

Knowledge Representation and Automated Formal Reasoning in Description Logic ALC

Vasyl Lenko¹, Volodymyr Pasichnyk¹, Natalia Kunanets¹ and Yurii Shcherbyna²

¹ Lviv Polytechnic National University, 12 S. Bandery str., Lviv, 79013, Ukraine

² Ivan Franko National University of Lviv, 1 Universytetska str., Lviv, 79000, Ukraine

Abstract

The paper presents a systematic analysis of the description logic formalism that has proved to be the distinguished approach for automated formal reasoning. The focus is on the principles of knowledge representation and reasoning within the framework of basic description logic ALC. An application of the specified formalism towards the domain of mass media reveals the benefits of automated reasoning for decision making.

Keywords 1

Description logic, automated formal reasoning, tableaux algorithm, ALC

1. Introduction

Description logics (DLs) play a fundamental role in modern ontologies that are based on OWL 2. They form the basis of direct model-theoretic semantics [1] for OWL 2 ontologies, which are de facto the standard in knowledge bases engineering. The historical evolution of ontologies from a frame model towards logic-based knowledge representation models is explained by the issues of an ambiguous interpretation of syntactic constructions in frames and limited generality of the associated reasoning methods [2]. Description logics, as decidable fragments of first-order logic, provide formal model-theoretic semantics and sound methods (forms) of reasoning over syntactic expressions.

For OWL 2, the most appropriate semantics among description logics is *SROIQ*, while for OWL it is description logic *SHOIN*. Description logics are named using abbreviations of sets of constructors (axioms), which form the basis for their grammar. For instance, *SROIQ* consists of the following constructors: S – denotes the set of constructors of the basic description logic ALC, extended with the transitive roles; R – denotes an extended set of role axioms, also called RBox; O – denotes nominals; I – denotes the inverse roles; Q – denotes qualified cardinality restrictions. The variety of description logics is explained by the combinatorial possibilities of the constructors' set selection but obtaining the description logic with attractive expressiveness and computability is a difficult research task. The reason is a fundamental trade-off between expressiveness and decidability of formal language – higher expressiveness leads to a higher probability of undecidability of reasoning problems. Adoption of a new description logic requires formal proofs of its properties as a decision procedure, as well as careful selection of a set of constructors.

We will conduct a systematic analysis of 1) the structure of description logics, based on the example of the description logic ALC; 2) the main reasoning problems that can be stated and solved in ALC; 3) the concept of a knowledge base in the context of description logics; 4) fundamental properties of description logics; 5) model-theoretic semantics and tableaux-based method for the inference of logical conclusion.

MomLeT+DS 2021: 3rd International Workshop on Modern Machine Learning Technologies and Data Science, June 5, 2021, Lviv-Shatsk, Ukraine

EMAIL: vs.lenko@gmail.com (V. Lenko); vpasichnyk@gmail.com (V. Pasichnyk); nek.lviv@gmail.com (N. Kunanets); yshcherbyna@yahoo.com (Y. Shcherbyna)

ORCID: 0000-0002-3109-480X (V. Lenko); 0000-0002-5231-6395 (V. Pasichnyk); 0000-0003-3007-2462 (N. Kunanets); 0000-0002-4942-2787 (Y. Shcherbyna)



© 2021 Copyright for this paper by its authors.
Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

2. Description logics

Description logics are a family of logic-based formal languages for knowledge representation and reasoning that date back to the late 1980s. The notion “description logic” corresponds to the ontological commitments of this formalism: the main aspects of the domain of interest are presented in the form of a “description” of its entities (concepts) and relationships (roles), which are created by the constructors (axioms) of the selected description logic. For the sake of knowledge reuse, syntactic expressions are classified by their generality and divided into two parts – a set of terminological axioms, TBox, and a set of statements about individuals, ABox, which together form the knowledge base, KB, also known as an ontology.

Table 1

Correspondence between the elements of formal logic and OWL 2

Predicate logic	Description logic	OWL 2
Unary predicate	Concept	Class
Binary predicate	Role	Property
Constant	Individual	Instance

A fundamental property of description logics is the presence of formal semantics, which allows to interpret correctly syntactic expressions, axioms, and operators, thus creating an unambiguous, shared understanding of when the statement itself is a logical consequence from the knowledge base. Since description logics are decidable fragments of predicate logic, the logical inference is executed using automated reasoning methods, which are based on the principles of semantic admissibility and deductive derivability of a statement. The peculiarity of reasoning in descriptive logics is that computation is always carried out over the entire knowledge base. The conceptual trade-off between the expressiveness of the representation language and the computational complexity of reasoning problems, as well as the number of statements in the knowledge base, give rise to the need for a variety of description logics. In particular, the family of description logics *EL* is characterized by limited expressiveness of the representation language but guarantees a polynomial time solution for some reasoning problems in large ontologies.

The expressiveness of description logics is always limited. This property is forced by the need to ensure the decidability of reasoning problems and can be tightened to ensure the decidability of reasoning problems in polynomial time with regards to the size of the knowledge base. Decidability, especially in polynomial time, is a major factor that promotes the use of description logics in knowledge base engineering, therefore the design of varieties of specialized description logics requires careful analysis and formal proofs of their properties. This need is caused by the fact that a synthesis of decidable description logic extensions can lead to undecidability. If the expressiveness of description logic is insufficient to represent the domain of interest, then instead of using undecidable description logics, it is better to build a custom application around the decidable DL or use alternative knowledge representation models. The definition of semantic structures that can be represented in description logic is carried out within the framework of formal model theory, which in particular defines the semantics of first-order logic.

Some kinds of description logics go beyond first-order logic and can use operators of modality, the concepts from fuzzy logic, or probability theory. Syntactically, the statements of description logics are similar by form to the expressions of modal logics, but their truth values are established according to A. Tarski’s semantic truth theory [3], which is the paradigm of truth definition in model theory. Despite the differences that are present in the varieties of description logics, the process of ontology engineering is mostly the same. At first, we define the dictionary of the domain of interest, which is later formalized into the TBox statements. Some applications may use only TBox, while others may the whole ontology and even databases. The logical consistency of an ontology structure, as well as processing of user queries to a knowledge base, are ensured by a module of automatic or semi-automatic logical reasoning, which is usually provided as part of an ontology development environment (i.e., Hermit reasoner [4] in Protégé IDE [5]).

3. Description logic ALC

The use of description logics for the conceptual modeling of the domain of interest involves the description of important abstractions that are inherent to it, as well as the individuals who are the instances of these abstractions. Description logics have four major components that participate in the process of creating a formal representation of the domain [2]:

1. *Concepts* are abstractions that are built from the concept names and role names using constructors of description logic. In model theory, the concept is interpreted as a set of elements, and it corresponds to a unary predicate in first-order logic.
2. *Roles* (role names) are the relationships between two concepts or individuals. In model theory, the role is interpreted as a set of pairs of elements that belong to the appropriate concept extensions, and it corresponds to a binary predicate in first-order logic.
3. *Individuals* (instances) are the embodiments of concept abstractions in the form of objects and are designed to name the object of a particular abstraction. In model theory, the individual is interpreted as an element of a concept's extension set, and it corresponds to a constant in first-order logic.
4. *Concept language* is a formal language that creates concepts and roles descriptions from primitives like concept names and role names, by the means of description logic operators. An interpretation of the concept language operators in a model theory puts them in line with the operators on sets, while in first-order logic they are compared with the operators on logical statements.

An attributive language with complements, *ALC*, belongs to the family of description logics *AL* – attributive languages that provide operators of atomic negation, concepts intersection, universal and existential quantifiers. Unlike the *AL* family, where only the complements of atomic concepts are available, the *ALC* language allows the complement of any well-formed concept [3]. The set of atomic concepts consists of the concept names, “top” \top and “bottom” \perp concepts; the rest are compound (complex) concepts because they are formed using the concept language operators. Thus, a set of *ALC* concept descriptions consists of concept names, role names, individual names, \top and \perp , concepts intersection and union, concept negation (complement), existential and value restrictions. The syntax of a formal language allows distinguishing well-formed expressions from arbitrary ones. An inductive definition of *ALC* concepts' syntax is as follows [2]:

- Every concept name is an *ALC* concept description;
- \top and \perp are *ALC* concepts descriptions;
- If C and D are *ALC* concept descriptions and r is a role name, then the following are also *ALC* concept descriptions:
 - $C \sqcap D$ (conjunction or intersection of concepts);
 - $C \sqcup D$ (disjunction or union of concepts);
 - $\neg C$ (negation or complement of concept);
 - $\exists r.C$ (existential restriction);
 - $\forall r.C$ (values restriction).

The formal semantics of a language allows to establish unambiguously the meaning of its well-formed expressions and is also required for the introduction of a truth predicate, which assigns a truth value to the logical expressions in the language. Semantics can be considered as a metalanguage, which contains the structures that are described by the syntax of the corresponding formal language. Since metalanguage is also a language, it is possible to construct its own semantics, and so on. A. Tarski's semantic truth theory [3] states: to avoid the Russell's or liar's paradoxes in a formal language, the truth predicate must be defined exclusively in a metalanguage. This paradigm preserves the consistency of the logical systems, which allow forming the statements like “This sentence is a lie”; the truth value of this statement is determined outside the language that was used for its formulation.

The semantics of description logics is an “interpretation” – a structure that is characterized by the following properties: 1) consists of a non-empty set of elements, which is called *interpretation domain*; 2) for each concept name defines its *extension*, i.e., a subset of elements from the interpretation domain; 3) for each role name defines a set of pairs of elements from the interpretation domain, which are related to each other by this role. Formally, an interpretation $I = (\Delta^I, \cdot^I)$ consists of a non-empty set Δ^I , which

is called the interpretation domain, and a mapping \cdot^I , which maps each concept name $A \in \mathcal{C}$ to a set $A^I \subseteq \Delta^I$, and each role name $r \in \mathcal{R}$ to a binary relation $r^I \subseteq \Delta^I \times \Delta^I$. Interpretation of the other concept descriptions is carried out by extending the mapping \cdot^I with the following rules [2]:

- $T^I = \Delta^I$ (entire interpretation domain),
- $\perp^I = \emptyset$ (empty set);
- $(C \sqcap D)^I = C^I \cap D^I$ (intersection of concepts extensions);
- $(C \sqcup D)^I = C^I \cup D^I$ (union of concepts extensions);
- $(\neg C)^I = \Delta^I \setminus C^I$ (interpretation domain without concept extension);
- $(\exists r. C)^I = \{d \in \Delta^I \mid \text{exists } e \in \Delta^I \text{ with } (d, e) \in r^I \text{ and } e \in C^I\}$;
- $(\forall r. C)^I = \{d \in \Delta^I \mid \text{for all } e \in \Delta^I, \text{ if } (d, e) \in r^I, \text{ then } e \in C^I\}$.

The properties of an interpretation are limited only by the rules that are explicitly specified in its definition, namely: the interpretation domain should not be empty, but can be of arbitrary power, even infinite; concept extension may contain any number of elements, from zero to the entire interpretation domain; the role name extension may contain any number of pairs of elements, from zero to all possible pairs. Modification of the interpretation I , by adding or removing elements from interpretation domain, concepts extensions, or roles extensions, gives rise to the alternative interpretations. For the sake of interpretation structure analysis, it is convenient to visualize it in the form of an oriented graph, where the arcs and vertices are labeled. In this graph, a vertex represents an element from the interpretation domain, while its labels denote the concepts that contain the specified element in their extensions; an arc represents a pair of elements contained in the extension of some role r , and the arc label corresponds to the name of that role.

Example 1. Create a model of the domain “IT project structure” using the description logic *ALC* and define its interpretation in the context of the “Smart City” project.

Solution. The first step of modeling is to determine the concept and role names that are important for us in the domain of interest. Let the set of concept names $\mathcal{C} = \{\text{Engineer}, \text{Technology}, \text{Project}\}$, the set of role names $\mathcal{R} = \{\text{is_part_of}, \text{has}\}$. The second step of modeling is to define an interpretation for the elements of these sets. Concepts and roles extensions usually adhere to the desired context and purposes of modeling:

$$\begin{aligned}\Delta^I &= \{\text{Andy, Mary, Javascript, UML, Smart_City}\}; \\ \text{Engineer}^I &= \{\text{Andy, Mary}\}; \\ \text{Technology}^I &= \{\text{JavaScript, UML, Smart_City}\}; \\ \text{Project}^I &= \{\text{Smart_City}\}; \\ \text{is_part_of}^I &= \{(\text{Andy, Smart_City}), (\text{UML, Smart_City})\}; \\ \text{has}^I &= \{(\text{Mary, UML}), (\text{Andy, Javascript})\}.\end{aligned}$$

The third step of modeling consists in the formalization of interpretation elements using the syntax of description logic *ALC*. It should be noted that there might be multiple options for interpretation elements representation and each of them might be valid, but their efficiency for achieving the goals of modeling might vary. For example, the instance “Smart_city” belongs to the extensions of the atomic concept *Project* and the compound concept *Project* \sqcap *Technology*, as well as the extensions of the concepts like $\neg \text{Engineer}$ and T . The instance “Andy” belongs to the extensions of the concepts *Engineer*, $\forall \text{has. Technology}$, $\exists \text{is_part_of}. \text{Project}$ and many others. Moreover, all elements of the interpretation domain always belong to the extension of the top concept “T” and at the same time, none of them belong to the extension of the bottom concept “ \perp ”, which corresponds to the state of absurdity in the systems of logic.

The specified interpretation of the conceptual model of the domain of interest can be visualized in the form of a directed graph, which facilitates the analysis of the relationships between the elements of concepts extensions and simplifies the task of clusters identification:

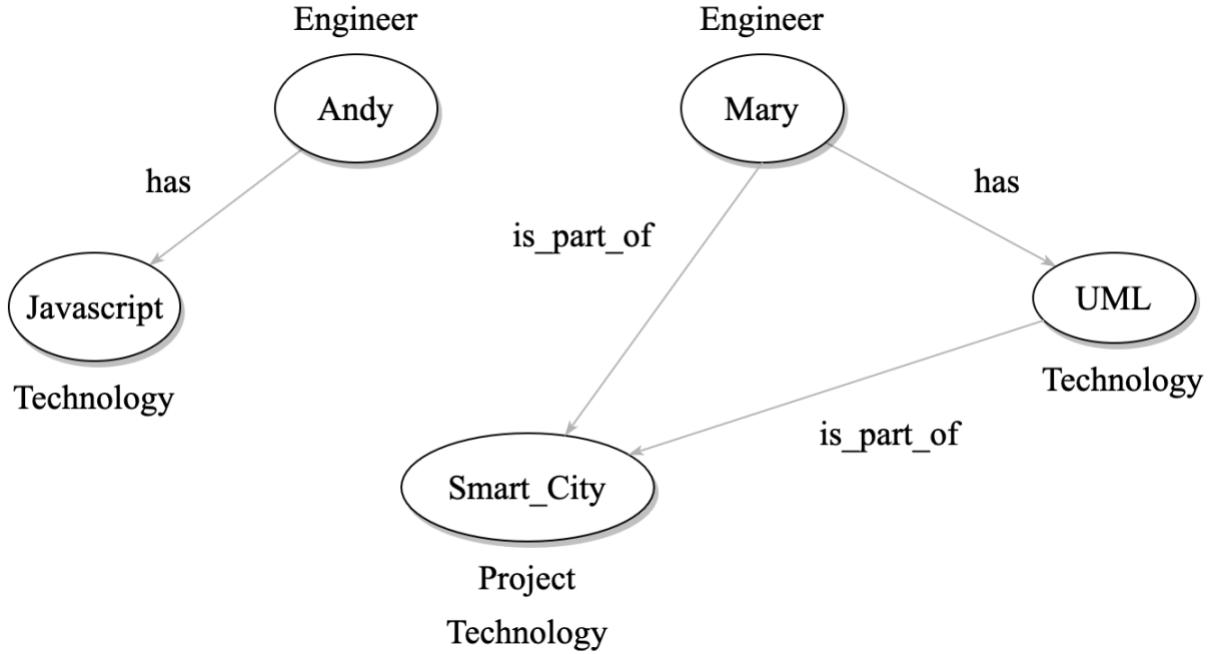


Figure 1: Visual representation of the interpretation in the form of a graph

The definition of the semantics of description logic ALC contains rules that allow to state some useful lemmas for the effective implementation of the reasoning process. Their syntax is very similar to the deductive forms of first-order logic arguments, namely, the law of double negation, de Morgan's laws, etc. Following are the lemmas of description logic ALC and their proofs can be obtained from the properties of interpretation I [2]:

- $\top^I = (C \sqcup \neg C)^I$ (the law of excluded middle, LEM);
- $\perp^I = (C \sqcap \neg C)^I$ (absurdity);
- $(\neg\neg C)^I = C^I$ (double negation elimination);
- $\neg(C \sqcap D)^I = (\neg C \sqcup \neg D)^I$ (de Morgan's law);
- $\neg(C \sqcup D)^I = (\neg C \sqcap \neg D)^I$ (de Morgan's law);
- $(\neg(\exists r. C))^I = (\forall r. \neg C)^I$ (variation of de Morgan's law);
- $(\neg(\forall r. C))^I = (\exists r. \neg C)^I$ (variation of de Morgan's law).

4. Knowledge bases in ALC

The ability to construct formal descriptions of the domain concepts and to combine them using concept language operators is an important, but not sufficient achievement. The sequences of syntactic transformations and combinations of formal descriptions can be infinite, so having limited time and memory resources, it makes sense to remember useful (by naming) and connected (by relations) descriptions, to avoid repetitive calculations. As a result, the following syntactic constructions are introduced to ALC description logic:

- $C \sqsubseteq D = C^I \sqsubseteq D^I$ (concept inclusion);
- $C \equiv D = C^I \sqsubseteq D^I \text{ and } D^I \sqsubseteq C^I$ (concepts equivalence);
- $a: C = a^I \in C^I$ (concept assertion);
- $(a, b): r = (a^I, b^I) \in r^I$ (role assertion).

These relations allow structuring the elements of the knowledge base in accordance with the classical approaches to systematization and reuse of concepts and their properties. For example, the equivalence operator \equiv establishes the meaning of the concept C through the description of the concept D , which corresponds to the encyclopedic format of the concept's definition. The representation of background knowledge is conveniently carried out through the operator of concept inclusion \sqsubseteq , which marks that concept C belongs to the more general concept D , and thus implicitly inherits from D some properties

and operational restrictions. Representation of concrete contexts requires the specification of concept and role instances in the form of assumptions. The operator “ $:$ ” establishes the affiliation of individual a to the concept C and a pair of individuals (a, b) to the role r , respectively. The usage of the specified operators is presented in the following examples:

$$\begin{aligned} \text{Technology} &\equiv \text{Methods} \sqcup \text{Tools} \\ \text{Engineer} &\equiv \text{Human} \sqcap \exists \text{has. Technology} \\ \text{Mary} &: \text{Engineer} \\ \text{UML} &: \text{Technology} \\ (\text{Mary}, \text{UML}) &: \text{has} \end{aligned}$$

In description logics, the knowledge base is usually divided into two structural parts: terminological, $TBox$, and declarative, $ABox$. In general-purpose knowledge bases or terminologies, the declarative part may be missing. The terminological part contains statements that describe the concepts and relationships between them (\equiv , \equiv), while declarative consists of statements that describe individuals of concepts and relationships between them in the form of roles ($:$). Comparing the structure of the knowledge base and the database, it is possible to establish a significant similarity between the $TBox$ and the database schema, as both contain general limitations: the form of the conceptual model of the domain and the form of the data, respectively. The $ABox$ is similar to the database content because it contains expressions related to the concrete instances of concepts and roles.

Formally, $TBox$ consists of a finite set of statements of the form $C \sqsubseteq D$, where C and D can be descriptions of both atomic and compound concepts. If $C \sqsubseteq D$ and $D \sqsubseteq C$, the notation $C \equiv D$ is used for the ease of operating. Each of these statements is called a general concept inclusion (GCI) or an axiom [2]. An interpretation I satisfies GCI if $C^I \sqsubseteq D^I$ is satisfied. If interpretation I satisfies every GCI containing $TBox T$, then it is called a model of T . The larger number of GCI $TBox$ contains, the fewer models exist for it, because for two $TBox T, T'$, if $T \sqsubseteq T'$ then each model T' is a model T . $TBox T$ divides a set of interpretations into two parts: a set of T models and a set of interpretations that do not satisfy all GCIs in T . Thus, $TBox$ focuses the modeling efforts exclusively on interpretations that meet the designer's intuitive expectations about the structure of the domain.

The declarative part of the knowledge base, $ABox$, consists of a finite set of assertions, which are represented in the form $a: C$, the concept assertion, or $(a, b): r$, the role assertion. It is assumed that an interpretation mapping I^A maps every instance a to the element $a^I \in \Delta^I$. An interpretation I satisfies the concept assertion $a: C$, if $a^I \in C^I$, and satisfies the role assertion $(a, b): r$, if $(a^I, b^I) \in r^I$. If interpretation I satisfies all concept and role assertions contained in the $ABox A$, then it is called a model of A .

The combination of $TBox T$ and $ABox A$ forms an ALC knowledge base $K = (T, A)$, in which concepts and their definitions are specified in the terminological part, and then used in the declarative part. An interpretation I , which is both a model of T and a model of A , is called a model of K . A knowledge base is similar to a database by structure, but they significantly differ in the approach of ensuring data consistency. If the data is not present in the database, then its validity is considered as “false” and can be interpreted as a violation of data integrity. For the knowledge base, it is quite acceptable to have a description of a concept in $TBox$ without a description of the concept instances in $ABox$. This paradigm is called the “open world assumption” [6].

The syntax of description logic ALC allows the creation of cyclic concept descriptions. For instance, the concept $\text{OpenSource} \equiv \text{Technology} \sqcap \forall \text{use. OpenSource}$ is cyclic since the concept name OpenSource appears in the description of its own definition. Such expressions can be satisfied by multiple interpretations and can lead to ambiguity of solutions for some reasoning problems. This issue can be mitigated by the introduction of a syntactic restriction, which increases the defining power of $TBox$ and introduces the notion of acyclic $TBox$. It is worth noting that the form of cyclicity of the OpenSource concept is simple and obvious, but there are hidden forms of cyclicity that arise due to the nesting of concept descriptions.

Let the expression $A \equiv C$ be a definition of the concept A , where C is a possibly compound concept, and T is a finite set of concept descriptions. If $A \equiv C \in T$ and the concept name B appears in C , then A

directly uses B . The concept A uses B if A directly uses B , or there is a concept name B' , such that A uses B' , and B' directly uses B . A finite set of concepts descriptions T is called an acyclic TBox, if: 1) there are no concept names in T that use themselves; 2) there are no concept names that appear more than once in the left part of the definition of the concept in T . If T is an acyclic TBox with $A \equiv C \in T$, then A is called strictly defined in T , and C is called the definition of A in T .

The importance of an acyclic TBox is that it can be unfolded into ABox by interpreting concept descriptions as macros. The mechanism of the process is to recursively replace all concept names in ABox with their definitions from TBox. Let $K = (T, A)$ be a knowledge base with acyclic TBox, which has the form $T = \{A_i \equiv C_i \mid 1 \leq i \leq m\}$. Let $A_0 = A$ and A_{j+1} be the result of the following substitutions: 1) find some $a: D \in A_j$, in which D directly uses A_i , $1 \leq i \leq m$; 2) replace in D all A_i with C_i . If no more substitutions can be applied in A_k , then A_k is the result of unfolding T into A . Unfolding is an important step in the process of solving many reasoning problems, but it can cause an exponential explosion of ABox size. Therefore, it is important to remember the strategies of the lazy and greedy concept descriptions unfolding.

5. Reasoning problems in ALC

The structure of a knowledge base in description logics allows to model concepts, roles, individuals, and relations between them. The number of statements in the knowledge base, the nesting of concept descriptions, model discovery, establish the need for reliable and effective methods of logical reasoning, which would allow obtaining answers about the important properties of a knowledge base. Let $K = (T, A)$ is an ALC knowledge base, C and D are possibly compound ALC concepts, and b is an individual name. Then, the basic reasoning problems inherent to the ALC knowledge base can be defined as follows [2]:

- C is *satisfied* with respect to T if there exists a model I of T and some element $d \in \Delta^I$ such that $d \in C^I$;
- C is *included* in D with respect to T , and is written as $T \models C \sqsubseteq D$ if $C^I \subseteq D^I$ in all models I of T ;
- C is *equivalent* to D with respect to T , and is written as $T \models C \equiv D$ if $C^I = D^I$ in all models I of T ;
- K is *consistent* if there is a model of K ;
- b is an *instance* of C with respect to K , and is written as $K \models b: C$ if $b^I \in C^I$ in all models I of K .

The sign of the semantic logical consequence \models , in contrast to the sign of the deductive logical consequence \vdash , is based on the inclusion of models of statements that it connects. When T is fixed, inclusion and equivalence with respect to T are binary relations of possibly compound concepts, so to denote this fact, the notations $C \sqsubseteq_T D$ and $C \equiv_T D$ are used, respectively. The problems of satisfaction, inclusion and equivalence are defined with respect to TBox only, but the problems of consistency between the knowledge base and concept's instance are defined with respect to both, TBox and ABox. The consistency property of TBox only, or ABox only, is considered as a case of consistency of the knowledge base of the specific form: $K = (T, \emptyset)$ for TBox and $K = (\emptyset, A)$ for ABox.

The relation of concept inclusion \sqsubseteq is transitive, and the ALC description logic is monotonic, i.e., the more statements the knowledge base contains, the more logical consequences it allows. Equivalences that can be derived from every TBox are called tautologies, for example, $T \models C \equiv D$ if and only if $T \models \top \equiv (\neg C \sqcup D)$. It is important to note that there are also hidden relationships between the basic problems of reasoning, which contribute to the process of their solution. Let $K = (T, A)$ is an ALC knowledge base, C and D are possibly compound ALC concepts, and b is an individual name. Then the following statements hold:

- $C \equiv_T D$ if and only if $C \sqsubseteq_T D$ and $D \sqsubseteq_T C$;
- $C \sqsubseteq_T D$ if and only if $C \sqcap \neg D$ is not satisfied with respect to T ;
- C is satisfied with respect to T if and only if $C \not\sqsubseteq \perp$;
- C is satisfied with respect to T if and only if $(T, \{b: C\})$ is consistent;
- $(T, A) \models b: C$ if and only if $(T, A \cup \{b: \neg C\})$ is inconsistent;

- If T is acyclic and A' is the result of the unfolding of T into A , then K is consistent if and only if (\emptyset, A') is consistent.

The outcome of the specified relationships between the reasoning problems is the fact that all the basic problems can be reduced to a common one – the consistency of a knowledge base. Thus, the presence of an efficient algorithm for calculating the consistency property of a knowledge base allows solving the other basic reasoning problems. It is worth noting that there are also reasoning problems that can't be reduced to the problem of knowledge base consistency or cause an exponential increase of the problem size during the reduction, i.e., query answering using conjunction.

Knowledge base engineering usually involves gradual filling of the TBox and ABox with statements about the domain of interest. Since the formal syntax, and concepts nesting, introduce some complexity for an engineer on establishing the properties of a knowledge base, there is a need for the auxiliary tools that will evaluate the consequences of knowledge base extension or modification. In description logics, this task is performed by the reasoning services, which injectively correspond to the basic problems of reasoning:

- Given a TBox T and a concept C , determine whether C is satisfied with respect to T ;
- Given a TBox T and the concepts C and D , determine whether C is included in D with respect to T ;
- Given a TBox T and the concepts C and D , determine whether C is equivalent to D with respect to T ;
- Given a knowledge base $K = (T, A)$, determine whether K is consistent;
- Given a knowledge base $K = (T, A)$, a concept C , and an individual name b , determine whether b is an instance of C with respect to K .

These descriptions are specifications of the reasoning services, which might be implemented in multiple ways. Additionally, the basic reasoning services can be reused to create more intelligent services that reveal the potential of an automated retrieval of a set of homogeneous objects and solving reasoning problems about them. The “generalized” reasoning services include the following ones [2]:

- Classification of a TBox: given a TBox T , determine the hierarchy of inclusions of all concept names that are present in T with respect to T . That is, for each pair of concept names A and B that are defined in T , determine whether $A \sqsubseteq_T B$ and whether $B \sqsubseteq_T A$. The result of the service is an inclusion hierarchy, which can be conveniently presented in the form of a graph or Hesse diagram for partial order;
- Checking the satisfiability of concepts in T : given a TBox T , for each concept name A , determine whether A is satisfied with respect to T . The dissatisfaction of a concept is the sign of a modeling error;
- Instance retrieval: given a concept C and a knowledge base K , determine all the individual names b such that b is an instance of C with respect to K . The result of the service is a set of individual names that affiliates with the concept C ;
- Realisation of an individual name: given an individual name b and a knowledge base K , determine all the concept names A that are defined in T , for which b is an instance of A with respect to K . The result of the service is a set of concept names that are the realisation of an individual name.

6. Reasoning in ALC with tableaux algorithm

Reasoning over the statements in a knowledge base is a process that facilitates the achievement of several goals: 1) substantiation or refutation of the existing knowledge; 2) generation of the new knowledge; 3) finding solutions to decision-making problems; 4) shared ontologies engineering [7]. In logic, the reasoning process is considered as a “mechanized” syntactic calculus over the statements of a formal system and consists in the application of deductive derivation rules or constructive properties of the associated semantics. A reasoning method should possess three qualitative properties, decidability, completeness, and soundness, to be considered a decision-making procedure in a system of formal logic. Decidability requires the presence of proof that a reasoning method terminates for all admissible kinds of input data, and potentially proof of the existence of an “effective” algorithm, which

is characterized by acceptable time complexity. The completeness requires proof that a “valid” result will be returned for *all* admissible input data with the desired property, while soundness requires proof that a “valid” result will be returned *only* for the admissible input data with the desired property.

There are a variety of reasoning methods associated with the description logic ALC, which are based on different calculus approaches. The most widespread reasoning methods are the resolution method, sequent calculus, automata-based calculus, tableaux methods, query rewriting [8]. For the expressive description logics, the tableaux methods are a default choice, i.e., the hyper-tableaux method is a core of HermiT reasoner, which integrates as a module into the Protégé ontology development environment. Since most of the basic reasoning problems in the description logic ALC can be reduced to the problem of knowledge base consistency, the properties of the tableaux method as a decision-making procedure will be formulated relative to the input data of the knowledge base consistency problem. The proofs of the tableaux method properties are based mainly on the semantics of the description logic ALC, namely the structural features of those models that can be expressed in the ALC concept language.

By definition, a knowledge base K is consistent if there is a model I of K ; this model is called a witness to the consistency of K and is denoted by $I \models K$. The core idea of tableaux methods is to prove the consistency of a knowledge base $K = (T, A)$ by constructing a model I of K . A knowledge base K is inconsistent, if during the construction of a witness to the consistency of K some obvious contradictions are revealed that refute the possibility of the existence of model I of K . The process of witness construction starts with an A and, if necessary, extends it with the outcomes of restrictions that are imposed by the semantics and the axioms contained in A or T . The efficiency and validity of tableaux methods are ensured by the structure of a witness they are designed to build, namely the tree model. The tree model significantly limits the number of potential witnesses (efficiency), facilitates detection of structural loops in tree branches (decidability), and allows proving the existence of possibly infinite models using the method of structural induction (soundness). Moreover, the proof of absence of a tree model for a knowledge base K indicates the absence of any model for K (completeness).

Tableaux method’s principles can be conveniently analyzed in the context of three situations: for $K = (\emptyset, A)$, for $K = (T, A)$, where T is acyclic, and in the general case for $K = (T, A)$. Since all the concepts contained in T or A can be transformed to a negated normal form using de Morgan’s laws or duality between the quantifiers of existence and universality, the expression $\neg C$ will correspond to the negated normal form of $\neg C$. An ABox A is considered normalized if the following conditions are met:

- all concepts contained in T or A are in the negated normal form, so the negation operator is applied only to the concept names and is not preceded by parentheses, quantifiers, or operators;
- each individual name contained in A is present in at least one statement of the form $a: C$;
- A is not empty, thus contains statements.

For the case $K = (\emptyset, A)$, when TBox is empty, the expansion rules mirror the semantics of compound concepts and apply it to the statements in A . Since the main principle of the tableaux method is to decompose compound concepts into sub-concepts, there always exists an iteration, where all compound concepts are completely decomposed into atomic ones, and the algorithm terminates. Each satisfied ALC concept has a tree-like model, and since the concept instances can be related by roles, a consistent ALC knowledge base K has a forest-like model. If K is consistent, then it has a model that consists of a nonempty set of disjointed trees, where the root of a tree represents some individual name in A and can be connected by edges with the roots of other trees. The tableaux method builds a forest-like model of an ABox A using syntactic expansion rules, which extend A with the new statements only if their conditions are met.

Table 2

Syntactic expansion rules for ALC ABox consistency

Name	Condition 1	Condition 2	Action
\sqcap -rule	$a: C \sqcap D$	$\{a: C, a: D\} \not\subseteq A$	$A \cup \{a: C, a: D\}$
\sqcup -rule	$a: C \sqcup D$	$\{a: C, a: D\} \cap A = \emptyset$	$A \cup \{a: X\}, X \in \{C, D\}$
\exists -rule	$a: \exists r. C \in A$	$\nexists b, \{(a, b): r, b: C\} \subseteq A$	$A \cup \{(a, d): r, d: C\}, d \text{ is new}$
\forall -rule	$\{a: \forall r. C, (a, b): r\} \subseteq A$	$b: C \notin A$	$A \cup \{b: C\}$

The tableaux method applies the expansion rules to the ABox statements until A is complete because it is easier to detect logical contradictions or clashes in the complete ABox. An ABox A contains a *clash* if for some individual name a and for some concept C , $\{a: C, a: \neg C\} \subseteq A$; if A does not contain a clash, then A is called *clash-free*. A is called *complete* if it contains a clash or none of the expansion rules can be applied due to unmet conditions.

Since any ALC ABox can be transformed to a normalized form, without reducing the generality, the algorithm for determining the consistency of an ABox will take as an input a normalized ABox. Then, the tableaux algorithm for ALC ABox consistency can be presented via the following pseudo-code [2]:

Algorithm *Consistency*(A):

If *Expand*(A) $\neq \emptyset$

Then return «consistent»

Else return «inconsistent»

Procedure *Expand*(A):

If A is not complete

Then select expansion rule R , whose conditions are satisfied in A , and select few statements α from A , which are eligible for R application

If exists $A' \in \exp(A, R, \alpha)$, such that *Expand*(A') $\neq \emptyset$

Then return *Expand*(A')

Else return \emptyset

Else

If A contains a clash

Then return \emptyset

Else return A

Here, the auxiliary function $\exp(A, R, \alpha)$ returns a set of ABox, which are formed by the application of rule R to the statement α in A . All the expansion rules, except the \sqcup -rule, are deterministic and return a set with one ABox. Depending on the way of its application, the \sqcup -rule can generate two different ABox, and it is mandatory to verify their consistency. The way of selecting the sequence of expansion rules R and statements α does not affect the result of the tableaux method but can drastically affect its performance.

The tableaux method for establishing the consistency of a knowledge base $K = (T, A)$, where T is an acyclic TBox, is largely based on the expansion rules and procedures used for $K = (\emptyset, A)$. This property is due to the fact that an acyclic TBox can be entirely unfolded into ABox A , and it actually reduces the current consistency problem to the previous case. The greedy strategies of unfolding T into A can lead to an exponential increase of the knowledge base size, so it is recommended to use the “lazy” unfolding strategies, which unfold concept descriptions only when necessary [9]. Below are the lazy unfolding rules, which together with the previous expansion rules form the basis of a tableaux method for establishing the consistency of a knowledge base with an acyclic TBox.

Table 3

Syntactic rules for expanding the axioms of an acyclic ALC TBox

Name	Condition 1	Condition 2	Action
\sqsubseteq -rule	$A \sqsubseteq C \in T, \quad a: A \in A$	$a: C \notin A$	$A \cup \{a: C\}$
\equiv_1 -rule	$A \equiv C \in T, \quad a: A \in A$	$a: C \notin A$	$A \cup \{a: C\}$
\equiv_2 -rule	$A \equiv C \in T, \quad a: \neg A \in A$	$a: \neg C \notin A$	$A \cup \{a: \neg C\}$

Establishing the consistency in the general case of a knowledge base $K = (T, A)$, where T may contain cyclic concept descriptions, requires some auxiliary constructions that will ensure decidability. Without reducing the generality, the input parameter of a tableaux algorithm is a normalized knowledge base $K = (T, A)$ that consists of a normalized TBox T and a normalized ABox A . A TBox T is called

normalized if all its axioms have the form $T \sqsubseteq E$, where E is reduced to the negated normal form. Reduction of axioms to the expected form can be carried out using the following properties:

- I satisfies $C \sqsubseteq D \iff I$ satisfies $T \sqsubseteq D \sqcup \neg C$;
- I satisfies $C \equiv D \iff I$ satisfies $T \sqsubseteq (D \sqcup \neg C) \sqcap (C \sqcup \neg D)$.

The set of expansion rules for a knowledge base consistency consists of five elements, and three of them, namely \sqcap -rule, \sqcup -rule, \forall -rule, are completely identical to the ABox expansion rules. The other two rules are also similar to their counterparts but have some important changes: \sqsubseteq -rule uses a normalized form of TBox axioms, while \exists -rule contains an additional requirement for a “non-blocking” individual, which is a property that ensures decidability or algorithm termination.

Table 4

Modified syntactic rules for expanding a general ALC knowledge base

Name	Condition 1	Condition 2	Action
\sqsubseteq -rule	$T \sqsubseteq D \sqcup \neg C \in T,$ $a: C \in A$	$a: D \notin A$	$A \cup \{a: D\}$
\exists -rule	$a: \exists r. C \in A,$ a is not blocked	$\nexists b, \{(a, b): r, b: C\} \subseteq A$	$A \cup \{(a, d): r, d: C\},$ d is new in A

A termination problem of the tableaux algorithm for general knowledge base consistency is caused by the possibility to combine \sqsubseteq -rule and \exists -rule, which without the blocking mechanism can generate the infinite repetitive sequences of individuals [10]. If an ABox does not contain clashes, then its model can describe the infinite sequences using loops. During the execution of the tableaux algorithm, only the \exists -rule adds new vertices (individuals) and edges (roles) to the ABox tree model. If the \exists -rule adds a new individual b and a new role assertion $(a, b): r$, then b is the successor of a and a is the predecessor of b . The transitive closure of roles between successor and predecessor creates the notions of descendant and ancestor, respectively. Let $con_A(a) = \{C \mid a: C \in A\}$ is a set of concept names that are present in the concept assertions of the form $a: C$. The individual name b is blocked in ALC ABox A if the following conditions are met: 1) a is the ancestor of b ; 2) $con_A(a) \supseteq con_A(b)$. Thus, if an individual name is blocked, then all its descendants are also blocked. Since root individuals don't have ancestors, they are never blocked.

Example 2. Verify the consistency of an ALC knowledge base $K = (T, A)$ of the “Mass media” domain using the tableaux method. The structure of the knowledge base is as follows:

```

 $T = \{$ 
    News  $\sqsubseteq \forall \text{has. Author};$ 
    Author  $\equiv \text{Human} \sqcup \text{Organization} \sqcup \text{Bot};$ 
    Bot  $\sqsubseteq \neg \text{Human} \sqcap \forall \text{has. Author};$ 
    Organization  $\sqsubseteq \exists \text{has. Human};$ 
 $\};$ 

 $A = \{$ 
    a: News;
    (a, b): has;
    b: Bot;
    (b, c): has;
    c: Organization;
 $\}.$ 

```

Solution. The first step consists in a normalization of the knowledge base, which serves as an input argument to the tableaux method. Note that the ABox is already normalized, but the statements from TBox still require normalization, which can be performed using the properties stated above and de Morgan's laws:

$$T = \{$$

$$\begin{aligned} T \sqsubseteq \forall \text{has. Author} \sqcup \neg \text{News}; \\ T \sqsubseteq (\text{Human} \sqcup \text{Organization} \sqcup \text{Bot}) \sqcup \neg \text{Author}; \\ T \sqsubseteq (\neg \text{Human} \sqcap \forall \text{has. Author}) \sqcup \neg \text{Bot}; \\ T \sqsubseteq \exists \text{has. Human} \sqcup \neg \text{Organization}; \end{aligned}$$

};

Upon providing the normalized knowledge base to the tableaux algorithm, its execution begins. At each iteration, the algorithm selects an expanding rule and a statement (or a set of statements) that satisfy its conditions. Let's select \sqsubseteq -rule and apply it to the statement $T \sqsubseteq \forall \text{has. Author} \sqcup \neg \text{News}$. The specified statement belongs to T , $a: \text{News} \in A$, and $a: \forall \text{has. Author} \notin A$, thus a new concept assertion is added to ABox $A = A \cup \{a: \forall \text{has. Author}\}$.

Let's apply the \sqsubseteq -rule to the rest of the statements contained in the TBox. The statement $T \sqsubseteq (\text{Human} \sqcup \text{Organization} \sqcup \text{Bot}) \sqcup \neg \text{Author}$ does not satisfy all conditions of the \sqsubseteq -rule, namely $d: \text{Author} \notin A$. The other statements do satisfy the rule conditions, so they can be expanded. After an application of the \sqsubseteq -rule to the statements in T , the ABox A is as follows:

$$A = \{$$

$a: \text{News};$	$a: \forall \text{has. Author};$
$(a, b): \text{has};$	$b: \neg \text{Human} \sqcap \forall \text{has. Author};$
$b: \text{Bot};$	$c: \exists \text{has. Human};$
$(b, c): \text{has};$	
$c: \text{Organization};$	

).

Next, let's apply the \exists -rule to the statement $c: \exists \text{has. Human}$. Since the individual name $c \in A$ and corresponds to the root in a tree model, then c is not blocked. Because $\nexists d, \{(c, d): \text{has}, d: \text{Human}\} \subseteq A$, the statement $c: \exists \text{has. Human}$ can be expanded by \exists -rule. As a result, the following statements are added to A : $(c, d): \text{has}$ and $d: \text{Human}$, where d is the new individual name in A .

$$A = \{$$

$a: \text{News};$	$a: \forall \text{has. Author};$
$(a, b): \text{has};$	$b: \neg \text{Human} \sqcap \forall \text{has. Author};$
$b: \text{Bot};$	$c: \exists \text{has. Human};$
$(b, c): \text{has};$	$(c, d): \text{has};$
$c: \text{Organization};$	$d: \text{Human};$

).

At the next iteration, let's select the \sqcap -rule and apply it to the statement $b: \neg \text{Human} \sqcap \forall \text{has. Author}$. Both conditions of the \sqcap -rule are satisfied, $b: \neg \text{Human} \notin A$ and $b: \forall \text{has. Author} \notin A$, therefore $A = A \cup \{b: \neg \text{Human}, b: \forall \text{has. Author}\}$. Then, let's apply the \forall -rule to the statements $a: \forall \text{has. Author}$ and $b: \forall \text{has. Author}$. Both of them satisfy the conditions of \forall -rule, $(a, b): \text{has} \in A$, $b: \text{Author} \notin A$ and $(b, c): \text{has} \in A$, $c: \text{Author} \notin A$, thus $A = A \cup \{b: \text{Author}, c: \text{Author}\}$. After the application of \sqcap -rule and \forall -rule, the ABox A is as follows:

$$A = \{$$

$a: \text{News};$	$a: \forall \text{has. Author};$	$b: \neg \text{Human};$
$(a, b): \text{has};$	$b: \neg \text{Human} \sqcap \forall \text{has. Author};$	$b: \forall \text{has. Author};$
$b: \text{Bot};$	$c: \exists \text{has. Human};$	$b: \text{Author};$
$(b, c): \text{has};$	$(c, d): \text{has};$	$c: \text{Author};$
$c: \text{Organization};$	$d: \text{Human};$	

).

Now let's apply the \sqsubseteq -rule to the statement $T \sqsubseteq (\text{Human} \sqcup \text{Organization} \sqcup \text{Bot}) \sqcup \neg \text{Author}$ twice, considering that $\{b: \text{Author}, c: \text{Author}\} \in A$. Therefore, $A = A \cup \{b: \text{Human} \sqcup \text{Organization} \sqcup \neg \text{Author}\}$.

Bot; c: Human \sqcup Organization \sqcup Bot}. Conditions of the \sqcup -rule are not satisfied because b: Bot \in A and c: Organization \in A. The other rules are not applicable to the statements in A or T, so A is complete. Moreover, A does not have any explicit clashes of the form $\{x: D; x: \neg D\} \subseteq A$, so A is clash-free.

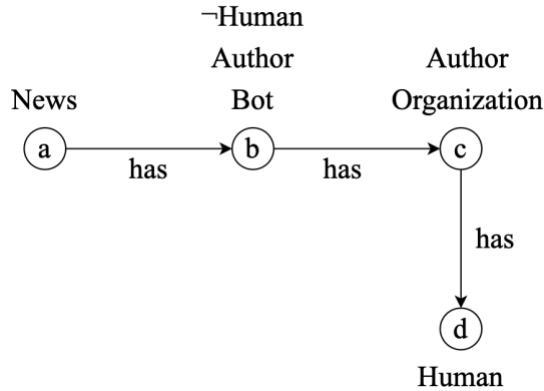


Figure 2: Visual representation of the found model

Answer. The ALC knowledge base $K = (T, A)$ of the “Mass media” domain is consistent. The tableaux method has created a witness to the consistency of the knowledge base K , namely a model I of K . Its visual representation contains all individuals and the atomic concepts that they belong to, as well as the roles that connect them.

7. Conclusion

System analysis of the description logic ALC, which acts as a basis for more expressive description logics like SROIQ^(D), identified the main principles of knowledge representation in the description logics, their model-theoretic semantics, the notion and structure of knowledge bases, core reasoning problems and reasoning over them with the tableaux method. An application of the ALC formalism to the representation of the “Mass media” domain highlighted the peculiarities of knowledge specification and the inference of implicit knowledge from the existing one. The benefits of automated reasoning are especially noticeable in large and complex knowledge bases.

8. Acknowledgements

The authors would like to thank Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler for writing “An Introduction to Description Logic” [2]. Many of the formal definitions and notations associated with the described logic are given in this excellent textbook.

9. References

- [1] W3C, OWL 2 Web Ontology Language Direct Semantics (Second Edition), 2012. URL: <https://www.w3.org/TR/owl2-direct-semantics>.
- [2] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An introduction to description logic, Cambridge University Press, Cambridge, 2017. doi:10.1017/9781139025355.
- [3] A. Tarski, Pojęcie prawdy w językach nauk dedukcyjnych, Prace Towarzystwa Naukowego Warszawskiego, Wydział III Nauk Matematyczno-Fizycznych, volume 34, 1933.
- [4] The Data and Knowledge Group at the Oxford University, Hermit OWL Reasoner. URL: <http://www.hermit-reasoner.com>.
- [5] Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, Protégé: a free, open-source ontology editor & framework for building intelligent systems. URL: <https://protege.stanford.edu>.

- [6] C. M. Keet, Open World Assumption, in: W. Dubitzky, O. Wolkenhauer, K. H. Cho, H. Yokota (Eds.), Encyclopedia of Systems Biology, Springer, New York, NY, 2013, p. 1567. doi:10.1007/978-1-4419-9863-7_734.
- [7] R. Brachman, H. Levesque, Knowledge Representation and Reasoning, Morgan Kaufmann Publishers Inc, San Francisco, CA, 2004.
- [8] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, The description logic handbook: theory, implementation, and applications, 2nd. ed., Cambridge University Press, Cambridge, 2007. doi:10.1017/CBO9780511711787.
- [9] D. Scott, The lattice of flow diagrams, in: E. Engeler (Ed.), Symposium on Semantics of Algorithmic Languages, volume 188 of Lecture Notes in Mathematics, Springer, Berlin, Heidelberg, 1971, pp. 311–366. doi:10.1007/BFb0059703.
- [10] H. Levesque, R. Brachman, Expressiveness and tractability in knowledge representation and reasoning, Computational Intelligence, volume 3, issue 1, (1987) 78–93. doi:10.1111/j.1467-8640.1987.tb00176.x.