repair is a hitting set of the set of all justifications. Hitting set duality takes a prominent place in the HST-algorithm, but we can use this property as the basis of a *direct* algorithm for computing justifications and minimal repairs.

Suppose that we have already computed some set S of justifications and some set P of minimal repairs for the entailment $\mathcal{O} \models \alpha$. How can we find a new justification or a minimal repair? As mentioned, each new justification

Algorithm 5. Maximizing non-entailment

Maximize$(\mathcal{O}, M, \alpha)$**:** compute a maximal subset $N \subseteq \mathcal{O}$ such that $M \subseteq N$ and $N \not\models \alpha$

input : an ontology $\mathcal{O}$, a subset $M \subseteq \mathcal{O}$, and an axiom α such that $M \not\models \alpha$

output : $N \subseteq \mathcal{O}$ such that $M \subseteq N \not\models \alpha$ but $N' \models \alpha$ for every N' with $N \subsetneq N' \subseteq \mathcal{O}$

```
1 N ← M;
2 for β ∈ O \ M do
3     if N ∪ {β} ⊭ α then
4         N ← N ∪ {β};
5 return N;
```

must be a hitting set for P, i.e., it should contain one axiom from every repair $R \in P$. Furthermore, it should be different from any of the previously computed justifications, i.e., it should miss one axiom from every $J \in S$. Suppose we have found a subset $M \subseteq \mathcal{O}$ satisfying these two requirements:

$$\forall R \in P : M \cap R \neq \emptyset, \tag{7}$$

$$\forall J \in S : J \setminus M \neq \emptyset. \tag{8}$$

If $M \models \alpha$, then, using Algorithm 3, we can extract a *minimal* subset $J' \subseteq M$ such that $J' \models \alpha$. Note that J' still misses at least one axiom from each $J \in S$ since (8) is preserved under removal of axioms from M. Therefore, J' is a *new justification* for $\mathcal{O} \models \alpha$. If $M \not\models \alpha$, then, similarly, by adding axioms $\beta \in \mathcal{O}$ to M preserving $M \not\models \alpha$, we can find a *maximal* superset N of M ($M \subseteq N \subseteq \mathcal{O}$) such that $N \not\models \alpha$: see Algorithm 5. Note that (7) is preserved under additions of elements to M, hence, $R' = \mathcal{O} \setminus N$ is a *new minimal repair* for $\mathcal{O} \models \alpha$. Thus, using any set M satisfying (7) and (8) we can find either a new justification or a new minimal repair.

How to find a set M satisfying Conditions (7) and (8)? These conditions require solving a rather complex combinatorial problem. Propositional (SAT) solvers, offer a convenient and effective way of solving such problems. In the following, we describe a propositional encoding of Conditions (7) and (8). The interested reader can find some background information on Propositional Logic and SAT in Appendix A.2.

To formulate the propositional encoding, we assign to each axiom $\beta \in \mathcal{O}$ a fresh propositional variable p_β. Then, every interpretation $\mathcal{I}$ determines a set $M = M(\mathcal{I}) = \{\beta \in \mathcal{O} \mid p_\beta^{\mathcal{I}} = 1\}$ of axioms whose corresponding propositional variable is true. We construct a propositional formula F such that $F^{\mathcal{I}} = 1$ if and only if $M(\mathcal{I})$ satisfies (7) and (8) for the given sets S of justifications and P of minimal repairs. Thus, to find a subset M satisfying (7) and (8), it is sufficient to find a model $\mathcal{I}$ of F and compute $M(\mathcal{I})$. We define F as follows:

$$F = F(S, P) = \bigwedge_{J \in S} \bigvee_{\beta \in J} \neg p_\beta \wedge \bigwedge_{R \in P} \bigvee_{\beta \in R} p_\beta. \tag{9}$$

Example 27. Let $\mathcal{O}$ be the ontology from Example 22. We assign propositional variables to axioms from $\mathcal{O}$ as follows:

- $A \sqsubseteq B \rightsquigarrow p_1$,
- $B \sqsubseteq C \rightsquigarrow p_2$,
- $A \sqsubseteq C \rightsquigarrow p_3$,
- $A \sqcap B \sqsubseteq \bot \rightsquigarrow p_4$.

Let S be the set of justifications J_1 and J_2 from Example 22 and P a set containing only repair R_1 from Example 26. Then according to (9) we have:

$$F = F(S, P) = (\neg p_1 \vee \neg p_2) \wedge (\neg p_3) \wedge (p_1 \vee p_3).$$

F has a model $\mathcal{I}$ with $p_1^\mathcal{I} = 1$ and $p_2^\mathcal{I} = p_3^\mathcal{I} = p_4^\mathcal{I} = 0$, which gives $M(\mathcal{I}) = \{A \sqsubseteq B\}$.

Once the set M determined by a model $\mathcal{I}$ of F is found, we can extract either a new justification J or a new repair R from M by minimizing entailment using Algorithm 3 or maximizing non-entailment using Algorithm 5. After that, we can update F according to (9) and compute a new model of F, if there exist any.

Example 28. Continuing Example 27, observe that $M(\mathcal{I}) = \{A \sqsubseteq B\} \not\models \alpha = A \sqsubseteq C$. By running Algorithm 5 for $\mathcal{O}$, $M = M(\mathcal{I})$ and α we compute N as follows:

N	β	$M \cup \{\beta\} \models^? \alpha = A \sqsubseteq C$
$\{A \sqsubseteq B\}$	$B \sqsubseteq C$	*yes*
$\{A \sqsubseteq B\}$	$A \sqsubseteq C$	*yes*
$\{A \sqsubseteq B\}$	$A \sqcap B \sqsubseteq \bot$	*yes*
$\{A \sqsubseteq B\}$	-	-

Hence, $R = \mathcal{O} \setminus N = \{B \sqsubseteq C, A \sqsubseteq C, A \sqcap B \sqsubseteq \bot\}$ is a new minimal repair for $\mathcal{O} \models \alpha$ (repair R_3 from Example 26). After we add this repair to P and recompute F according to (9), we obtain a formula with an additional conjunct:

$$F = F(S, P) = (\neg p_1 \vee \neg p_2) \wedge (\neg p_3) \wedge (p_1 \vee p_3) \wedge \underline{(p_2 \vee p_3 \vee p_4)}.$$

The interpretation $\mathcal{I}$ from Example 27 is no longer a model for F, but we can find a new model $\mathcal{I}$ of F with $p_1^\mathcal{I} = p_4^\mathcal{I} = 1$ and $p_2^\mathcal{I} = p_3^\mathcal{I} = 0$. For this model, we have $M(\mathcal{I}) = \{A \sqsubseteq B, A \sqcap B \sqsubseteq \bot\} \models \alpha$. By running Algorithm 3 for M and α, we obtain a new justification $J = \{A \sqsubseteq B, A \sqcap B \sqsubseteq \bot\}$ (justification J_3 from Example 22). The new justification, when added to S, gives us another conjunct for F:

Algorithm 6. Computing all justifications using a SAT solver

ComputeJustificationsSAT($\mathcal{O}, \alpha$): compute all justifications for $\mathcal{O} \models \alpha$

input : ontology $\mathcal{O}$ and axiom α such that $\mathcal{O} \models \alpha$

output : the set of all minimal subsets $J \subseteq \mathcal{O}$ such that $J \models \alpha$

```
1  S ← ∅;
2  F ← ⊤;
3  while ∃I : F^I = 1 do
4  |  I ← choose I : F^I = 1;
5  |  M ← {β | p_β^I = 1};
6  |  if M ⊨ α then
7  |  |  J ← Minimize(M, α);
8  |  |  S ← S ∪ {J};
9  |  |  F ← F ∧ ⋁{¬p_β | β ∈ J};
10 |  else
11 |  |  N ← Maximize(O, M, α);
12 |  |  F ← F ∧ ⋁{p_β | β ∈ O \ N};
13 return S;
```

$$F = F(S, P) = (\neg p_1 \vee \neg p_2) \wedge (\neg p_3) \wedge \underline{(\neg p_1 \vee \neg p_4)} \wedge (p_1 \vee p_3) \wedge (p_2 \vee p_3 \vee p_4).$$

This formula F is now unsatisfiable.

Note that if F is unsatisfiable, then S already contains all justifications for $\mathcal{O} \models \alpha$ and P contains all minimal repairs. Indeed, if S does not contain some justification J for $\mathcal{O} \models \alpha$ then $M = J$ clearly satisfies (7) and (8), hence, the interpretation $\mathcal{I} = \mathcal{I}(M)$ defined by $p_\alpha^{\mathcal{I}} = 1$ if and only if $\alpha \in M$, is a model of F. Similarly, if P does not contain some minimal repair R for $\mathcal{O} \models \alpha$, then $M = \mathcal{O} \setminus R$ satisfies (7) and (8), hence, the interpretation $\mathcal{I} = \mathcal{I}(M)$ is likewise a model of F. To conclude, either F is satisfiable and from its model we can compute a new justification or a minimal repair and extend F with the corresponding conjunct or F is unsatisfiable, in which case we have computed all justifications and minimal repairs.

Algorithm 6 summarizes the described procedure for computing all justifications using a SAT solver. We start by creating an empty set S of justifications (Line 1) and a formula F that is always true (Line 2). Then, in a loop (Lines 3–12), as long as F is satisfiable (which is checked using a SAT solver), we take any model $\mathcal{I}$ of F (Line 4), extract the corresponding set $M = M(\mathcal{I})$ that it defines (Line 5), and check the entailment $M \models \alpha$. If the entailment holds, using Algorithm 3 we compute a justification for $M \models \alpha$ (Line 7), which, by monotonicity of entailment, is also a justification for $\mathcal{O} \models \alpha$. This justification is then added to S (Line 8) and F is extended with a new conjunct for this justification according to (9) (Line 9). If the entailment does not hold, we compute a maximal

superset N of M such that $N \not\models \alpha$ using Algorithm 5 (Line 11) and extend F with the corresponding conjunct for the new repair $R = \mathcal{O} \setminus N$ according to (9) (Line 12). As soon as F becomes unsatisfiable, we return the set S of computed justifications (Line 13).

Example 29. Consider the entailment $\mathcal{O} \models \alpha$ from Example 22 and propositional encoding of axioms in $\mathcal{O}$ from Example 27. The following table shows a run of Algorithm 6 for the inputs $\mathcal{O}$ and α. Every row in this table corresponds to one iteration of the while-loop (Lines 3–12). The first column gives the value of the interpretation $\mathcal{I}$ for F computed in this iteration. The second column shows the value of M computed for this interpretation and whether the entailment $M \models \alpha$ holds. The third column shows the result of minimizing the entailment or maximizing the non-entailment using Algorithms 3 and 5. The last column shows the conjunct that is added to F for the corresponding justification or repair.

$p_1^{\mathcal{I}}$	$p_2^{\mathcal{I}}$	$p_3^{\mathcal{I}}$	$p_4^{\mathcal{I}}$	$M \models^? \alpha$	$\min(M) \models \alpha / \max(M) \not\models \alpha$	C
0	0	0	0	$\emptyset \not\models \alpha$	$\{A \sqsubseteq B\} \not\models \alpha$	$p_2 \vee p_3 \vee p_4$
0	1	0	0	$\{B \sqsubseteq C\} \not\models \alpha$	$\{B \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\} \not\models \alpha$	$p_1 \vee p_3$
1	1	0	0	$\{A \sqsubseteq B,\ B \sqsubseteq C\} \models \alpha$	$\{A \sqsubseteq B,\ B \sqsubseteq C\} \models \alpha$	$\neg p_1 \vee \neg p_2$
0	0	1	1	$\{A \sqsubseteq C,\ A \sqcap B \sqsubseteq \bot\} \models \alpha$	$\{A \sqsubseteq C\} \models \alpha$	$\neg p_3$
1	0	0	1	$\{A \sqsubseteq B,\ A \sqcap B \sqsubseteq \bot\} \models \alpha$	$\{A \sqsubseteq B,\ A \sqcap B \sqsubseteq \bot\} \models \alpha$	$\neg p_1 \vee \neg p_4$

Algorithm 6 can be easily turned into an algorithm for computing repairs (in addition or instead of justifications), by saving the repairs $\mathcal{O} \setminus N$ for N computed in Line 11.

Let us briefly discuss similarities and differences between Algorithm 4 and Algorithm 6. Both algorithms work by systematically exploring subsets of $\mathcal{O}$ and minimizing entailments from such subset to compute justifications. Algorithm 4 constructs such subsets ($\mathcal{O} \setminus H$) manually by removing one axiom appearing in the previously computed justification (if there is any) in all possible ways. Algorithm 6 enumerates such subsets M with a help of a SAT solver. The main difference is that Algorithm 4 may encounter the same subsets many times (on different branches), whereas the propositional encoding used in Algorithm 6 ensures that such subsets never repeat. The following example shows a situation where Algorithm 4 performs exponentially many iterations of the while-loop, whereas Algorithm 6 has only quadratically many iterations.

Example 30. Consider axioms $\beta_i = A \sqsubseteq B \sqcap D_i$, $\gamma_i = B \sqsubseteq C \sqcap D_i$ $(1 \leq i \leq n)$, and the ontology $\mathcal{O} = \{\beta_i, \gamma_i \mid 1 \leq i \leq n\}$, where A, B, C, and D_i $(1 \leq i \leq n)$ are atomic concepts. Clearly $\mathcal{O} \models \alpha = A \sqsubseteq C$. Furthermore, there are exactly n^2 justifications for $\mathcal{O} \models \alpha$: $J_{ij} = \{\beta_i, \gamma_j\}$ $(1 \leq i, j \leq n)$ and exactly 2 minimal repairs: $R_1 = \{\beta_i \mid 1 \leq i \leq n\}$, $R_2 = \{\gamma_j \mid 1 \leq j \leq n\}$. Hence Algorithm 6 will perform exactly $n^2 + 2 + 1$ calls to a SAT solver with a formula F of the size at

most $c \cdot (n^2 \cdot 2 + 2 \cdot n)$ for some constant c.[5] On the other hand, each HS-tree $T = (V, E, L)$ for $\mathcal{O} \models \alpha$ has at least 2^n nodes. Indeed, every non-leaf node $v \in V$ must be labeled by some $L(v) = J_{ij}$ with $1 \leq i, j \leq n$, which contains two axioms. Hence every non-leaf node of $v \in V$ must have two successor nodes (see Condition 1 of Definition 12). For every leaf node $v \in V$, the value $H(v)$ must be a repair for $\mathcal{O} \models \alpha$, so $H(v)$ must be a super-set of either R_1 or R_2. Hence $H(v)$ contains at least n elements, which means that the path from v to the root of T has at least n edges. Therefore, T is a binary tree whose leafs have the level n or higher. Hence T has at least 2^n nodes.

Of course, an iteration of Algorithm 4 cannot be directly compared to an iteration of Algorithm 6. Both iterations use at most one call to Algorithm 3, but Algorithm 6 may also require a call to Algorithm 5, as well as checking satisfiability of F. The latter requires solving an NP-compete problem, for which no polynomial algorithm is known so far. In order to check satisfiability of F, a SAT solver usually tries several (in worst-case exponentially many) propositional interpretations until a model of F is found. As each such interpretation $\mathcal{I}$ corresponds to a subset $M(\mathcal{I}) \subseteq \mathcal{O}$, this process can be compared to the enumeration of subsets in Algorithm 4. However, a SAT solver usually implements a number of sophisticated optimizations, which make the search for models very efficient in practice, whereas the subset enumeration strategy used Algorithm 4 is rather simplistic. Hence Algorithm 6 is likely to win in speed. On the other hand, Algorithm 6 requires saving all justifications (and minimal repairs) in the propositional formula F, which might result in a formula of exponential size, if the number of such justifications or repairs is exponential. In this regard, Algorithm 4 could be more memory efficient since saving (all) justifications is optional (see the discussion at the end of Sect. 4.2). Hence both algorithms have their own advantages and disadvantages.

5 Summary and Outlook

In this course, we have looked in-depth into the most common algorithms for reasoning and explanation in Description Logics. We have seen that the development of such algorithms is a complicated process already for the relatively simple DL $\mathcal{ALC}$. To show correctness of algorithms, one usually needs to prove several theoretical properties, such as soundness, completeness and termination. The algorithmic complexity analysis is helpful to understand the worst-case behavior of algorithms and to compare different algorithms across several dimensions such as (non-deterministic) time and space complexity. Identifying the exact computational complexity for various DLs and reasoning problems has, therefore, been one of the central research topics in DLs. The DL Complexity Navigator[6] provides an interactive overview of many of these results.

[5] The conjuncts for J_{ij} in F consist of two negated propositional variables, the conjunct for R_1 and R_2 in F consist of n propositional variables.

[6] http://www.cs.man.ac.uk/~ezolin/dl/.

Proving correctness and complexity results often requires understanding of model-theoretic properties of the languages. As we have seen in Sect. 3, for reasoning with $\mathcal{ALC}$ ontologies, it is sufficient to restrict the search to a special kind of *tree model* represented by tableaux. This so-called *tree model property* was argued to be one of the main reasons for decidability and the relatively low complexity of *Modal Logics*, the siblings of Description Logics [66]. For pure $\mathcal{ALC}$ concept satisfiability, i.e., without background ontologies, it is sufficient to consider tree models of a bounded depth (Sect. 3.1). With additional background ontologies, the tree models are no longer finite and special *blocking techniques* are required to ensure termination of tableau algorithms (Sect. 3.2). When moving to very expressive DLs, such as $\mathcal{SROIQ}$ [29] (the language underpinning the OWL 2 Direct Semantics), eventually the tree model property is lost and proving termination of tableau procedures, while still ensuring soundness and completeness, becomes increasingly difficult. It is not very surprising that when increasing the *expressivity* of languages, i.e., when adding new ways to construct concepts and axioms, the complexity of the reasoning problems increases as well. For example, the time complexity of all standard reasoning problems in $\mathcal{SROIQ}$ becomes *non-deterministic doubly exponential* [31], whereas it is "only" *deterministic exponential* for $\mathcal{ALC}$ (see the remark after Theorem 2).

The theoretical analysis of algorithms does not always give an accurate prediction about their practical performance. Often a situation that triggers the worst-case behavior of an algorithm represents some corner case, which rarely appears in practice. When it comes to practical efficiency, some other properties of algorithms become more important. For example, despite a relatively high algorithmic complexity (see Theorems 1 and 2), tableau algorithms remain among the fastest DL reasoning algorithms to date. This phenomenon can be explained by a range of optimization techniques that have been developed for tableau algorithms in the past two decades.

All state-of-the-art tableau reasoners, e.g., FaCT++ [62], HermiT [42], Konclude [58], MoRE [48], and Pellet [56], apply a significant range of optimizations. The optimizations can be categorized into those for *preprocessing*, *consistency checking*, and for *higher level reasoning tasks*. Examples of higher level reasoning tasks are *classification*, where one computes all subsumption relationships between atomic concepts or *materialization*, where one extends the ontology, for each individual (pair of individuals), with assertions to capture the atomic concepts (roles) of which the individual (the pair of individuals) is an instance.

Most reasoning systems preprocess the input ontologies. The simplest form of preprocessing is the presented conversion into negation normal form (Definition 2 in Sect. 3), which is not used for improving performance, but rather to allow for using fewer tableau rules. Other standard preprocessing optimizations include *lexical normalization* and *simplification*, which aim at identifying syntactic equivalences, contradictions and tautologies [4, Sect. 9.5]. A well-known and very important optimization for improving performance is *absorption*, which aims at rewriting general concept inclusion axioms to avoid non-determinism in the tableau algorithm. For example, here we suggested to convert an axiom of

the form $A \sqcap B \sqsubseteq C$ into $\top \sqsubseteq \neg A \sqcup \neg B \sqcup C$ to allow for handling them with the $\top$-Rule. This introduces, however, a non-deterministic decision for each axiom and each node. Instead, practical tableau systems use a variant of the $\sqsubseteq$-Rule introduced in Table 4 restricted to atomic concepts on the left-hand side, i.e., for an axiom of the form $A \sqsubseteq C$ in the ontology, a node with A in its label, but C not in its label, the node's label is extended with C. With this rule, one can transform $A \sqcap B \sqsubseteq C$ into $A \sqsubseteq \neg B \sqcup C$, which already reduces the amount of non-determinism. Binary absorption [30] further allows for a conjunction of (two) atomic concepts on the left-hand side of a general concept inclusion, i.e., one can completely avoid the non-deterministic decisions for our example axiom $A \sqcap B \sqsubseteq C$. Further absorption techniques include *role absorption* [61], *nominal absorption* [55], and partial absorption [57].

As outlined in Sect. 3, consistency checking is the core reasoning task of a tableau-based reasoner. Since these checks typically occur very often, many optimizations are known including *model merging* techniques [24], *lazy unfolding*, *semantic branching*, *boolean constraint propagation*, *dependency directed backtracking* and *backjumping*, and *caching*. We refer interested readers to the DL Handbook [4, Sect. 9] for a more detailed descriptions of the latter optimizations. The HermiT reasoner further tries to reduce non-determinism by combining hypertableau [8] and hyper-resolution [47] techniques. In order to reduce the size of the tableau, modern DL reasoners several blocking strategies such as *anywhere blocking* [42] or *core blocking* [19].

Higher level reasoning tasks are usually reduced to a multitude of consistency checks such that they benefit from the optimizations of this task as much as possible. Many OWL reasoners, solve the classification problem using an Enhanced Traversal (ET) classification algorithm [5] similar to the one used in early description logic reasoners. To construct a concept hierarchy, the algorithm starts with the empty hierarchy and then iteratively inserts each concept from the ontology into the hierarchy. Each insertion step typically requires one or more subsumption tests—checks whether a subsumption relationship holds between two concepts—in order to determine the proper position of a class in the hierarchy constructed thus far. A more recent alternative to the ET algorithm is the *known/possible set classification* approach [20].

Despite the wide range of implemented optimization techniques, the reasoning performance might not be sufficient for some applications. The OWL 2 standard addresses this by introducing so-called OWL profiles [41], which are fragments of OWL 2 that restrict the allowed constructors in order to allow for tractable reasoning procedures. For example, the OWL 2 EL profile (based on the Description Logic $\mathcal{EL}$, a fragment of $\mathcal{ALC}$) allows for a one-pass classification of ontologies, i.e., repetitive subsumption tests are not needed. Some reasoners, e.g., Konclude, combine tableau procedures with tractable algorithms for handling those parts of an ontologies that are in the OWL 2 EL profile. Similarly, MoRe combines the (hyper-)tableau reasoner HermiT with the specialized OWL 2 EL reasoner ELK [34].

Many optimizations try to avoid unnecessary operations by making algorithms more *goal-directed* and thus reducing the *search space*. We have seen several examples of such optimizations in Sect. 4 when considering algorithms for computing justifications and repairs. Such optimizations typically do not reduce the worst case complexity of algorithms but they can significantly improve their behavior in typical cases. For example, Algorithm 3 for computing one justification, in practice, does not start with the whole ontology $J = \mathcal{O}$ (Line 3), but with a subset $J \subseteq \mathcal{O}$ such that $J \models \alpha$. If a small subset J like this is found, the number of subsequent entailment tests performed by the algorithm can significantly be reduced. The initial subset J can be found, for example, by starting with $J = \emptyset$ and repeatedly adding to J axioms from $\mathcal{O}$ until $J \models \alpha$. This part of the algorithm, called the *expansion phase*, requires additional entailment tests. To find a J that is as small as possible, one usually tries to first add axioms that are most *likely* to cause the entailment $J \models \alpha$, e.g., the axioms $\beta \in \mathcal{O}$ that contain symbols from α or from the previously added axioms in J. The initial subset $J \subseteq \mathcal{O}$ such that $\mathcal{J} \models \alpha$ can also be found using algorithms for computing modules of ontologies. A *(logical) module of $\mathcal{O}$ for a set of symbols Σ* is a subset $M \subseteq \mathcal{O}$ such that for every axiom β formulated using only symbols in Σ, if $\mathcal{O} \models \beta$ then $M \models \beta$. In our case we are interested in Σ consisting of all symbols in α. Some types of modules, e.g., *locality-based modules* can be computed in polynomial time without performing any subsumption tests [16]. It is also possible to reduce the number of entailment tests when minimizing the entailment $\mathcal{J} \models \alpha$ by removing several axioms at a time instead of one axiom like in Algorithm 3. Further details of optimization techniques for computing justifications can be found in the PhD thesis of Horridge [26].

Another way to optimize an algorithm is to use an existing "off-the-shelf" tool that is already optimized for solving a certain class of problems. One of the most popular examples of such tools are SAT solvers. In Sect. 4.4 we have shown how SAT solvers can be used for computing justifications and repairs (see Algorithm 6). This algorithm or variations thereof are implemented in several tools such as EL+SAT [53,67], EL2MUS [1], and SATPin [39]. The SAT solvers used in these tools are not only only used to find new candidate subsets M for justifications or complements of repairs, but also to check the entailments $\mathcal{O} \models \alpha$. This has been possible by using different, *consequence-based* algorithms for reasoning with ontologies. In contrast to tableau algorithms, consequence-based algorithms do not construct (representations of) models, but instead derive logical consequence of axioms using a number of dedicated inference rules. Thus, to prove the entailment $\mathcal{O} \models \alpha$, it is sufficient to show how the axiom α can be *derived* using these rules and the axioms in $\mathcal{O}$. Each *inference step* $\alpha_1, \ldots, \alpha_n \vdash \alpha$ used in this derivation can be encoded as a propositional formula $p_{\alpha_1} \wedge \cdots \wedge p_{\alpha_n} \rightarrow p_\alpha$, thus reducing DL entailment to propositional (Horn) entailment. Consequence-based procedures have been first formulated for the simple DL $\mathcal{EL}$ to show that entailment in this language can be solved in polynomial time [13]. The above mentioned tools for computing justifications are targeted to this

language. Since then, consequence-based procedures have also been extended to more expressive (even non-polynomial) DLs [3,7,15,32,54].

One of the benefits of consequence-based algorithms is that they can be used to provide better explanations for the obtained reasoning results. Justification for entailments $\mathcal{O} \models \alpha$ tell *which* axioms of the ontology are responsible for the entailment, but not *how* the entailed axiom was obtained from them. This limitation has been mainly due to the *black-box* nature of the tableau-based reasoning algorithms: since tableau algorithms are based on constructing models, they cannot provide information supporting *positive* entailment tests $\mathcal{O} \models \alpha$ since in such cases no counter-model for the entailment $\mathcal{O} \models \alpha$ exists (see Lemma 2). In contrast, consequence-based algorithms can provide explanations for entailment in the form of derivations (or proofs). In practice, computing derivations for a given subsumption has not been an easy task because if in addition to computing all consequences, we also save all inference steps by which they were produced, the amount of memory required to store all this information can double. A *goal-directed* procedure for generation of inferences [33] can be used to mitigate this problem. The (black-box) algorithms for computing justifications have also been extended to provide some inference steps that derive (simple) *intermediate conclusions*, which can improve understanding of explanations [27,28].

Ontologies and the reasoning techniques described in this course are successfully employed in many domains, e.g., to reason over the environment of (autonomous) cars [17,68], in information integration tasks [35,38], or, most prominently, in medicine, life sciences, and bio-informatics [21,25,63]. The standardization efforts of the World Wide Web Consortium (W3C) for the DL-based Web Ontology Language OWL have certainly helped in promoting the use of logic-based knowledge representation and reasoning. While modern search engines have picked up the ideas of using structured or formal knowledge, this is often not in the form OWL (or DL) ontologies. For example, Google's knowledge graph and Facebook's Social Graph are based on proprietary formats. The same holds for Wikidata, although Semantic Web standards are also supported (e.g., a SPARQL [23] query interface and data dumps in the Resource Description Format (RDF) [52] are available). We attribute this to several reasons: While large companies such as Google recognized the importance of structured knowledge, they rather use their proprietary formats, possibly for business reasons. A contributing challenge is also that even the tractable fragments of DLs do not offer the performance required at Web scale. Furthermore, the knowledge in the Web is inherently inconsistent, which is challenging for logic-based approaches. DLs and OWL also lack features that are important for some applications. For example, Wikidata captures when a fact was true, e.g., the former German chancellor Helmut Kohl was married to Hannelore Kohl from 1960 to 2001. This is difficult to model using DLs since roles can only relate two elements, but research to address these issues is on-going [37,40]. Summing up, it is widely accepted today that structuring and formalizing knowledge is important and that significant advances were made in the last years; nevertheless, research is still needed in several directions.

A Appendix

For the convenience of interested readers, in this appendix we recap some background material used in this course, such as the basic notions for describing the (theoretical) complexity of algorithms, and the propositional satisfiability problem.

A.1 Computational Complexity

A *decision problem* (for an input set X) is simply a mapping $P\colon X \to \{yes, no\}$. Note that X can be an arbitrary set of objects. For example, for the concept subsumption problem, X consists of all possible *pairs* $\langle \mathcal{O}, C \sqsubseteq D\rangle$ where the first component is an ontology $\mathcal{O}$ and the second component is a concept subsumption $C \sqsubseteq D$. An algorithm A *solves* (or *decides*) a decision problem P for X, if A accepts each value $x \in X$ as input, terminates for all these values, and returns the (correct) result $A(x) = P(x)$.

There are several dimensions according to which one can measure the *computational complexity* of problems and algorithms. We say that an algorithm A has an *(upper) time complexity* $f(n)$ if for each input $x \in X$ with the size (e.g., the number of symbols) n, the algorithm A terminates after at most $f(n)$ steps. A problem P for X is *solvable in time* $f(n)$ if there exists an algorithm A that solves P and has the time complexity $f(n)$. We say that a problem P is solvable in *polynomial time* if there exists a polynomial function $f(n)$ such that P is solvable in time $f(n)$. A problem P is solvable in *exponential time* (*doubly exponential time*, ...) if there exists a polynomial function $f(n)$ such that P is solvable in time $2^{f(n)}$ ($2^{2^{f(n)}}$, ...). Analogously to the algorithmic time complexity, one can define the algorithmic *space complexity*: a problem P for X is solvable in space $f(n)$ if there exists an algorithm A that solves P such that for each input $x \in X$ with the size n, the algorithm A uses at most $f(n)$ units of memory at every step of the computation.

Another dimension of the computational complexity is based on the notion of a non-deterministic computation. An algorithm A is said to be *non-deterministic* if the result of some operations that it can perform is not uniquely determined. Thus, the algorithm can produce different results for different *runs* even with the same input. A non-deterministic algorithm A *solves* a problem P for X if, for each $x \in X$ such that $P(x) = no$, each run of A terminates with the result *no*, and for each $x \in X$ such that $P(x) = yes$, there exists *at least one run* for which the algorithm terminates and produces *yes*. The intuition is that, if one has an unlimited number of identical computers, then one can solve the problem P by starting the algorithm A *in parallel* on all of these computers; if $P(x) = yes$, one of them is guaranteed to return *yes* (provided the results of all non-deterministic instructions are chosen at random).

The time and space complexity measures are also extended to non-deterministic algorithms. For example, a non-deterministic algorithm A has the *(upper) time complexity* $f(n)$ if, for every input $x \in X$ of the size n, *every run* of A terminates after at most $f(n)$ steps. We say that a problem P for X is *solvable*

in non-deterministic time $f(n)$ if there exists a non-deterministic algorithm A that solves P and has the time complexity $f(n)$. Thus, a problem P is solvable in *non-deterministic polynomial (exponential, doubly exponential, ...) time* if P is solvable in non-deterministic time $f(n)$, where $f(n)$ is a polynomial (exponential, doubly exponential) function. The non-deterministic space complexity is defined similarly.

A common way to solve a problem is to reduce it to another problem, for which a solution is known. A decision problem $P_1 \colon X \to \{yes, no\}$ *is (many-one) reducible* to a decision problem $P_2 \colon Y \to \{yes, no\}$ if there exists an algorithm $R \colon X \to Y$ (that takes an input from X and produces an output from Y) such that for every $x \in X$, we have $P_1(x) = P_2(R(x))$. In this case the algorithm R is called *a reduction* from P_1 to P_2. Depending on the time or space complexity of the algorithm R (i.e., the maximal number of steps or memory units consumed for inputs of size n), the complexity bounds of the problems are also transferred by the reduction. Usually one is interested in *polynomial reductions*, where the number of steps for computing each $R(x)$ is bounded by a polynomial function in the size of x. In this case, if the complexity of P_2 is polynomial, exponential, or doubly exponential (for deterministic or non-deterministic, time or space complexity), then P_1 has the same complexity as P_2.

A.2 Propositional Logic and SAT

The *vocabulary* of Propositional Logic consists of a countably infinite set P of *propositional variables*, *Boolean constants*: $\top$ (Verum), $\bot$ (Falsum), and *Boolean operators*: $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation) and $\to$ (implication). *Propositional formulas* are constructed from these symbols according to the grammar:

$$F, G ::= p \mid \top \mid \bot \mid F \wedge G \mid F \vee G \mid \neg F \mid F \to G, \tag{10}$$

where $p \in P$. A *propositional interpretation* $\mathcal{I}$ assigns to each propositional variable $p \in P$ a *truth value* $p^{\mathcal{I}} \in \{1, 0\}$ (1 means '*true*', 0 means '*false*') and is extended to other propositional formulas by induction over the grammar definition (10) as follows:

- $\top^{\mathcal{I}} = 1$ and $\bot^{\mathcal{I}} = 0$ for each $\mathcal{I}$,
- $(F \wedge G)^{\mathcal{I}} = 1$ if and only if $F^{\mathcal{I}} = 1$ and $G^{\mathcal{I}} = 1$,
- $(F \vee G)^{\mathcal{I}} = 1$ if and only if $F^{\mathcal{I}} = 1$ or $G^{\mathcal{I}} = 1$,
- $(\neg F)^{\mathcal{I}} = 1$ if and only if $F^{\mathcal{I}} = 0$,
- $(F \to G)^{\mathcal{I}} = 1$ if and only if $F^{\mathcal{I}} = 0$ or $G^{\mathcal{I}} = 1$.

If $F^{\mathcal{I}} = 1$ then we say that $\mathcal{I}$ is a *model* of F (or F is *satisfied* in $\mathcal{I}$). We say that F *is satisfiable* if F has at least one model; otherwise F is *unsatisfiable*. A *propositional satisfiability problem* (short: *SAT*) is the following decision problem:

- Given: a propositional formula F,

– Return: *yes* if F is satisfiable and *no* otherwise.

SAT is a classical example of a *non-deterministic polynomial* (short: *NP*) problem: it can be solved using an algorithm that non-deterministically choses a propositional interpretation $\mathcal{I}$, computes (in polynomial time) the value $F^{\mathcal{I}}$ and returns *yes* if $F^{\mathcal{I}} = 1$ and *no* if $F^{\mathcal{I}} = 0$. It can be shown that each problem solvable by a non-deterministic polynomial algorithm has a polynomial reduction to SAT, which means that SAT is actually an *NP-complete* problem. Currently, the most efficient algorithms for solving SAT are based on (extensions of) the *Davis-Putnam-Logemann-Loveland* (short: *DPLL*) procedure, which systematically explores interpretations in a goal-directed way. A program that implements an algorithm for solving SAT is called a *SAT-solver*. Usually a SAT-solver not only decides satisfiability of a given propositional formula F, but can also output a *model* of F in case F is satisfiable.

References

1. Arif, M.F., Mencía, C., Marques-Silva, J.: Efficient MUS enumeration of horn formulae with applications to axiom pinpointing. CoRR abs/1505.04365 (2015)
2. Baader, F.: Description logics. In: Tessaris, S., et al. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 1–39. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03754-2_1
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 364–369 (2005)
4. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications, 2nd edn. Cambridge University Press, Cambridge (2007)
5. Baader, F., Franconi, E., Hollunder, B., Nebel, B., Profitlich, H.J.: An empirical analysis of optimization techniques for terminological representation systems. Appl. Intell. **4**(2), 109–132 (1994)
6. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
7. Bate, A., Motik, B., Grau, B.C., Cucala, D.T., Simancik, F., Horrocks, I.: Consequence-based reasoning for description logics with disjunctions and number restrictions. J. Artif. Intell. Res. **63**, 625–690 (2018)
8. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper tableaux. In: Alferes, J.J., Pereira, L.M., Orlowska, E. (eds.) JELIA 1996. LNCS, vol. 1126, pp. 1–17. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61630-6_1
9. Bienvenu, M., Bourgaux, C.: Inconsistency-tolerant querying of description logic knowledge bases. In: Pan, J.Z., et al. (eds.) Reasoning Web 2016. LNCS, vol. 9885, pp. 156–202. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49493-7_5
10. Bienvenu, M., Ortiz, M.: Ontology-mediated query answering with data-tractable description logics. In: Faber, W., Paschke, A. (eds.) Reasoning Web 2015. LNCS, vol. 9203, pp. 218–307. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21768-0_9
11. Bonatti, P.A., Faella, M., Petrova, I.M., Sauro, L.: A new semantics for overriding in description logics. Artif. Intell. **222**, 1–48 (2015)

12. Botoeva, E., Konev, B., Lutz, C., Ryzhikov, V., Wolter, F., Zakharyaschev, M.: Inseparability and conservative extensions of description logic ontologies: a survey. In: Pan, J.Z., et al. (eds.) Reasoning Web 2016. LNCS, vol. 9885, pp. 27–89. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49493-7_2
13. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and - what else? In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 298–302. IOS Press (2004)
14. Casini, G., Straccia, U.: Defeasible inheritance-based description logics. J. Artif. Intell. Res. **48**, 415–473 (2013)
15. Cucala, D.T., Grau, B.C., Horrocks, I.: Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. In: Lang, J. (ed.) Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018), pp. 1970–1976. ijcai.org (2018)
16. Cuenca Grau, B., Horrocks, I., Kazakov, Y., Sattler, U.: Modular reuse of ontologies: theory and practice. J. Artif. Intell. Res. **31**, 273–318 (2008)
17. Feld, M., Müller, C.: The automotive ontology: managing knowledge inside the vehicle and sharing it between cars. In: Proceedings of the 3rd International Conference on Automotive User Interfaces and Interactive Vehicular Applications, AutomotiveUI 2011, pp. 79–86. ACM, New York (2011). http://doi.acm.org/10.1145/2381416.2381429
18. Giordano, L., Gliozzi, V., Olivetti, N., Pozzato, G.L.: A non-monotonic description logic for reasoning about typicality. Artif. Intell. **195**, 165–202 (2013)
19. Glimm, B., Horrocks, I., Motik, B.: Optimized description logic reasoning via core blocking. In: Giesl, J., Hähnle, R. (eds.) IJCAR 2010. LNCS (LNAI), vol. 6173, pp. 457–471. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14203-1_39
20. Glimm, B., Horrocks, I., Motik, B., Shearer, R., Stoilos, G.: A novel approach to ontology classification. J. Web Semant. **14**, 84–101 (2012)
21. Golbreich, C., Zhang, S., Bodenreider, O.: The foundational model of anatomy in OWL: experience and perspectives. J. Web Semant. **4**(3), 181–195 (2006)
22. Greiner, R., Smith, B.A., Wilkerson, R.W.: A correction to the algorithm in Reiter's theory of diagnosis. In: Readings in Model-Based Diagnosis, pp. 49–53. Morgan Kaufmann Publishers Inc. (1992)
23. Group, T.W.W. (ed.): SPARQL 1.1 Overview. W3C Recommendation, 21 March 2013. http://www.w3.org/TR/sparql11-overview/
24. Haarslev, V., Möller, R., Turhan, A.-Y.: Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In: Goré, R., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS, vol. 2083, pp. 61–75. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45744-5_6
25. Hoehndorf, R., Dumontier, M., Gkoutos, G.V.: Evaluation of research in biomedical ontologies. Briefings Bioinform. **14**(6), 696–712 (2012)
26. Horridge, M.: Justification based explanation in ontologies. Ph.D. thesis, University of Manchester, UK (2011)
27. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: Sheth, A., et al. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 323–338. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88564-1_21
28. Horridge, M., Parsia, B., Sattler, U.: Justification oriented proofs in OWL. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 354–369. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17746-0_23

29. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Doherty, P., Mylopoulos, J., Welty, C.A. (eds.) Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 57–67. AAAI Press (2006)
30. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proceedings of the 19th International Workshop on Description Logics (DL 2006), vol. 189. CEUR (2006)
31. Kazakov, Y.: $\mathcal{RIQ}$ and $\mathcal{SROIQ}$ are harder than $\mathcal{SHOIQ}$. In: Brewka, G., Lang, J. (eds.) Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), pp. 274–284. AAAI Press (2008)
32. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 2040–2045. IJCAI (2009)
33. Kazakov, Y., Klinov, P.: Goal-directed tracing of inferences in EL ontologies. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8797, pp. 196–211. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11915-1_13
34. Kazakov, Y., Krötzsch, M., Simančík, F.: ELK: a reasoner for OWL EL ontologies. System description, University of Oxford (2012)
35. Kharlamov, E., et al.: Ontology based data access in statoil. Web Semant. Sci. Serv. Agents World Wide Web **44**, 3–36 (2017)
36. Kontchakov, R., Zakharyaschev, M.: An introduction to description logics and query rewriting. In: Koubarakis, M., et al. (eds.) Reasoning Web 2014. LNCS, vol. 8714, pp. 195–244. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10587-1_5
37. Krötzsch, M., Marx, M., Ozaki, A., Thost, V.: Attributed description logics: reasoning on knowledge graphs. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, 13–19 July 2018. pp. 5309–5313. ijcai.org (2018). https://doi.org/10.24963/ijcai.2018/743
38. Maier, A., Schnurr, H.-P., Sure, Y.: Ontology-based information integration in the automotive industry. In: Fensel, D., Sycara, K., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 897–912. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39718-2_57
39. Manthey, N., Peñaloza, R., Rudolph, S.: Efficient axiom pinpointing in $\mathcal{EL}$ using SAT technology. In: Lenzerini, M., Peñaloza, R. (eds.) Proceedings of the 29th International Workshop on Description Logics (DL 2016). CEUR Workshop Proceedings, vol. 1577. CEUR-WS.org (2016). http://ceur-ws.org/Vol-1577/paper_33.pdf
40. Motik, B.: Representing and querying validity time in RDF and OWL: a logic-based approach. J. Web Semant. **12**, 3–21 (2012). https://doi.org/10.1016/j.websem.2011.11.004
41. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Language: Profiles. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/owl2-profiles/
42. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. Artif. Intell. Res. **36**, 165–228 (2009)
43. Ortiz, M., Šimkus, M.: Reasoning and query answering in description logics. In: Eiter, T., Krennwallner, T. (eds.) Reasoning Web 2012. LNCS, vol. 7487, pp. 1–53. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33158-9_1
44. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/owl2-overview/

45. Peñaloza, R.: Explaining axiom pinpointing. In: Lutz, C., Sattler, U., Tinelli, C., Turhan, A.-Y., Wolter, F. (eds.) Description Logic, Theory Combination, and All That. LNCS, vol. 11560, pp. 475–496. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22102-7_22
46. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**(1), 57–95 (1987)
47. Robinson, J.A.: Automatic deduction with hyper-resolution. Int. J. Comput. Math. **1**(3), 227–234 (1965)
48. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: modular combination of OWL reasoners for ontology classification. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7649, pp. 1–16. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35176-1_1
49. Rudolph, S.: Foundations of description logics. In: Polleres, A., et al. (eds.) Reasoning Web 2011. LNCS, vol. 6848, pp. 76–136. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23032-5_2
50. Sattler, U.: Reasoning in description logics: basics, extensions, and relatives. In: Antoniou, G., et al. (eds.) Reasoning Web 2007. LNCS, vol. 4636, pp. 154–182. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74615-7_2
51. Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. J. Artif. Intell. **48**, 1–26 (1991)
52. Schreiber, G., Raimond, Y. (eds.): RDF 1.1 Primer. W3C Working Group Note, 24 June 2014. http://www.w3.org/TR/rdf11-primer/
53. Sebastiani, R., Vescovi, M.: Axiom pinpointing in lightweight description logics via horn-SAT encoding and conflict analysis. In: Schmidt, R.A. (ed.) CADE 2009. LNCS (LNAI), vol. 5663, pp. 84–99. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02959-2_6
54. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond horn ontologies. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011), pp. 1093–1098. AAAI Press/IJCAI (2011)
55. Sirin, E.: From wine to water: optimizing description logic reasoning for nominals. In: Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 90–99. AAAI Press (2006)
56. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: a practical OWL-DL reasoner. J. Web Semant. **5**(2), 51–53 (2007)
57. Steigmiller, A., Glimm, B., Liebig, T.: Optimised absorption for expressive description logics. In: Proceedings of the 27th International Workshop on Description Logics (DL 2014). CEUR Workshop Proceedings, vol. 1193. CEUR-WS.org (2014). https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2014/StGL14b.pdf
58. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. Web Semant. **27–28**, 78–85 (2014)
59. Straccia, U.: All about fuzzy description logics and applications. In: Faber, W., Paschke, A. (eds.) Reasoning Web 2015. LNCS, vol. 9203, pp. 1–31. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21768-0_1
60. Tobies, S.: Complexity results and practical algorithms for logics in knowledge representation. Ph.D. thesis, RWTH Aachen, Germany (2001)
61. Tsarkov, D., Horrocks, I.: Efficient reasoning with range and domain constraints. In: Proceedings of the 17th International Workshop on Description Logics (DL 2004), vol. 104. CEUR (2004)

62. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_26
63. Tudorache, T., Nyulas, C.I., Noy, N.F., Musen, M.A.: Using semantic web in ICD-11: three years down the road. In: Alani, H., et al. (eds.) ISWC 2013. LNCS, vol. 8219, pp. 195–211. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41338-4_13
64. Turhan, A.-Y.: Reasoning and explanation in $\mathcal{EL}$ and in expressive description logics. In: Aßmann, U., Bartho, A., Wende, C. (eds.) Reasoning Web 2010. LNCS, vol. 6325, pp. 1–27. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15543-7_1
65. Turhan, A.-Y.: Introductions to description logics – a guided tour. In: Rudolph, S., Gottlob, G., Horrocks, I., van Harmelen, F. (eds.) Reasoning Web 2013. LNCS, vol. 8067, pp. 150–161. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39784-4_3
66. Vardi, M.Y.: Why is modal logic so robustly decidable? In: Immerman, N., Kolaitis, P.G. (eds.) Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop 1996, Princeton, New Jersey, USA, 14–17 January 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 31, pp. 149–183. DIMACS/AMS (1996)
67. Vescovi, M.: Exploiting SAT and SMT techniques for automated reasoning and ontology manipulation in description logics. Ph.D. thesis, University of Trento, Italy (2011). http://eprints-phd.biblio.unitn.it/477/
68. Zhao, L., Ichise, R., Mita, S., Sasaki, Y.: Core ontologies for safe autonomous driving. In: Villata, S., Pan, J.Z., Dragoni, M. (eds.) Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC-2015), Bethlehem, PA, USA, 11 October 2015. CEUR Workshop Proceedings, vol. 1486. CEUR-WS.org (2015). http://ceur-ws.org/Vol-1486/paper_9.pdf