Imperial College London
Department of Computing

Tableau Compiled Labelled Deductive Systems with an application to Description Logics

by

Aikaterini Marazopoulou

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London.

September 2009

Acknowledgements

I owe my sincerest gratitude to my thesis supervisor, Dr Krysia Broda, for her insightful guidance towards the completion of this project. I am deeply indebted to her for the endless hours she spent with me during the summer, for her inexhaustible enthusiasm, and for her constant encouragement.

I would also like to thank my family, my parents and my sister, for their unlimited and unconditional support throughout the years. There are not enough words to describe how grateful I am for all the things they have done for me.

TABLEAU CLDS WITH AN APPLICATION TO DESCRIPTION LOGICS

AIKATERINI MARAZOPOULOU
SUPERVISED BY DR KRYSIA BRODA

Abstract. Labelled deductive systems (LDS) provide a unifying framework for different logics of the same family. Compiled labelled deductive systems extend the notion of LDS to provide a uniform proof system and semantics for logics of different families. Usually, the proof system associated with a CLDS is natural deduction. Such CLDSs have been well studied for several logics (normal modal logics, substructural logics, fuzzy logics etc.). The purpose of this project is to provide a tableau proof procedure for Compiled Labelled Deductive Systems, something that, to the extend of our knowledge, has not been formally done in the past. Moreover, we present a tableau CLDS for several description logics.

Contents

1. Introduction

1.1. **Related work.** The existence of a plethora of logical systems that vary from each other in several ways (syntax of the underlying language, semantics, or the notion of derivability relation just to name a few) is an undeniable fact. This motivated Gabbay to introduce a general framework, Labelled Deductive Systems (LDS) [17], that would allow the uniform representation of logics that belong to the same family. The main idea of this approach is to annotate every formula with a label that contains some "extra information". For example, in the case of modal logics, labels can refer to names of worlds and a modal formula could be annotated with the worlds where it is true (of course, this is an oversimplification of how modal logics are actually modelled as labelled deductive systems). In that sense, LDS do not handle formulas but labelled formulas. Moreover, derivation rules do not only apply to formulas, but to labels as well.

Compiled Labelled Deductive Systems (CLDS) is an extension of LDS introduced by Russo in [23]. CLDS constitute a unifying framework for logics belonging to different families. To be more specific, CLDS provide a common notion of derivability relation and semantic entailment for families of different logics. This is achieved by using a "compilation" technique into first-order logic, i.e. by translating part of the CLDS into first-order logic, and using first-order semantics. This unifying approach can be applied to any logic that is first-order axiomatizable.

Various logics have been formalized as Compiled Labelled Deductive Systems. Besides normal modal logics [23], CLDS have been studied for conditional logics [11], as well as for Access Control Logic [12]. Moreover, there have been applications to substructural logics [9], propositional resource logics [8], and fuzzy logics [1]. In the aforementioned cases, the proof system associated with the CLDS was natural deduction. It has to be mentioned that there exists a brief description of a tableau CLDS for hybrid logic [13], however the details of the formalization and of the proofs were not fully worked out.

Since this project is considerably related to description logics, it seems appropriate to present the related work in this area too. Description Logics (DL) are in general knowledge representation formalisms. Knowledge representation systems based on description logics contain a DL-knowledge base and provide ways to reason about the content of the base (reasoning tasks). A DL knowledge base consists of two components, the TBox that contains terminological information (i.e. information about classes of objects) and the ABox which contains assertional information (i.e. information regarding individuals). One of the most important characteristics of description logics is that they are decidable. It is crucial that there exist "efficient" algorithms for the reasoning tasks.

In description logics the vast majority of reasoning algorithms are tableau-based algorithms. It has to be noted that there exist tableau algorithms for all the description languages that we will use in this report see for example [3] for the basic tableau algorithms, as well as [19] for a tableau algorithm for the very expressive description language $\mathcal{SROIQ}$ (which is more expressive than the ones we consider here).

1.2. **Our work.** In this report we present a tableau CLDS for description logics which is based on the description language $\mathcal{ALC}$. Like in the case of CLDS

for modal logics, the "basic" $\mathcal{ALC}$-CLDS can be extended by adding axioms in the labelling algebra that express certain properties of the roles. This allows us to formalize a family of description languages in the CLDS framework and capture their differences in the labelling algebra. Furthermore, we describe in detail how the information of a knowledge base can be incorporated into the CLDS-framework and how it can be used for reasoning.

Another important part of this work is that the proof procedure associated with the $\mathcal{ALC}$-CLDS is a tableau proof procedure. We formalize the notion of tableau in the CLDS framework and we present a sound, complete, and terminating tableau calculus for the $\mathcal{ALC}$-CLDS. The formalization of the tableau and the tableau rules, as well as the techniques used to prove the soundness and completeness of the tableau calculus for the $\mathcal{ALC}$-CLDS could be generalized and applied in a CLDS for another logic.

We also consider several interesting extensions of the $\mathcal{ALC}$-CLDS. First of all, we examine the case of CLDS for Description Languages with more concept constructors. To be more specific, we present an outline of a CLDS for the description language $\mathcal{ALCN}$ that contains number restrictions. Next, we roughly describe a CLDS for temporal description logics. This is a first step towards exploring how more than one labels can be used in a CLDS. Finally, we present a small application of how description logic programs [18] can be described in the $\mathcal{ALC}$-CLDS.

1.3. **Structure of the report.** The next four sections of this report are dedicated to preliminaries. All this introductory information is necessary in order to keep this report as self-contained as possible. To be more specific, in section 2 we briefly introduce the basic notions of first-order logic. Section 3 focuses on the formal description of trees that will be later used to formalize the notion of tableau. The fourth section is a concise introduction to Compiled Labelled Deductive Systems and is kept to an abstract level. Finally, section 5 presents the basic notions of description logics and of the language $\mathcal{ALC}$ in particular.

The main part of the report starts in section 6 with the introduction of the basic CLDS for the description language $\mathcal{ALC}$, the $\mathcal{ALC}$-CLDS. The next section contains a formal description of a tableau calculus for the $\mathcal{ALC}$-CLDS. In section 8 we introduce the knowledge base in the $\mathcal{ALC}$-CLDS and in section 9 we examine how to simulate the reasoning tasks of description logics in the $\mathcal{ALC}$-CLDS under the empty knowledge base. Sections 10 and 11 address the case of $\mathcal{ALC}$-CLDS and the corresponding tableau in the presence of a non-empty knowledge base that contains only terminological information. In this case the "naive" tableau algorithm may not terminate. In order to obtain a decision procedure, we introduce in section 12 a blocking technique. Similarly, sections 13 and 14 focus on $\mathcal{ALC}$-CLDS and the corresponding tableau calculus under a full knowledge base (that contains assertional information as well). Finally, in sections 15 and 16 axioms are added in the labelling algebra of the CLDS.

The remaining sections describe some possible extensions and directions for future work. To be more specific, section 17 studies the formalization of a more expressive description language, and section 18 the formalization of temporal description logics in the CLDS framework. Finally, section 19 contains a small

application regarding description logic programs, that combine description logics and logic programming.

2. First-order Logic

In this section we will briefly describe the basic notions of first-order logic and we will introduce the notation used in the rest of the report. This is not intended to be an in-depth analysis of first-order logic and there are excellent textbooks that cover this field in detail. The reason, however, we need to present some first-order notions is that first-order logic is closely related to CLDS. To be more specific, the semantics of CLDS are first-order semantics. Moreover, some of the languages used in the CLDS may be first-order languages.

2.1. **Syntax.**

We can distinguish between elements that are part of every first-order language, and those that are different among them depending on the application domain. The common elements in all first-order languages are the following:

- A (finite or countably infinite) set $\mathcal{X}$ of individual variable symbols. We will use the letters x and y to denote individual variables (with appropriate indices when necessary).
- The logical connectives $\neg$, $\wedge$, $\vee$, and $\rightarrow$.
- The quantifiers $\forall$ (universal quantifier), and $\exists$ (existential quantifier).
- The punctuation symbols '(' (left parenthesis), ')' (right parenthesis), and ',' (comma).

On the other hand, the elements that vary from language to language are:

- A (finite or countably infinite) set of predicate symbols $\mathcal{P} = \{P_0, \ldots\}$.
- A (finite or countably infinite) set of function symbols $\mathcal{F} = \{f_0, \ldots, \}$.
- A (finite or countably infinite) set of constant symbols $\mathcal{C} = \{c_0, \ldots, \}$.

A first-order language can be determined by specifying the set of predicate symbols, the set of function symbols, and the set of constant symbols. We write $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$ to denote the first-order language $\mathcal{L}$ determined by $\mathcal{P}, \mathcal{F}$, and $\mathcal{C}$ (we say that $\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$ is the signature of $\mathcal{L}$).

Every predicate symbol and every function symbol is associated with a positive integer that represents the arity of the predicate or the function. We define a mapping *arity* that associates every predicate and function symbol to its arity. For example, $arity(P_0) = 2$ means that P_0 is a binary predicate symbol. When no confusion arises, the arity of a predicate (of function) will not be explicitly stated. To be more specific, if we write $P_0(t_1)$, it is implied that P_0 is a unary predicate symbol ($arity(P_0) = 1$).

Definition 1 (Term)**.** Let $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$ be a first-order language. The set of terms for this language is the smallest set that meets the following conditions:

- Any variable is a term of $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$.
- Any constant symbol (member of $\mathcal{C}$) is a term of $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$.
- If f is a function symbol, $arity(f) = n$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term of $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$.

Definition 2 (Atomic formula)**.** Let $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$ be a first-order language. If $P^n \in \mathcal{P}$ is a predicate symbol and $t_1, \ldots, t_n$ are terms of $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$, then $P(t_1, \ldots, t_n)$ is an atomic formula of $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$.

Definition 3 (Well-formed formulas)**.** The set of well-formed formulas (or simply formulas) of a first-order language $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ is the smallest set that meets the following conditions:

- Any atomic formula of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ is a formula of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$.
- If ϕ and ψ are formulas of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ and x is an individual variable symbol, then the following are also formulas of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$:

$$\neg\phi \qquad (\phi \wedge \psi) \qquad (\phi \vee \psi) \qquad (\phi \rightarrow \psi) \qquad \forall x\phi \qquad \exists x\phi$$

Sometimes, $\phi \wedge \psi$ will be used as an abbreviation for $(\phi \wedge \psi)$. Similar abbreviations will be used for the the formulas $(\phi \vee \psi)$ and $(\phi \rightarrow \psi)$. Moreover, we will use $\top$ as an abbreviation for $(\phi \vee \neg\phi)$ and $\bot$ as an abbreviation for $(\phi \wedge \neg\phi)$, where ϕ is an arbitrary well-formed formula of a first-order language. Finally, $\phi \leftrightarrow \psi$ is an abbreviation for $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$.

Definition 4 (Free-variable occurrences)**.** Let ϕ, ψ be formulas of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$. We distinguish the following cases:

- If ϕ is an atomic formula, then the free-variable occurrences of ϕ are all the variable occurrences in ϕ.
- The free-variable occurrences of $\neg\phi$ are the free-variable occurrences of ϕ.
- The free-variable occurrences of $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \rightarrow \psi$ are the free-variable occurrences of ϕ and the free-variable occurrences of ψ.
- The free-variable occurrences of $\forall x\phi$ and $\exists x\phi$ are the free-variable occurrences of ϕ, excluding the occurrences of the variable x.

Definition 5 (Sentence)**.** A *sentence* of $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ is a formula of this language that contains no free variables.

Finally, if an expression (a formula or a term) does not contain any variables at all, free or not free, it is said to be *ground*.

2.2. **Semantics.**

Definition 6 (First-order structure)**.** A *structure* for the first-order language $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ is a pair $\mathcal{M} = \langle D, I\rangle$ where:

- D is a non-empty set, called the domain of $\mathcal{M}$.
- I is a mapping, called an interpretation, that maps:
  - Every constant symbol $c \in \mathcal{C}$ to some member of the domain $c^I \in D$.
  - Every n-ary function symbol $f \in \mathcal{F}$ to some n-ary function $f^I : D^n \rightarrow D$
  - Every n-ary predicate symbol $P \in \mathcal{P}$ to some n-place relation P^I on D ($P^I \subseteq D^n$).

Definition 7 (Assignment)**.** An *assignment* in a structure $\mathcal{M} = \langle D, I\rangle$ is a function $A : \mathcal{X} \rightarrow D$ that maps every individual variable to an element of the domain ($x^A \in D$).

Definition 8 (Interpretation of terms)**.** Let $\mathcal{L}\langle\mathcal{P},\mathcal{F},\mathcal{C}\rangle$ be a first-order language, $\mathcal{M} = \langle D, I\rangle$ a structure for $\mathcal{L}$, and A an assignment in $\mathcal{M}$. Every term t of $\mathcal{L}$ can be associated with an element of the domain as follows:

- If t is a constant symbol c, then $c^{I,A} = c^I$.
- If t is an individual variable x, then $x^{I,A} = x^A$.

- If t is of the form $f(t_1, \ldots, t_n)$, then $[f(t_1, \ldots, t_n)]^{I,A} = f^I(t_1^{I,A}, \ldots, t_n^{I,A})$.

Definition 9 (x-variant)**.** Let A and A' be two assignments in a structure $\mathcal{M}$. We say that A is an *x-variant* of A' iff A and A' assign the same values to all variables except possibly x. If $y \in \mathcal{X}$:

$$A'(y) = \begin{cases} A(y) \text{ if } y \neq x \\ d \in D \text{ if } y = x \end{cases}$$

Definition 10 (Truth)**.** A formula ϕ is said to be *true* in a first-order structure $\mathcal{M}$ and under an assignment A in $\mathcal{M}$ (notation:$\mathcal{M}, A \models_{FOL} \phi$) when:

If ϕ is an atomic formula:

$$\mathcal{M}, A \models_{FOL} P(t_1, \ldots, t_n) \qquad \text{iff } (t_1^{I,A}, \ldots, t_2^{I,A}) \in P^I$$

If ϕ and ψ are first-order formulas:

$$\begin{array}{ll}
\mathcal{M}, A \models_{FOL} \neg\phi & \text{iff not } \mathcal{M}, A \models_{FOL} \phi \\
\mathcal{M}, A \models_{FOL} \phi \wedge \psi & \text{iff } \mathcal{M}, A \models_{FOL} \phi \text{ and } \mathcal{M}, A \models_{FOL} \psi \\
\mathcal{M}, A \models_{FOL} \phi \vee \psi & \text{iff } \mathcal{M}, A \models_{FOL} \phi \text{ or } \mathcal{M}, A \models_{FOL} \psi \\
\mathcal{M}, A \models_{FOL} \phi \rightarrow \psi & \text{iff if } \mathcal{M}, A \models_{FOL} \phi \text{ then } \mathcal{M}, A \models_{FOL} \psi \\
\mathcal{M}, A \models_{FOL} \forall x\phi & \text{iff } \mathcal{M}, A' \models_{FOL} \phi \text{ for all x-variants } A' \text{ of } A \\
\mathcal{M}, A \models_{FOL} \exists x\phi & \text{iff } \mathcal{M}, A' \models_{FOL} \phi \text{ for some x-variant } A' \text{ of } A
\end{array}$$

In the case of the abbreviations $\top$ and $\bot$ it easy to show that:

$$\mathcal{M}, A \models_{FOL} \top \text{ and } \mathcal{M}, A \not\models_{FOL} \bot$$

Notice that if ϕ is a sentence, then the assignment A does not affect the truth of ϕ. To emphasize that, we will write $\mathcal{M} \models_{FOL} \phi$ to denote that ϕ is true in $\mathcal{M}$ when ϕ is a sentence. In this case we say that $\mathcal{M}$ is a *model* of the sentence ϕ.

In this report we adopt the following notational convention. Let $\forall x\phi$ be a sentence and $\mathcal{M}$ a first-order structure.

$$\mathcal{M} \models_{FOL} \forall x\phi \text{ if and only if } \mathcal{M}, [x \mapsto d] \models_{FOL} \phi \text{ for all } d \in D,$$

where by $[x \mapsto d]$ we denote an assignment that maps x to an element d of the domain. Similarly, if $\exists x\phi$ is a sentence,

$$\mathcal{M} \models_{FOL} \exists x\phi \text{ if and only if } \mathcal{M}, [x \mapsto d] \models_{FOL} \phi \text{ for some } d \in D.$$

Definition 11 (Satisfiability)**.** A formula ϕ is *satisfiable* if there exists a first-order structure $\mathcal{M}$ and an assignment A in $\mathcal{M}$ such that $\mathcal{M}, A \models_{FOL} \phi$.

Definition 12 (Valid)**.** A formula ϕ is *valid* iff for every structure $\mathcal{M}$ for the language and for every assignment A in $\mathcal{M}$, ϕ is true in $\mathcal{M}$ under A.

2.3. **Herbrand Structures.**

Definition 13 (Herbrand Universe)**.** Let $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$ be a first-order language and $\mathcal{C} \neq \emptyset$ (if $\mathcal{C} = \emptyset$ we consider $\mathcal{C}$ to contains a single constant symbol $\mathcal{C} = \{c\}$). Let $\mathcal{U}_0 = \mathcal{C}$. For $i \geq 0$, $\mathcal{U}_{i+1}$ is defined inductively as follows:

$$\mathcal{U}_{i+1} = \mathcal{U}_i \cup \{f(t_1, \ldots, t_i) \mid t_1, \ldots, t_i \in \mathcal{U}_i \text{ and } f \in \mathcal{F}\}.$$

The *Herbrand Universe* of $\mathcal{L}$ is $\mathcal{U} = \lim_{i \to \infty} \mathcal{U}_i$.

Similarly we can define the Herbrand Universe of a set S of first-order formulas. To be more specific, $\mathcal{U}_0 \subseteq \mathcal{C}$ is defined as the set of constants that appear in S (or if no constants appear in S it is considered to contain a single constant symbol like before).

Definition 14 (Herbrand Base)**.** The *Herbrand Base* of $\mathcal{L}$ if the set of ground atomic formulas of the form $P(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are members of the Herbrand universe of $\mathcal{L}$.

Similarly, the Herbrand base of a set of formulas can be defined as the subset of the Herbrand base that contains exactly the predicate symbols that appear in the set of formulas.

Definition 15 (Herbrand Structure)**.** Let $\mathcal{L} = \langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$ be a first-order language. A *Herbrand structure* for $\mathcal{L}$ is a first-order structure $\mathcal{M}_H = \langle \mathcal{U}, \mathcal{H} \rangle$ where

- $\mathcal{U}$ is the Herbrand universe of $\mathcal{L}$.
- $\mathcal{H}$ is a function (known as Herbrand interpretation) that maps
  - Every constant $c \in \mathcal{C}$ to itself $c^{\mathcal{H}} = c$ ($c \in \mathcal{U}$).
  - Every n-ary function symbol $f \in \mathcal{F}$ to the n-ary function $f^{\mathcal{H}} : \mathcal{U}^n \to \mathcal{U}$ such that $f^{\mathcal{H}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ (remember that $f(t_1, \ldots, t_n) \in \mathcal{U}$).

Notice that the Herbrand interpretation $\mathcal{H}$ does not specify what should be assigned to predicate symbols. Therefore, since what should be assigned to constant and function symbols is fixed by definition 15, a Herbrand interpretation can be determined by the set of ground atoms that are true or in other words, by a subset of the Herbrand base.

3. Trees

Trees have an important role in this report, as tableaux will be defined as labelled trees. In this section we will provide all the necessary definitions that are related to trees (both labelled and unlabelled).

3.1. **Unlabelled Trees.** Let Σ be a set of symbols, known as the alphabet. Members of Σ are usually called letters. A sequence of letters from an alphabet Σ is known as a word over Σ. Words are represented by simply putting the letters one after another. For example, *ababa* is a word over the alphabet $\{a, b\}$. In what follows, we will often use the following notation: for some letter $a \in \Sigma$ and some $k \in \mathbb{N}$, a^k denotes a word that consists of k consecutive occurrences of letter a:

$$a^k = \underbrace{aa \dots a}_{k \text{ times}}$$

Concatenation between two words u and v will be written either as uv or by using a binary operator $u \cdot v$. Moreover, the size (or length) of a word corresponds (informally) to the number of letters that the word contains. We will not give a formal definition since it is a rather intuitive notion. The size of a word w will be denoted by $|w|$. For example, $|abbbaaa| = 7$. Finally, we will use the symbol ϵ to denote the empty word, namely the word with size equal to 0.

Definition 16 (Substring)**.** Let u, v be two finite words over an alphabet Σ. We say that u is a *substring* of v if there exist two finite words w, w' over Σ such that $v = wuw'$.

Notice that if both w, w' are the empty word then the substring coincides with the word itself. If at least one of w, w' is required to be a non-empty word, then u is a *strict substring* of v.

In what follows, we will be mainly concerned with substrings of a word that appear in "the beginning of the word". Such substrings are referred to as prefixes and a formal definition is given below.

Definition 17 (Prefix (finite words))**.** Let Σ be a countable alphabet and u, w two finite words over Σ ($u, w \in \Sigma^*$). We say that u is a prefix of w (notation: $u \leq w$) if there exists a word $v \in \Sigma^*$ such that $w = uv$.

Example 18. Assume that we have the alphabet $\Sigma = \{a, b\}$. Some finite words over Σ are the following: *abbba*, *ab*, and the empty word ϵ (obviously they are not the only words of Σ).

The finite word *aba* is a word over Σ and is a substring of the word *bababb*. This is because there exist words over Σ, the words *b* and *bb*, such that *bababb* can be written as $b \cdot aba \cdot bb$.

The finite word *ab* is a word over Σ and is a prefix of *abbba* because there is a word over Σ, the word *ba*, such that $abb \cdot ba$ is *abbba*. Similarly, *ab* is a prefix of *ab* because $ab = ab \cdot \epsilon$. The finite word *abb* is a word over Σ and is a prefix of *abbba* but not of *ab*. The prefixes of *ab* are the following words: ϵ, a, ab.

Now we are able to move on and define the notion of trees. Informally, a tree is a set of words over an alphabet that meets certain closure conditions.

Definition 19 (Tree)**.** A *tree* $\mathcal{T}$ is a set of words over an alphabet Σ ($\mathcal{T} \subseteq \Sigma^*$) that is prefix closed, i.e. if $w \in \mathcal{T}$, then $u \in \mathcal{T}$ for all $u \leq w$.

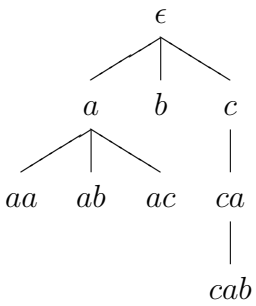


FIGURE 1. Graphical representation of tree $\mathcal{T}_1 = \{\epsilon, a, b, c, aa, ab, ac, ca, cab\}$.

Notice that the root of such a tree is the empty word ϵ. Moreover, a tree is said to be *finite* if it is a finite set of words over an alphabet, or in other words, if it contains finitely many nodes.

Example 20. Assume that $\Sigma = \{a, b, c\}$. The set of words

$$\mathcal{T}_1 = \{\epsilon, a, b, c, aa, ab, ac, ca, cab\}$$

is indeed a tree. Its graphical representation is shown in figure 1. On the other hand, the set of words $\mathcal{T}_2 = \{\epsilon, a, ca\}$ is not a tree. According to the definition, since $ca \in \mathcal{T}_2$, all the prefixes of ca should be in $\mathcal{T}_2$. However, c is a prefix of ca and $c \notin \mathcal{T}_2$.

In the scope of this report we will only use binary trees, i.e. trees where each node has at most two children. To define binary trees it is enough to restrict the above general definition of trees (definition 19) to alphabets that contain only two distinct symbols. Without loss of generality, we will consider binary trees to contain words over the alphabet $\{0, 1\}$. We will also impose one more restriction in order to obtain binary trees of a specific form. We will require every node that has a child, to have a 0-child.

Definition 21 (Binary Tree)**.** A binary tree is a set $\mathcal{T} \subseteq \{0, 1\}^*$ that has the following properties:

- It is prefix closed.
- If $w1 \in \mathcal{T}$, then $w0 \in \mathcal{T}$. That means that if a node has only one child, that is the 0-child.

An example of a binary tree is presented in figure 2. Notice that the set $\{\epsilon, 0, 1, 00, 01, 11\}$ although it is prefix closed, it is not a binary tree according to definition 21. Since node 11 is in this set, node 10 should also belong to the set, which does not. In other words, node 1 has a 1-child, but it has not a 0-child.

Every node of the tree uniquely identifies a finite path from the root of the tree to this node. Therefore, given a node, we can define the corresponding path from the root to that node as follows:

Definition 22 (Finite path)**.** Let $\mathcal{T}$ be a tree and n a node of $\mathcal{T}$. The (finite) path that corresponds to n is the set $p_n = \{v \in \mathcal{T} \mid v \leq l\}$.

If the node is a leaf, then the corresponding path is called branch. In other words, a branch is a path to a *leaf* node. A leaf node is informally a node that

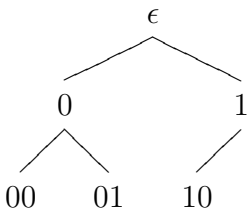


FIGURE 2. Graphical representation of the binary tree $\mathcal{T} = \{\epsilon, 0, 1, 00, 01, 10\}$

has no children. This is equivalent to saying that a leaf is a maximal element of the tree w.r.t. $\leq$.

Definition 23 (Leaf node)**.** Let $\mathcal{T}$ be a tree. A leaf node is a $\leq$-maximal element of $\mathcal{T}$.

Definition 24 (Finite branch)**.** Let $\mathcal{T}$ be a tree, and l a leaf node of $\mathcal{T}$. The (finite) branch that corresponds to l is the set $b_l = \{v \in \mathcal{T} \mid v \leq l\}$.

Since b_l is defined as a set of nodes, we can use the notation $|b_l|$ to denote the cardinality of the set, i.e. the number of nodes of the branch. Of course the same holds for paths.

It is not hard to generalize the above description for infinite trees. An infinite tree would be an infinite set of words over an alphabet. Notice that an infinite tree does not contain infinite words. It consists of infinitely many finite words. The following definition for infinite branches was taken from [3].

Definition 25 (Infinite branch)**.** Let $\mathcal{T}$ be an (infinite) tree. An infinite branch is a prefix closed subset of $\mathcal{T}$ such that for every $i \geq 0$ there exists exactly one node $n \in \mathcal{T}$ with $|n| = i$.

3.2. **Labelled Trees.** Informally, a labelled tree is an (unlabelled) tree which has a label assigned to every one of its nodes.

Definition 26 ($\mathcal{L}$-Labelled Tree)**.** An $\mathcal{L}$-labelled tree is a pair $\mathcal{T}_\mathcal{L} = \langle \mathcal{T}, f \rangle$ where $\mathcal{T}$ is an unlabelled tree and $f : \mathcal{T} \to \mathcal{L}$ a function that maps every node of the unlabelled tree to a member of the set $\mathcal{L}$.

A node of the labelled tree $\mathcal{T}_\mathcal{L} = \langle \mathcal{T}, f \rangle$ is a pair $\langle n, f(n) \rangle$, where $n \in \mathcal{T}$ and $f(n)$ is the label of the node.

Example 27. Consider again the binary tree of figure 2: $\mathcal{T} = \{\epsilon, 0, 1, 00, 01, 10\}$. Let $\mathcal{L} = \{a, b, c\}$ be the set of labels of the tree, and f a function that maps the label a to a node if the node contains at least one 1, otherwise it assigns b. A graphical representation of the labelled tree $\mathcal{T}_\mathcal{L} = \langle \mathcal{T}, f \rangle$ is shown in figure 3.

A branch of $\mathcal{T}_\mathcal{L}$ is a pair $B = \langle b_l, f \rangle$, where b_l is the branch of $\mathcal{T}$ whose leaf node is l. A labelled node $\langle n, f(n) \rangle$ belongs to a labelled branch B iff $n \in b_l$:

$$\langle n, f(n) \rangle \in B \Leftrightarrow n \in b_l$$

In order to simplify the notation we will sometimes use the notation $label \in B$ as an abbreviation for "there exists a node $n \in b_l$ such that $f(n) = label$":

$$label \in B \text{ iff there exists } n \in b_l \text{ such that } f(n) = label$$

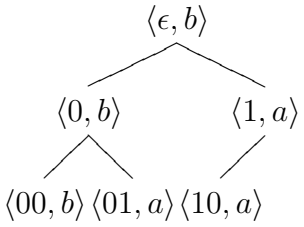


FIGURE 3. Graphical representation of the labelled tree $\mathcal{T}_{\mathcal{L}} = \langle \mathcal{T}, f \rangle = \langle \{\epsilon, 0, 1, 00, 01, 10\}, f, \{a, b, c\} \rangle$.

Similar definitions can be given in for a labelled path: $P = \langle p_n, f \rangle$ where p is the path from the root to node n. Moreover, we will say that a path $P_1 = \langle p_1, f \rangle$ is contained in $P_2 = \langle p_2, f \rangle$ and we will write $P_1 \subseteq P_2$ iff $p_1 \subseteq p_2$.

Example 28. Consider again the labelled tree of figure 3. The labelled branch whose leaf node is $\langle 00, b \rangle$ is $B = \langle b_{00}, f \rangle = \langle \{\epsilon, 0, 00\}, f \rangle$. In this case, $b \in B$ because there exists a node labelled with b (actually all the nodes of this branch are labelled with b). On the other hand, $a \notin B$ because there does not exist a node of B that is labelled with a. Let $P = \langle p_0, f \rangle$ be the path to node 0. Obviously $P \subseteq B$.

4. Compiled Labelled Deductive Systems

Labelled Deductive Systems (LDS) were introduced by Gabbay [17] in an attempt to provide a general framework for different logics that belong to the same family and therefore, they share some semantical and proof-theoretical notions. The basic idea is to annotate every formula of a logic with a label, which provides more information for this formula.

Compiled Labelled Deductive Systems (CLDS) are based on the LDS framework and provide a uniform representation for logics belonging to different families, but whose semantics is axiomatizable in first-order logic [10]. The notions of CLDS as will be described in this report were introduced in [23] for a Modal Labelled Deductive System (MLDS).

In this section we will describe the basic notions of CLDS in an abstract level. By that, we mean that we will not specify the logic that the CLDS formalizes (for example description or modal logics). Instead, we will use an arbitrary language and we will later narrow down the description to CLDS for Description Logics.

4.1. Languages and syntax of a general CLDS.

We will first describe the language of a CLDS. The following definitions are given here as presented in [10].

Definition 29 (CLDS language)**.** A *CLDS language* is an ordered pair $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ where $\mathcal{L}_{\#}$ is the language of the logic to formalize, and $\mathcal{L}_L$ is the labelling language.

The language $\mathcal{L}_{\#}$ could be a propositional or a first-order language. The labelling language $\mathcal{L}_L$ could be a suitable fragment of a first-order language.

Definition 30 (Labelling language $\mathcal{L}_L$)**.** A *labelling language* is a first-order language $\mathcal{L}_L\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$.

Sometimes it is convenient to skolemize formulas of the labelling language. To achieve that, we define a new language (the semi-extended labelling language) which extends the labelling language with appropriate Skolem function symbols.

Definition 31 (Semi-extended labelling language)**.** Given a CLDS language $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$, the *semi-extended labelling language* that corresponds to $\mathcal{L}_L$ is denoted by $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$ and is the language $\mathcal{L}_L$ extended with a set of Skolem function symbols.

Definition 32 (Labelling algebra)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language. A *labelling algebra* $\mathcal{A}$ is a first-order theory[1] written in $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$.

Definition 33 (Label)**.** Given a CLDS language $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$, a *label* is a *ground term* of the semi-extended labelling language $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$.

Generally, CLDS formulas are of two different types: declarative units which express that a formula is associated with a specific label, and Relation-literals that contain information on how labels are related to each other.

Definition 34 (Declarative unit)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language. A *declarative unit* is a pair $(a : \lambda)$ where a is a formula of $\mathcal{L}_{\#}$ and λ is a label.

In what follows we will sometimes use $(a : \lambda)$ as an abbreviation for $(a : \lambda)$.

[1] A theory is a set of sentences that is closed under logical consequence.

Definition 35 (Relation-predicate)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language, R an n-ary predicate of $\mathcal{L}_{\#}$, and $\lambda_1, \ldots, \lambda_n$ labels. A *Relation-predicate* is a predicate of the form $R(\lambda_1, \ldots, \lambda_n)$.

Sometimes we will make use of *Relation-literals* namely of literals[2] of the form $R(\lambda_1, \ldots, \lambda_n)$ or $\neg R(\lambda_1, \ldots, \lambda_n)$.

It has to be noted that in [10] the term R-predicate is used instead of Relation-predicate because the labelling language contained only one predicate symbol, R. However, since we will later use labelling languages that have more than one predicate symbols, we opted the more general term Relation-predicate.

Definition 36 (CLDS-formula)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS-language. A CLDS-formula is either a declarative unit or a Relation-literal.

Notice that a CLDS language is not a set of CLDS-formulas.

Definition 37 (Diagram)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language. A *diagram* $\mathcal{D}$ is a set of Relation-literals.

Definition 38 (Configuration)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language. A *configuration* $\mathcal{C}$ is a pair $\langle \mathcal{D}, \mathcal{F} \rangle$ where $\mathcal{D}$ is a diagram and $\mathcal{F}$ is a function that maps every label to a set of $\mathcal{L}_{\#}$-formulas.

Equivalently, a configuration can be defined as a pair $\mathcal{C} = \langle \mathcal{D}, S \rangle$ where $\mathcal{D}$ is the diagram and S a set of declarative units as follows:

$$S = \{(a : t) \mid t \text{ is a label and } a \in \mathcal{F}(t)\}.$$

More generally, a configuration can be defined as a set of CLDS-formulas $\mathcal{C} = \mathcal{D} \cup S$. In this case we can directly refer to the size of a configuration (i.e. the number of Relation-literals and declarative units it contains) as the cardinality of the set $\mathcal{D} \cup S$: $|\mathcal{C}| = |\mathcal{D} \cup \ S|$.

It is easy to show that the above three definitions are equivalent. In what follows we will use mainly the last one, however, all three of them may be used depending on the context.

Definition 39 (Compiled Labelled Deductive System)**.** Let $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ be a CLDS language. A *Compiled Labelled Deductive System* (CLDS) is a tuple

$$\langle \langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle, \mathcal{A}, \mathcal{R} \rangle$$

where $\mathcal{A}$ is a labelling algebra, and $\mathcal{R}$ is a set of inference rules.

Inference rules are related to the proof system that will be used. In [10] the proof system is natural deduction and the inference rules are defined accordingly. Here we will use a tableau proof procedure, therefore the set of inference rules will be a set of tableau rules. More details on the tableau rules will be provided in the following chapters.

A CLDS for the language $\mathcal{L}_{\#}$ will be referred to as $\mathcal{L}_{\#}$-CLDS.

[2] A literal is an atom or a negated atom.

4.2. **Semantics.** In order to provide a model-theoretic semantics for a compiled labelled deductive system, the CLDS language is translated into first-order logic and first-order semantics is used.

The main idea of the translation is that each declarative unit $(a:\lambda)$ will be translated to a unary predicate $[a]^*(\lambda)$. Relation-literals are translated as themselves. The relationships between different monadic predicates are expressed by first-order axiom schemata that reflect the semantics of the topmost operator of each formula. The semi-extended language $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$ enriched with the set of all the monadic predicates that correspond to well-formed formulas of $\mathcal{L}_{\#}$ forms the *extended labelling language*, which is used to describe the aforementioned first-order axiom schemata.

Definition 40 (Extended labelling language)**.** Let $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$ be a semi-extended labelling language, and let $a_1, \ldots, a_n, \ldots$ be an enumeration of $\mathcal{L}_{\#}$-formulas. The *extended labelling language* $\mathrm{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$ is defined as the language $\mathrm{Func}(\mathcal{L}_{\#}, \mathcal{L}_L)$ extended with the set $\{[a_1]^*, \ldots, [a_n]^*, \ldots\}$ of monadic predicate symbols.

Definition 41 (Extended algebra)**.** An *extended algebra* $\mathcal{A}^+$ is a first-order theory written in $\mathrm{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$ that extends the labelling algebra $\mathcal{A}$ of a CLDS with axiom schemata on the monadic predicates.

Definition 42 (First-order translation of a configuration)**.** Given a CLDS and a configuration $\mathcal{C} = \langle \mathcal{D}, \mathcal{F} \rangle$, the translation of $\mathcal{C}$ into first-order logic (notation: $\mathrm{FOT}(\mathcal{C})$) is the theory written in $\mathrm{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$ and defined as:

$$\mathrm{FOT}(\mathcal{C}) = \mathcal{D} \cup \mathcal{DU}$$

where $\mathcal{DU} = \{[a]^*(\lambda) \mid a \in \mathcal{F}(\lambda), \lambda \text{ is a label}\}$.

Definition 43 (Semantic structure of a CLDS)**.** Given a CLDS $\mathfrak{C}$ and its associated extended algebra $\mathcal{A}^+$, $\mathcal{M}$ is a semantic structure of $\mathfrak{C}$ if and only if $\mathcal{M}$ is a model of $\mathcal{A}^+$ (i.e. every formula of $\mathcal{A}^+$ is true in $\mathcal{M}$).

According to this definition, a semantic structure for a given CLDS is a first-order structure for the corresponding extended labelling language.

Now it is possible to define the notion of satisfiability for declarative units, Relation-literals, and configurations.

Definition 44 (Satisfiability)**.** Let $\mathfrak{C}$ be a compiled labelled deductive system.

- A declarative unit $(a:\lambda)$ is satisfiable iff there exists a semantic structure $\mathcal{M}$ of $\mathfrak{C}$ such that $\mathcal{M} \models_{FOL} [a]^*(\lambda)$ ($\mathcal{M}$ is said to satisfy $(a:\lambda)$). We write:

$$\mathcal{M} \models_{CLDS} (a:\lambda)$$

- A Relation-literal Δ is satisfiable iff there exists a semantic structure $\mathcal{M}$ of $\mathfrak{C}$ such that $\mathcal{M} \models_{FOL} \Delta$ ($\mathcal{M}$ is said to satisfy Δ). We write:

$$\mathcal{M} \models_{CLDS} \Delta$$

- A configuration $\mathcal{C}$ is satisfiable iff there exists a semantic structure $\mathcal{M}$ of $\mathfrak{C}$ such that $\mathcal{M}$ satisfies all declarative units and Relation-literals of $\mathcal{C}$. We write:

$$\mathcal{M} \models_{CLDS} \mathcal{C}$$

5. Description Logics

Description languages consist in general of two basic building blocks: concepts that are used to represent classes of objects, and roles that describe the relationships between different objects. Atomic concepts can be combined in order to create more complex ones through the use of constructors. Similarly, in more expressive description languages, complex roles can be created from atomic ones.

In what follows we will describe in a concise manner the basic notions of description logics. A more detailed introduction can be found in [3].

5.1. **Syntax of $\mathcal{ALC}$.** $\mathcal{ALC}$ (Attributive Language with Complement) was introduced in [24] and is one of the simplest non-trivial description languages. The alphabet of $\mathcal{ALC}$ consists of a set of atomic concepts $\mathcal{N}_C$, a set of atomic roles $\mathcal{N}_R$, the logical symbols $\sqcap, \sqcup, \exists, \forall$ and the punctuation symbols "." (dot), "(" (left parenthesis), ")" (right parenthesis). Concepts in $\mathcal{ALC}$ are formed according to the following syntax rule:

$$C ::= \mathsf{A} \mid \neg\mathsf{C} \mid (\mathsf{C} \sqcup \mathsf{D}) \mid (\mathsf{C} \sqcap \mathsf{D}) \mid \forall r.\mathsf{C} \mid \exists r.\mathsf{C}$$

where the symbol A is used for atomic concepts, C for (complex) concepts, and r for atomic roles.

5.2. **Semantics of $\mathcal{ALC}$.** The semantics is defined in terms of an interpretation $\mathcal{I}$, which consists of a non-empty empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation), and a function $\cdot^{\mathcal{I}}$ that maps atomic concepts to a subset of the domain and atomic roles to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$:

$$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}, \text{where } \mathsf{A} \text{ is an atomic concept}$$
$$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}, \text{where } r \text{ is an atomic role}$$

This interpretation is inductively extended to concept descriptions.

$$\begin{aligned}
[\neg C]^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
[C \sqcap D]^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
[C \sqcup D]^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
[\forall r.C]^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \text{for all } b, (a,b) \in r^{\mathcal{I}} \text{ implies } b \in C^{\mathcal{I}}\} \\
[\exists r.C]^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \text{there exists a } b \text{ such that: } (a,b) \in r^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}
\end{aligned}$$

We will use the following abbreviations: $\top = (\mathsf{A} \sqcup \neg\mathsf{A})$ and $\bot = (\mathsf{A} \sqcap \neg\mathsf{A})$. The interpretation of these concepts is: $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$.

A concept C is *satisfiable* iff there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$ (in this case we say that $\mathcal{I}$ is a *model* for C).

5.3. **Reasoning Tasks.** A knowledge base expressed in description logics consists of two components: the TBox that contains information about concepts and how they are related to each other, and the ABox which contains information about individuals.

5.3.1. *Terminological Knowledge (TBox).* A TBox is a finite set of statements of the following four types [16]:

$\mathsf{A} \doteq \mathsf{C}$	Concept Definition
$\mathsf{A} \dot{\sqsubseteq} \mathsf{C}$	Primitive Concept Specification
$\mathsf{C} \doteq \mathsf{D}$	Concept Equation
$\mathsf{C} \dot{\sqsubseteq} \mathsf{D}$	Concept Inclusion

A TBox that contains only primitive concept specifications and concept definitions is called a *simple* TBox. We say that an atomic concept A *directly uses* another atomic concept A' if A' appears in the left-hand side of the definition of A. Let *uses* be the transitive closure of the relation *directly uses*. A TBox contains a *cycle* if an atomic concept A *uses* itself [3]. If a TBox contains concept equations and/or concept inclusions it is called a *free* TBox. Notice that concept definitions and primitive concept specifications are special cases of concept equation and concept inclusion respectively. Therefore, a free TBox may also contain concept definitions and primitive concept specifications.

Example 45. Assume that *Mother*, *Female*, and *Person* are atomic concept names, and *hasChild* is an atomic role name. A TBox for this application domain may contain the following statements:

$$\mathit{Mother} \doteq \mathit{Female} \sqcap \exists \mathit{hasChild}.\top \tag{5.1}$$

$$\mathit{Mother} \dot{\sqsubseteq} \mathit{Female} \tag{5.2}$$

$$\mathit{Mother} \sqcap \forall \mathit{hasChild}.\mathit{Female} \dot{\sqsubseteq} \mathit{Person} \sqcap \exists \mathit{hasChild}.\top \tag{5.3}$$

The first one is a concept definition which says that someone is a mother if that someone is a female and moreover has a child. Statement 5.2 is a primitive concept specification. It captures the fact that every mother is female. Finally, 5.3 is a concept inclusion according to which everyone that is a mother and has only female children is also a person that has at least one child.

An interpretation $\mathcal{I}$ satisfies a statement $\mathsf{A} \doteq \mathsf{C}$ (similarly $C \doteq D$) iff $\mathsf{A}^{\mathcal{I}} = \mathsf{C}^{\mathcal{I}}$ ($C^{\mathcal{I}} = D^{\mathcal{I}}$), and a statement $\mathsf{A} \dot{\sqsubseteq} \mathsf{C}$ ($C \dot{\sqsubseteq} D$) iff $\mathsf{A}^{\mathcal{I}} \subseteq \mathsf{C}^{\mathcal{I}}$ ($C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is a *model* for a TBox if it satisfies all the statements of the TBox.

The reasoning tasks that are associated with the TBox are:

- Concept Satisfiability: A concept C is satisfiable with respect to a TBox $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $\mathsf{C}^{\mathcal{I}} \neq \emptyset$. If there exists no such interpretation $\mathcal{I}$, then the concept C is said to be *unsatisfiable* with respect to $\mathcal{T}$.
- Subsumption: Concept C is subsumed by concept D with respect to a TBox $\mathcal{T}$ (notation: $\mathsf{C} \sqsubseteq_{\mathcal{T}} \mathsf{D}$) iff $\mathsf{C}^{\mathcal{I}} \subseteq \mathsf{D}^{\mathcal{I}}$ for every interpretation $\mathcal{I}$ that is a model of $\mathcal{T}$.

Subsumption can be reduced to satisfiability. To be more specific, a concept C is subsumed by a concept D if and only if the concept $(\mathsf{C} \sqcap \neg \mathsf{D})$ is unsatisfiable.

Example 46. Let $\mathcal{T}$ be the TBox presented in example 45 and assume that we want to check if the concept $\mathit{Mother} \sqcap \neg \mathit{Female}$ is satisfiable w.r.t. $\mathcal{T}$. Suppose that it is satisfiable w.r.t. $\mathcal{T}$. That means that there exists an interpretation $\mathcal{I}$ such that $(\mathit{Mother} \sqcap \neg \mathit{Female})^{\mathcal{I}} \neq \emptyset$, or equivalently $\mathit{Mother}^{\mathcal{I}} \cap (\neg \mathit{Female})^{\mathcal{I}} \neq \emptyset$. It follows that there exists an element of the domain $d \in \Delta^{\mathcal{I}}$ such that $d \in \mathit{Mother}^{\mathcal{I}}$

and $d \in (\neg Female)^{\mathcal{I}}$. This means that $d \notin Female^{\mathcal{I}}$. However, $\mathcal{I}$ must be a model of the TBox since we have assumed that the initial concept is satisfiable w.r.t. $\mathcal{T}$. It follows that $\mathcal{I}$ satisfies axiom 5.2: $Mother^{\mathcal{I}} \subseteq Female^{\mathcal{I}}$. In other words, for all elements $d \in \Delta^{\mathcal{I}}$ such that $d \in Mother^{\mathcal{I}}$ it holds that $d \in Female^{\mathcal{I}}$. This is a contradiction, therefore, the initial concept is not satisfiable w.r.t. $\mathcal{T}$.

Notice however, that the concept is satisfiable w.r.t. the empty knowledge base. We can define an interpretation such that there exists a $d \in \Delta^{\mathcal{I}}$ that belongs to the interpretation of *Mother* but not in the interpretation of *Female*.

5.3.2. *Assertional Knowledge (ABox).* As we have mentioned earlier, the ABox contains information regarding individuals. To include this kind of information the alphabet of $\mathcal{ALC}$ is extended with a set of symbols for individuals, $\mathcal{N}_I$. In what follows we will use the letters a, b to refer to individuals. To define the corresponding semantics, the interpretation function $\cdot^{\mathcal{I}}$ is extended in order to map individuals to elements of the domain:

$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$$

At this point we have to mention that if the Unique Name Assumption (UNA) is adopted, then if two individuals have different names, they are mapped to different elements of the domain: $a \neq b$ iff $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. In the literature, some authors enforce the UNA while others do not. In this report we will not adopt the Unique Name Assumption. The reasons will become clear later, when a tableau rule for number restrictions will be introduced (section 17).

An ABox contains concept assertions of the form $C(a)$ meaning that the individual a belongs to the interpretation of the concept C, and role assertions $r(a, b)$ which denote that the individual b is a filler of the role r for the individual a.

An interpretation $\mathcal{I}$ satisfies the assertion $C(a)$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and the assertion $r(a, b)$ iff $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. $\mathcal{I}$ is a *model* for an ABox if it satisfies all the assertions in the ABox. These definitions can be generalized for satisfiability w.r.t. a TBox. To be more specific, an interpretation is said to satisfy an assertion of the ABox w.r.t. a TBox, if it satisfies the assertion and in addition it is a model of the TBox.

The main reasoning tasks associated with the ABox are the following.

- Consistency of the ABox: An ABox $\mathcal{A}$ is said to be consistent with respect to a TBox $\mathcal{T}$ if there is an interpretation $\mathcal{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$.
- Instance checking: Given an ABox $\mathcal{A}$, an individual a is an instance of a concept description C with respect to a TBox $\mathcal{T}$ if for every interpretation $\mathcal{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$, $a^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$.

It is easy to see that instance checking can be reduced to ABox inconsistency [3]. To be more specific, a is an instance of C w.r.t. an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ if and only if $\mathcal{A} \cup \{\neg \mathsf{C}(a)\}$ is inconsistent w.r.t. $\mathcal{T}$.

Example 47. Let *MARY* and *ANN* be individual names. Given the concept and role names defined in the previous examples, a possible ABox $\mathcal{A}$ contains the following statements:

$$Mother(ANN) \tag{5.4}$$

$$Female(MARY) \tag{5.5}$$

$$hasChild(ANN, MARY) \tag{5.6}$$

Ann is asserted to be a mother, Mary is female, and Mary is the child of Ann. It can be easily seen that this is a consistent ABox w.r.t. $\mathcal{T}$. However, if we add the following assertion

$$\neg Female(ANN) \tag{5.7}$$

then $\mathcal{A}$ is not consistent w.r.t. $\mathcal{T}$.

A final remark regarding the ABox is that the information contained in an ABox is assumed to be incomplete. In other words, if something is not explicitly asserted in the ABox, it is *not* assumed to be false. It is just unknown. In the previous example, we do not know if Mary is a mother. In some interpretations of $\mathcal{A}$ Mary is a mother and in some others she is not. That means that the ABox has "open-world" semantics.

Finally, it is worth mentioning that the axioms of both the TBox and the ABox are *not* $\mathcal{ALC}$ formulas (or more generally, they are not formulas of any description language). They are rather axioms in a meta-language.

An $\mathcal{ALC}$ language can be determined by the set of atomic concept symbols, the set of atomic role symbols, and the set of individual symbols that it contains ($\mathcal{N}_C, \mathcal{N}_R$ and $\mathcal{N}_I$ respectively). In what follows, we will use the notation $\mathcal{ALC}\langle\mathcal{N}_C, \mathcal{N}_R, \mathcal{N}_I\rangle$ to denote the $\mathcal{ALC}$ language that has the specified sets of symbols.

5.4. **More expressive description languages.** In general, more expressive description languages can be obtained by extending $\mathcal{ALC}$ with additional concept and role constructors, as well as with axioms for the roles.

First of all, we will briefly describe some commonly used concept constructors, besides those already contained in $\mathcal{ALC}$.

• Number restrictions ($\mathcal{N}$). The number of fillers of a role r is restricted to be smaller or equal (at most restrictions), or bigger or equal (at least restriction) than a number n. The interpretation of such concepts is:

$$\text{At most restriction:} \quad (\leq_n r)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \|\{b \mid (a,b) \in r^{\mathcal{I}}\}\| \leq n\}$$
$$\text{At least restriction:} \quad (\geq_n r)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \|\{b \mid (a,b) \in r^{\mathcal{I}}\}\| \geq n\}$$

where $\|S\|$ denotes the size of set S.

• Quantified number restrictions ($\mathcal{Q}$). They extend number restrictions in the sense that the number of fillers of r *that belong to the interpretation of* C is restricted by n. The interpretation of concepts using quantified number restrictions is:

$$\text{At most restriction:} \quad (\leq_n r.\mathsf{C})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \|\{b \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in \mathsf{C}^{\mathcal{I}}\}\| \leq n\}$$
$$\text{At least restriction:} \quad (\geq_n r.\mathsf{C})^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} \mid \|\{b \mid (a,b) \in r^{\mathcal{I}} \text{ and } b \in \mathsf{C}^{\mathcal{I}}\}\| \geq n\}$$

Role constructors are used in order to create complex roles. Remember that $\mathcal{ALC}$ contains only atomic roles (denoted by r). We will use the symbol R for complex (not atomic) roles. Some common role constructors are the following:

• Inverse roles ($\mathcal{I}$). Given an $\mathcal{ALC}$-role R, we use the notation R^- to denote the inverse role of R. The interpretation of an inverse role is:

$$(R^-)^{\mathcal{I}} = \{(b,a) \mid (a,b) \in R^{\mathcal{I}}\}.$$

• Role intersection ($\cap$). Given two $\mathcal{ALC}$-roles R_1 and R_2 the intersection of the roles $(R_1 \cap R_2)$ is a new role interpreted as follows:

$$(R_1 \cap R_2)^{\mathcal{I}} = R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}.$$

• Role chain (composition) ($\circ$). Let R_1 and R_2 be two $\mathcal{ALC}$-roles. Their composition $(R_1 \circ R_2)$ is a role defined as follows:

$$(R_1 \circ R_2)^{\mathcal{I}} = \{(a,b) \mid \text{ there exists } c \in \Delta^{\mathcal{I}} \text{ such that } (a,c) \in R_1^{\mathcal{I}} \text{ and } (c,b) \in R_2^{\mathcal{I}}\}$$

• Role complement ($\neg$). Like the complement of a concept C is a concept $\neg\mathsf{C}$ whose interpretation contains the domain elements that do not belong to the interpretation of C, the complement of a role R is another role that is interpreted as all the pairs of elements that do not belong to $R^{\mathcal{I}}$.

$$(\neg R)^{\mathcal{I}} = (\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}) \backslash R^{\mathcal{I}}$$

Finally, it is possible to have role axioms which are used to assert that certain roles have some properties. The most common role axioms are the following:

• Transitive roles ($\mathcal{S}$). An axiom of the form $\mathit{Trans}(R)$ is used to assert that R is a transitive role.

• Role hierarchy ($\mathcal{H}$). An axiom of the form $R_1 \dot{\sqsubseteq}_R R_2$ is used to denote that $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.

Notice that the role axioms resemble the axioms of the TBox in the sense that they describe properties of the roles, like TBox contains statements regarding properties of the concepts. This is why some authors introduce a new component to the knowledge base, the RBox, that contains all the role axioms. In this case, a knowledge base is a triple $\mathcal{K} = \langle \mathcal{T}, \mathcal{A}, \mathcal{R} \rangle$. A more detailed description of RBoxes can be found in [19].

As a convention, description languages are named by adding the names of the extra constructors and role axioms they contain to the end of "$\mathcal{ALC}$". For example, the description language that extends $\mathcal{ALC}$ with number restrictions is $\mathcal{ALCN}$. However, $\mathcal{ALC}$ extended with transitive roles is widely known as $\mathcal{S}$. Therefore, the language $\mathcal{SHIN}$ is the language $\mathcal{ALC}$ with transitive and inverse roles, role hierarchy and number restrictions.

6. A CLDS for the Description Language $\mathcal{ALC}$

In this section we will describe a Compiled Labelled Deductive System for the description language $\mathcal{ALC}$. Obviously, in this case $\mathcal{L}_{\#}$ is $\mathcal{ALC}$. The labelling language is a subset of the language given in definition 30. To be more specific, it is a first-order language that contains a set of constant symbols, a set of individual variables, and a set of predicate symbols, along with the logical connectives and quantifiers. It contains no function symbols. From now on we will use the symbol $\mathcal{L}_L^{\mathcal{ALC}}$ for this labelling language.

Definition 48 (Labelling language $\mathcal{L}_L^{\mathcal{ALC}}$)**.** Fix a language $\mathcal{ALC}\langle\mathcal{N}_C, \mathcal{N}_R, \mathcal{N}_I\rangle$. The labelling language of the CLDS for $\mathcal{ALC}\langle\mathcal{N}_C, \mathcal{N}_R, \mathcal{N}_I\rangle$ is the first-order language $\mathcal{L}\langle\mathcal{P}, \mathcal{F}, \mathcal{C}\rangle$ where:

- $\mathcal{P}$ is a set of binary predicate symbols. Each one of these predicates will be the "equivalent" of an $\mathcal{ALC}$-role. Therefore, the labelling language needs to contain as many such predicates as the cardinality of $\mathcal{N}_R$. In order to highlight this correspondence, we will use the notation R_r for the binary predicate of the labelling algebra that corresponds to the $\mathcal{ALC}$-role r.
- $\mathcal{F} = \emptyset$: The labelling language contains no function symbols.
- $\mathcal{C}$ is a set of constants. We consider the set of constants to consist of two disjoint sets of symbols: $\mathcal{C}_{ind}$ that may be empty, and the non-empty set $\mathcal{C}_{gen}$ that is considered to contain countably infinite elements ($\mathcal{C} = \mathcal{C}_{ind} \cup \mathcal{C}_{gen}$). Every constant of $\mathcal{C}_{ind}$ corresponds to an $\mathcal{ALC}$-individual. We will use the same notation as in the case of predicate symbols: c_a will denote the constant of the labelling language that corresponds to the individual a. Obviously, $|\mathcal{C}_{ind}| = |\mathcal{N}_I|$. Therefore, if the ABox is empty, $\mathcal{C}_{ind}$ will be the empty set. Regarding the set $\mathcal{C}_{gen}$, we will use the notation c_0, c_1,... for its elements.

Therefore, the CLDS-language in this case is the ordered pair $\langle\mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}}\rangle$.

Example 49. Some examples of CLDS formulas for the language $\langle\mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}}\rangle$ are given below:

$$(\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_a \qquad \text{declarative unit}$$
$$R_r(c_a, c_b) \qquad \text{Relation-literal}$$

Generally, the semi-extended labelling language is obtained by extending the labelling language with appropriate skolem function symbols. In the case of the $\mathcal{ALC}$-CLDS, however, the semi-extended labelling language is assumed to coincide with the labelling language.

The labelling algebra $\mathcal{A}$ is a theory written in the semi-extended labelling language of the CLDS Func($\mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}}$). It is used to capture some properties of the $\mathcal{ALC}$-roles.

Example 50. In this example, we will present some common axioms of the labelling algebra for an $\mathcal{ALC}$-CLDS.

- Reflexive roles: Let *likes* be a role that associates a person with another person that he/she likes. For example $likes(MARY, ANN)$ means that Mary likes Ann. Assuming that every person likes himself/herself, the role *likes* needs to be reflexive. That can be expressed by adding in the labelling algebra the following axiom:

$$\forall x(likes(x, x))^{3}$$

TABLE 1. Axioms of $\text{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$ for the $\mathcal{ALC}$-CLDS

Name	Axiom
Ax-$\sqcap$	$\forall x\left([\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(x) \rightarrow ([\mathsf{C}_1]^*(x) \wedge [\mathsf{C}_2]^*(x))\right)$
Ax-$\sqcup$	$\forall x\left([\mathsf{C}_1 \sqcup \mathsf{C}_2]^*(x) \rightarrow ([\mathsf{C}_1]^*(x) \vee [\mathsf{C}_2]^*(x))\right)$
Ax-$\neg$	$\forall x\left([\neg\mathsf{C}]^*(x) \leftrightarrow \neg[\mathsf{C}]^*(x)\right)$
Ax-$\forall$	$\forall x\left([\forall r.\mathsf{C}]^*(x) \rightarrow \forall y\left(R_r(x,y) \rightarrow [\mathsf{C}]^*(y)\right)\right)$
Ax-$\exists$	$\forall x\left([\exists r.\mathsf{C}]^*(x) \rightarrow \exists y\left(R_r(x,y) \wedge [\mathsf{C}]^*(y)\right)\right)$

• Irreflexive roles: Let *hasMother* be a role that associates a person with his/her mother. It is a fact that no one can be his/her own mother, therefore this role has to be irreflexive. That can be expressed with the following axiom of the labelling algebra:

$$\forall x \neg \mathit{hasMother}(x, x).$$

• Transitive roles: Consider for example the role *isYounger*. The assertion $\mathit{isYounger}(\mathit{MARY}, \mathit{ANN})$ means that Mary is younger than Ann. Obviously this is a transitive relation: if Mary is younger than Ann and Ann is younger than John, then Mary is younger than John.

$$\forall x_0 \forall x_1 \forall x_2 (\mathit{isYounger}(x_0, x_1) \wedge \mathit{isYounger}(x_1, x_2) \rightarrow \mathit{isYounger}(x_0, x_2))$$

• Serial roles: Consider the role *hasMother* that associates a person with his/her mother. Since every person has a mother, *hasMother* is a serial role. This can be expressed by adding the following axiom in the labelling algebra:

$$\forall x_0 \exists x_1 (\mathit{hasMother}(x_0, x_1))$$

It is worth mentioning that this can be expressed in $\mathcal{ALC}$ by the following axiom in the TBox:

$$\top \dot{\sqsubseteq} \exists \mathit{hasMother}.\top$$

Notice that the labelling algebra contains information regarding the $\mathcal{ALC}$-roles. Information for the concepts is expressed in the TBox.

The extended labelling algebra $\mathcal{A}^+$ is a theory written in $\text{Mon}(\mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}})$ that extends $\mathcal{A}$. Its role is to "capture" the translation of $\mathcal{ALC}$ into first-order logic. The axioms of $\mathcal{A}^+$ for the $\mathcal{ALC}$-CLDS are presented in table 1. Notice that most of the axioms are left to right implications and not equivalences. This is mainly due to the reason that, as we will see later, tableau rules "deconstruct" a formula. During the construction of the tableau we will never need to "build" the conjunction of two formulas. The procedure works the other way around. The conjunction of two formulas will be "broken down" to its containing formulas.

[3]According to the syntax rules of first-order logic as presented earlier in this work, the correct syntax for this axiom would be: $\forall x \mathit{likes}(x, x)$. The parentheses have been added only to improve readability.

7. Tableau for $\mathcal{ALC}$-CLDS

7.1. **General Introduction.** In a few words, tableaux methods are formal proof procedures. There exist tableau methods for several logics, including propositional and first-order logic, modal logics, hybrid logics, and description logics. An introduction to the basic notions of tableaux methods as well as tableaux methods for specific logics can be found in [14].

Generally, a tableau method begins with a formula (or a set of formulas). Then the tableau is expanded using appropriate tableau rules until a closing condition is met. Usually (but not always) the expansion of the tableau consists of "breaking down" the initial formula into its subformulas.

7.2. **Preliminaries on tableaux for CLDS.** A CLDS tableaux will be represented as a labelled binary tree, whose labels are CLDS-formulas.

Definition 51 (CLDS-tableau)**.** A CLDS-tableau is a CLDS-labelled tree $\mathcal{T}_C = \langle \mathcal{T}, f \rangle$, where $\mathcal{T}$ is an unlabelled *binary* tree.

At this point it has to be made clear that the above definition presents the necessary but not the sufficient conditions under which a tree is a CLDS-tableau. In other words, every CLDS-tableau is a tree of the form described in definition 51, but not every such tree is a CLDS-tableau. A tree that corresponds to a CLDS-tableau must fulfil some additional conditions that will be explained in the following sections.

Since tableaux are defined as labelled trees, a branch of a tableau is in fact a labelled branch. In what follows, we will drop the qualification "labelled" for branches, when no confusion arises. If the branch refers to a labelled tree (a tableau to be more specific), it is a labelled branch.

Definition 52 (Closed branch)**.** Let $\mathcal{T}_C$ be a CLDS-tableau. A branch B of $\mathcal{T}_C$ is *closed* if and only if one of the following holds:

(1) $(a : \lambda) \in B$ and $(\neg a : \lambda) \in B$ for some declarative unit $(a : \lambda)$.
(2) $\Delta \in B$ and $\overline{\Delta} \in B$ for some Relation-literal Δ.

Equivalently, a tableau branch is said to be *open* if and only if it is not closed.

A CLDS-tableau is closed iff every branch of the tableau is closed. Similarly, a CLDS-tableau is open iff it has at least one open branch.

It has to be noted that in description logics the definition of a closed branch does not require the second condition of definition 52. That is because the ABox contains only role assertions, namely it does not contain negated Relation-literals.

It is easy to establish a correspondence between a CLDS-configuration and a branch of a CLDS-tableau. Recall that every branch of a CLDS-tableau is labelled with a set of CLDS-formulas, and a configuration can be defined as a set of CLDS-formulas. Therefore, the set of labels that appear on a branch can define a configuration, and a configuration can be represented as a branch of a CLDS-tableau.

Definition 53 (Corresponding configuration of a branch)**.** Let $\mathcal{T}_C$ be a CLDS-tableau, and B a branch of $\mathcal{T}_C$. The *corresponding configuration* of B is the configuration that contains exactly the CLDS-formulas that appear on the branch:

$$\mathcal{C}_{co}(B) = \{\phi \mid \phi \in B\}$$

Definition 54 (Corresponding branch of a configuration)**.** Let $\mathcal{C}$ be a CLDS-configuration. The *corresponding branch* of $\mathcal{C}$ is a branch with as many nodes as the number of formulas in $\mathcal{C}$ and whose nodes are labelled with exactly the CLDS-formulas of the configuration:

$$B_{co}(\mathcal{C}) = \langle b, f \rangle$$

where b is an unlabelled branch with as many nodes as the number of formulas contained in the configuration: $|b| = |\mathcal{C}|$, and f is a bijective function that maps every node of the branch to a formula that belongs in configuration $\mathcal{C}$ ($f : b \to \mathcal{C}$).

Proposition 55. Let $\mathcal{C}$ be a CLDS-configuration and B a branch of a CLDS-tableau. If ϕ is a CLDS-formula, the following hold:

$$\phi \in B_{co}(\mathcal{C}) \text{ iff } \phi \in \mathcal{C} \qquad \text{and} \qquad \phi \in \mathcal{C}_{co}(B) \text{ iff } \phi \in B$$

Proof. Directly from the way $B_{co}(\mathcal{C})$ and $\mathcal{C}_{co}(B)$ are defined. □

Proposition 56. Let $\mathcal{C}$ be a CLDS-configuration and $B_{co}(\mathcal{C})$ the corresponding branch. The configuration that corresponds to $B_{co}(\mathcal{C})$ coincides with $\mathcal{C}$:

$$\mathcal{C}_{co}(B_{co}(\mathcal{C})) = \mathcal{C}$$

Similarly, let B be a branch of a CLDS-tableau, and $\mathcal{C}_{co}(B)$ the corresponding configuration. The branch that corresponds to $\mathcal{C}(B)$ is identical to B:

$$B_{co}(\mathcal{C}_{co}(B)) = B$$

Proof. We have to show that $\mathcal{C}_{co}(B_{co}(\mathcal{C}))$ contains the same CLDS-formulas as $\mathcal{C}$. From proposition 55: $\phi \in \mathcal{C}_{co}(B_{co}(\mathcal{C}))$ iff $\phi \in B_{co}(\mathcal{C})$ iff $\phi \in \mathcal{C}$.

Similarly, we have to show that $B_{co}(\mathcal{C}_{co}(B))$ contains the same labels as B. From proposition 55: $\phi \in B_{co}(\mathcal{C}_{co}(B))$ iff $\phi \in \mathcal{C}_{co}(B)$ iff $\phi \in B$. □

Lemma 57. Let $\mathcal{C}$ be a CLDS-configuration and $B(\mathcal{C})$ the corresponding branch. If $\mathcal{C}$ is satisfiable, then $B_{co}(\mathcal{C})$ is open.

Proof. This proof makes use of the axioms of the extended labelling algebra. In the case of $\mathcal{ALC}$-CLDS these axioms are presented in table 1. To be more specific, only axiom Ax-$\neg$ is used. This lemma can be proved for any CLDS system whose extended labelling algebra contains the following (more general) form of axiom Ax-$\neg$:

$$\text{Ax-}\neg\text{(gen)} \quad \forall x \left([\neg a]^* (x) \leftrightarrow (\neg [a]^* (x)) \right)$$

We will prove this lemma for a general CLDS whose labelling algebra contains Ax-$\neg$(gen).

Assume $\mathcal{C}$ is satisfiable. We have to show that $B(\mathcal{C})$ is open. Since $\mathcal{C}$ is satisfiable, there exists a semantic structure $\mathcal{M} = \langle D, I \rangle$ such that $\mathcal{M}$ satisfies all CLDS-formulas that belong to $\mathcal{C}$ (definition 44). Assume $B(\mathcal{C})$ is not open. That means that it is closed therefore, one of the following holds:

- There exist a formula $a \in \mathcal{L}_{\#}$ and a label λ such that $(a : \lambda) \in B(\mathcal{C})$ and $(\neg a : \lambda) \in B(\mathcal{C})$. Hence (from proposition 55)

  $$(a : \lambda) \in \mathcal{C} \text{ and } (\neg a : \lambda) \in \mathcal{C}.$$

  $\mathcal{M}$ satisfies all the formulas that belong to configuration $\mathcal{C}$, thus

  $$\mathcal{M} \models_{CLDS} (a : \lambda) \text{ and } \mathcal{M} \models_{CLDS} (\neg a : \lambda).$$

or equivalently $\mathcal{M} \models_{FOL} [a]^*(\lambda)$ and $\mathcal{M} \models_{FOL} [\neg a]^*(\lambda)$. It follows that

$$\lambda^I \in [[a]^*]^I \text{ and } \lambda^I \in [[\neg a]^*]^I.$$

However, $\mathcal{M}$ satisfies Ax-¬(gen) of the extended labelling algebra:

$$\mathcal{M} \models_{FOL} \forall x \left([\neg a]^* (x) \leftrightarrow \neg [a]^* (x)\right)$$

which is equivalent to

$$\mathcal{M}, [x \mapsto d] \models_{FOL} \left([\neg a]^* (x) \leftrightarrow \neg [a]^* (x)\right) \text{ for all } d \in D.$$

Since $\lambda^I \in D$, it follows that $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} ([\neg a]^*(x) \rightarrow \neg [a]^*(x))$. That means that if $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} [\neg a]^*(x)$, then it is the case that $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} \neg [a]^*(x)$. However, $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} [\neg a]^*(x)$ is equivalent to $\lambda^I \in [[\neg a]^*]^I$ which we have shown that it holds. Therefore, $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} \neg [a]^*(x)$. or equivalently $\lambda^I \notin [[a]^*]^I$. This is a contradiction since we have shown that $\lambda^I \in [[a]^*]^I$.

- There exists a Relation-literal such that $R_r(\lambda, \lambda') \in B(\mathcal{C})$ and $\neg R_r(\lambda, \lambda') \in B(\mathcal{C})$. Equivalently (from proposition 55)

$$R_r(\lambda, \lambda') \in \mathcal{C} \text{ and } \neg R_r(\lambda, \lambda') \in \mathcal{C}.$$

$\mathcal{M}$ satisfies all the formulas that belong to $\mathcal{C}$, therefore

$$\mathcal{M} \models_{CLDS} R_r(\lambda, \lambda') \text{ and } \mathcal{M} \models_{CLDS} \neg R_r(\lambda, \lambda')$$

which is equivalent to

$$\mathcal{M} \models_{FOL} R_r(\lambda, \lambda') \text{ and } \mathcal{M} \models_{FOL} \neg R_r(\lambda, \lambda').$$

or equivalently $\mathcal{M} \models_{FOL} (R_r(\lambda, \lambda') \wedge \neg R_r(\lambda, \lambda')) \Leftrightarrow \mathcal{M} \models_{FOL} \bot$. That is again impossible.

Therefore, $B_{co}(\mathcal{C})$ is open. □

7.3. **Tableau Expansion Rules.** To begin with, it is assumed that all $\mathcal{ALC}$-concepts (that appear in the declarative units) are in negation normal form (NNF). That means that negation (¬) appears only in front of atomic concepts (and atomic roles, however, since we only deal with atomic roles this case will not arise here). Every $\mathcal{ALC}$-concept can be transformed to an equivalent concept in NNF by using De Morgan's rules

$$\neg(\mathsf{C}_1 \sqcap \mathsf{C}_2) \equiv (\neg\mathsf{C}_1 \sqcup \neg\mathsf{C}_2) \qquad \neg(\mathsf{C}_1 \sqcup \mathsf{C}_2) \equiv (\neg\mathsf{C}_1 \sqcap \neg\mathsf{C}_2)$$

and the following well-known relationships between ∀ and ∃:

$$\neg(\forall r.\mathsf{C}) \equiv \exists r.\neg\mathsf{C} \qquad \neg(\exists r.\mathsf{C}) \equiv \forall r.\neg\mathsf{C}$$

This transformation can be done in linear time [15], therefore it does not affect the complexity of the reasoning algorithms for description logics (which is not smaller than P-time [3]). This assumption reduces the number of the tableau expansion rules, since there is no need to have rules that handle the negation.

Example 58. Consider for example the declarative unit $(\neg\forall r_1.\exists r_2(\mathsf{A}_1 \sqcap \mathsf{A}_2) : \lambda)$. The $\mathcal{ALC}$-part of this formula has to be transformed into NNF:

$$\begin{aligned} \neg\forall r_1.\exists r_2.(\mathsf{A}_1 \sqcap \mathsf{A}_2) &\equiv \exists r_1.\neg\exists r_2.(\mathsf{A}_1 \sqcap \mathsf{A}_2) \\ &\equiv \exists r_1.\forall r_2.\neg(\mathsf{A}_1 \sqcap \mathsf{A}_2) \\ &\equiv \exists r_1.\forall r_2.(\neg\mathsf{A}_1 \sqcup \neg\mathsf{A}_2) \end{aligned}$$

Therefore, the CLDS-formula that will be used is $(\exists r_1.\forall r_2.(\neg \mathsf{A}_1 \sqcup \neg \mathsf{A}_2) : \lambda)$.

We will now present the basic the basic expansion rules of an $\mathcal{ALC}$-CLDS system. Let $\mathcal{T}_{AC} = \langle \mathcal{T}, f \rangle$ be an $\mathcal{ALC}$-CLDS tableau, $B = \langle b_l, f \rangle$ an open branch of $\mathcal{T}_{AC}$ (remember that l is the leaf node of b_l).

- The $\sqcap$-rule:
  Informally, if the CLDS-formula $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda)$ appears in the branch B, and not both $(\mathsf{C}_1 : \lambda)$ and $(\mathsf{C}_2 : \lambda)$ appear in B, extend the branch with whichever of those two formulas is missing, or with both of them if none appears on the branch.
  More formally, if $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda) \in B$ then:
  – If $(\mathsf{C}_1 : \lambda) \notin B$ and $(\mathsf{C}_2 : \lambda) \notin B$ then add two nodes to the end of B: $l0$ labelled $(\mathsf{C}_1 : \lambda)$ and $l00$ labelled $(\mathsf{C}_2 : \lambda)$.
  – If $(\mathsf{C}_1 : \lambda) \in B$ and $(\mathsf{C}_2 : \lambda) \notin B$ then add a node to the end of B: $l0$ labelled $(\mathsf{C}_2 : \lambda)$.
  – If $(\mathsf{C}_1 : \lambda) \notin B$ and $(\mathsf{C}_2 : \lambda) \in B$ then add a node to the end of B: $l0$ labelled $(\mathsf{C}_1 : \lambda)$.
  This rule is represented as follows:

$$\frac{\langle n, ((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda) \rangle \ , \ n \in b_l}{\langle l0, (\mathsf{C}_1 : \lambda) \rangle \ , \ \langle l00, (\mathsf{C}_2 : \lambda) \rangle}$$

- The $\sqcup$-rule:
  Informally, if the CLDS-formula $((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda)$ appears in the branch B, and not both of the formulas $(\mathsf{C}_1 : \lambda)$ and $(\mathsf{C}_2 : \lambda)$ appears in B, extend B with whichever of those two formulas is missing, or if both of them are missing then "split" the branch and add one of them in each end of the branch.
  A more formal way to explain this rule is given below. If $((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda) \in B$ then:
  – If $(\mathsf{C}_1 : \lambda) \notin B$ and $(\mathsf{C}_2 : \lambda) \notin B$ then add two nodes to the end of B: $l0$ labelled $(\mathsf{C}_1 : \lambda)$ and $l1$ labelled $(\mathsf{C}_2 : \lambda)$.
  – If $(\mathsf{C}_1 : \lambda) \in B$ and $(\mathsf{C}_2 : \lambda) \notin B$ then add a node to the end of B: $l0$ labelled $(\mathsf{C}_2 : \lambda)$.
  – If $(\mathsf{C}_1 : \lambda) \notin B$ and $(\mathsf{C}_2 : \lambda) \in B$ then add a node to the end of B: $l0$ labelled $(\mathsf{C}_1 : \lambda)$.
  This rule is represented as follows:

$$\frac{\langle n, ((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda) \rangle \ , \ n \in b_l}{\langle l0, (\mathsf{C}_1 : \lambda) \rangle \mid \langle l1, (\mathsf{C}_2 : \lambda) \rangle}$$

- The $\forall$-rule:
  Informally, if the CLDS-formula $(\forall r.\mathsf{C} : \lambda_1)$ appears in the branch B, and the formula $R_r(\lambda_1, \lambda_2)$ also appears in the branch, then extend B with a node labelled $(\mathsf{C} : \lambda_2)$.
  Formally, if $(\forall r.\mathsf{C} : \lambda_1) \in b$ and $R_r(\lambda_1, \lambda_2) \in B$ then add a node to the end of B: $l0$ labelled $(\mathsf{C} : \lambda_2)$.
  This rule is represented as follows:

$$\frac{\langle n, (\forall r.\mathsf{C} : \lambda_1) \rangle \ , \ \langle n', R_r(\lambda_1, \lambda_2) \rangle \ , \ n, n' \in b_l}{\langle l0, (\mathsf{C} : \lambda_2) \rangle}$$

- The $\exists$-rule:

  Informally, if the CLDS-formula $(\exists r.\mathsf{C}:\lambda_1)$ appears in the branch B, then extend B with two nodes labelled $R_r(\lambda_1,\lambda_2)$ and $(\mathsf{C}:\lambda_2)$, where λ_2 is a CLDS label that does not appear already in the branch B.

  Formally, if $(\exists r.\mathsf{C}:\lambda_1) \in B$ and there exists no label λ such that $R_r(\lambda_1,\lambda) \in B$ and $(\mathsf{C}:\lambda) \in B$ then add two nodes to the end of B: $l0$ labelled $R_r(\lambda_1,\lambda_2)$ and $l00$ labelled $(\mathsf{C}:\lambda_2)$, where λ_2 is *new* in the branch B.

  This rule is represented as follows:

$$\frac{\langle n, (\exists r.\mathsf{C}:\lambda_1)\rangle \ ,\ n \in b_l}{\langle l0, R_r(\lambda_1,\lambda_2)\rangle \ ,\ \langle l00, (\mathsf{C}:\lambda_2)\rangle} \qquad (\lambda_2 \text{ is "new" to the branch})$$

The tableau expansion rules are the inference rules of the CLDS (remember that a CLDS is a tuple $\langle\langle \mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}}\rangle, \mathcal{A}, \mathcal{R}\rangle$). In what follows we will write $\mathcal{R} = \{(\forall), (\sqcap)\}$ for a set of inference rules that consists of the $\forall$-rule and the $\sqcap$-rule (the same holds of course for the rest of the rules).

Now some remarks regarding the $\exists$-rule. In the case of the $\mathcal{ALC}$-CLDS, the CLDS labels are just the constants of the labelling language, i.e. $\mathcal{C} = \mathcal{C}_{ind} \cup \mathcal{C}_{gen}$. The labels that can be added to the tableau by an application of the $\exists$-rule, are only the constants that belong to $\mathcal{C}_{gen}$. Remember that the constants of $\mathcal{C}_{ind}$ are "reserved" for individuals of the ABox. We will consider a well-founded strict total order on $\mathcal{C}_{gen}$, denoted by $\prec_{gen}$, and we will define an order $\prec$ on $\mathcal{C}$ as follows:

$$\prec = \prec_{gen} \cup \{(c_x, c_i) \mid c_x \in \mathcal{C}_{ind}, c_i \in \mathcal{C}_{gen}\}.$$

In other words, the elements of $\mathcal{C}_{ind}$ are considered to appear before the elements of $\mathcal{C}_{gen}$. It can be proved that $\prec$ is a well-founded order as well. The new labels introduced by the $\exists$-rule will be introduced according to this ordering. Taking this into consideration, the new label λ_2 introduced by the $\exists$-rule is $\lambda_1 \prec \lambda_2$. If $\lambda \prec \lambda'$ and $\lambda \neq \lambda'$ we say that λ' is a *successor* of λ. We say that a label λ' is a *direct successor* of λ if λ' is a successor of λ and for every label λ'' if λ'' is a successor of λ, then it is also a successor of λ'. It is not hard to show that every CLDS-label has exactly one direct successor.

Lemma 59. Let $\prec$ be a well-founded total ordering on a set S. Every element of S has at most one direct successor w.r.t. $\prec$.

Proof. Let $s \in S$ and assume that s has two (distinct) direct successors, say s_1 and s_2 ($s_1 \neq s_2$). It follows that $s \prec s_1$ and $s \prec s_2$. Since $\prec$ is a total order it is the case that either $s_1 \prec s_2$ or $s_2 \prec s_1$ (totality of $\prec$). Assume that $s_1 \prec s_2$. Notice that since s_2 is a direct successor of s and s_1 is a (direct) successor of s, it should hold that s_1 is a successor of s_2: $s_2 \prec s_1$. However, since $\prec$ is antisymmetric, $s_1 \prec s_2$ and $s_2 \prec s_1$ implies that $s_1 = s_2$. This is a contradiction. The proof in the case that $s_2 \prec s_1$ is similar. □

A CLDS-label c is a *direct R_r-successor* of the CLDS-label c' if there exists a Relation-predicate $R_r(c', c)$. More generally, c is said to be a *R_r-successor* of c' if it is a direct R_r-successor of c' or if there exists a CLDS-label c'' such that $R_r(c', c'')$ and c'' is an R_r-successor of c. Since there may exist more than one role symbols in $\mathcal{ALC}$ and consequently more than one Relation-predicate symbols in

the labelling language, a label may have R_{r_i}-successors for several values of i. We will use the term R-successor to refer to an R_{r_i}-successor for an arbitrary i.

Notice that the notion of a successor (w.r.t. $\prec$) is not the same as that of an R_r-successor. The $\prec$ relation describes the order with which labels are introduced to a branch the tableau, while the R_r-successors describe the structure of the diagram, i.e. how labels are related to each other.

The $\sqcup$-rule is said to be a *non-deterministic* rule, as it creates two branches. Similarly, the $\exists$-rule is a *generating* rule since it introduces a new label into the branch. What appears above the line in the representation of the tableau rules is the preconditions of the rule, i.e. the conditions under which a rule can be used.

We will say that a tableau rule is applied to a node n if $\langle n, \phi\rangle$ appears on the preconditions of the rule, where ϕ is an arbitrary CLDS-formula. According to this convention, the $\forall$-rule applies to the nodes n and n', while the rest of the rules apply to node n. Sometimes we will say that a tableau rule is applied to a branch and we will mean that it is applied to a node of the branch.

The application of an expansion rule to a tableau branch results in a new tableau which differs from the original one only in the extended branch. Notice that the definition of the rules makes sure that the resulting structure is prefix closed, and in cases where only one child has to be added to a node, this is the 0-child.

Definition 60 (Initial Tableau for a Configuration)**.** Given a configuration $\mathcal{C}$, the *initial tableau* for $\mathcal{C}$ is the corresponding branch $B_{co}(\mathcal{C}) = \langle b, f\rangle$, where b is the set of nodes $\{\epsilon, 0, 00, \ldots, 0^{|\mathcal{C}|-1}\}$.

What this definition says is that the initial tableau for a (finite) configuration $\mathcal{C}$ is a tree that consists of a single branch $b = \{\epsilon, 0, \ldots, 0^{|\mathcal{C}|-1}\}$ and is labelled with exactly the formulas of $\mathcal{C}$. Notice that b contains as many nodes as the formulas that appear in $\mathcal{C}$. The first node is ϵ, the root of the tree, the second is 0^1, the third is 0^2, and the $|\mathcal{C}|$-th is $0^{|\mathcal{C}|-1}$.

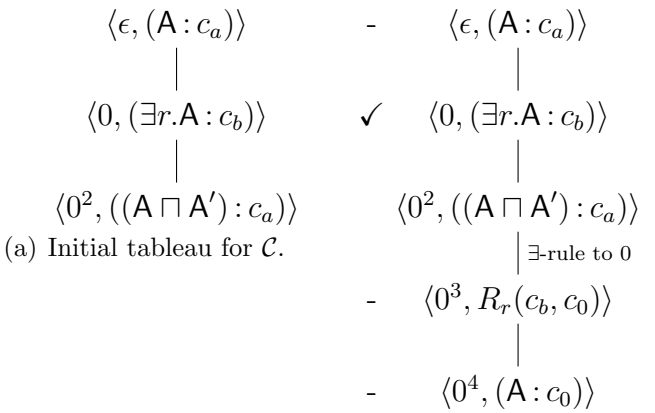
Starting from the initial tableau for a configuration we apply tableau expansion rules until none is applicable. The order with which the rules are applied is not important.

Definition 61 (Tableau for a configuration $\mathcal{C}$)**.** Let $\mathcal{C}$ be a finite configuration.

- The initial tableau for $\mathcal{C}$ (see definition 60) is a tableau for the configuration $\mathcal{C}$.
- If $\mathcal{T}_C$ is a tableau for $\mathcal{C}$ and $\mathcal{T}'_C$ results from $\mathcal{T}_C$ by applying one of the tableau expansion rules, then $\mathcal{T}'_C$ is a tableau for the configuration $\mathcal{C}$.

Remark 62. If $\mathcal{C}_0$ is a configuration and $\mathcal{T}_C$ is a CLDS-tableau for $\mathcal{C}_0$ (not necessarily the initial one) then every branch B of $\mathcal{T}_C$ contains the initial tableau for $\mathcal{C}_0$. Stated otherwise, $B_{co}(\mathcal{C}_0) \subseteq B$ for any branch B of $\mathcal{T}_C$. That means that the initial configuration $\mathcal{C}_0$ is contained in the corresponding configuration of B: $\mathcal{C}_0 \subseteq \mathcal{C}_{co}(B)$.

It has to be noted though, that technically $B_{co}(\mathcal{C}_0)$ is not a branch any more since the tableau has been expanded and the last node of $B_{co}(\mathcal{C}_0)$ is not a leaf node. However, we will use the same notation $B_{co}(\mathcal{C}_0)$ to refer to the path that corresponds to the initial configuration.



(a) Initial tableau for $\mathcal{C}$.

(b) Tableau for $\mathcal{C}$ after the application of the $\exists$-rule to node 0.

FIGURE 4. Tableaux for the configuration of example 63, $\mathcal{C} = \{(\mathsf{A} : c_a), (\exists r.\mathsf{A} : c_b), ((\mathsf{A} \sqcap \mathsf{A}') : c_a).\}$

Example 63. Consider for example the initial configuration

$$\mathcal{C} = \{(\mathsf{A} : c_a), (\exists r.\mathsf{A} : c_b), ((\mathsf{A} \sqcap \mathsf{A}') : c_a)\}.$$

The initial tableau for $\mathcal{C}$ will be the corresponding branch, i.e. a branch with three nodes each one labelled with a formula of $\mathcal{C}$. It is not important which node is labelled with which label. The initial tableau for $\mathcal{C}$ is presented in figure 4(a).

Assume that we apply the $\exists$-rule to node 0. Two nodes will be added to the end of the branch. The new label introduced by the rule will be a constant of $\mathcal{C}_{gen}$, c_0. The corresponding tableau, after the application of the $\exists$-rule, is shown in figure 4(b).

The symbols that appear on the left side of the nodes are not part of the tableau. We will use them as they make easier the application of the tableau rules. The symbol ✓ next to a node, means that the node has been expanded and no more tableau-rules can be applied to it. On the other hand, the symbol - denotes that the node is labelled with a formula such that no rules can be applied to it (i.e. with a Relation-predicate, or with a declarative unit with an atomic $\mathcal{ALC}$ concept). Moreover, the label "$\exists$-rule to 0" is not part of the tableau and it is used only in order to facilitate reading the tableau. This label means that nodes 0^3 and 0^4 result from the application of the $\exists$-rule to node 0.

8. An $\mathcal{ALC}$-CLDS for an Empty Knowledge Base

In the case of Description Logics, reasoning is done with respect to a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ that consists of an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$. We will first handle the simple case when the knowledge base is empty, i.e. both the ABox and the TBox contain no axioms.

In this case, the available reasoning tasks are concept satisfiability and concept subsumption (which can be reduced to concept unsatisfiability). We will now prove that checking if an $\mathcal{ALC}$ concept C is satisfiable w.r.t. the empty knowledge base is equivalent to checking whether the configuration $\{\mathsf{C} : \lambda\}$ is satisfiable, where $c \in \mathcal{C}$ is an $\mathcal{ALC}$-CLDS label. It has to be noted that in the proof of this correspondence, the set of inference rules of the CLDS plays absolutely no role.

Proposition 64 (Correspondence between concept satisfiability w.r.t the empty TBox in $\mathcal{ALC}$ and in $\mathcal{ALC}$-CLDS)**.** Let C be an $\mathcal{ALC}$ concept and fix an $\mathcal{ALC}$-CLDS system $\langle \langle \mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}} \rangle, \emptyset, \mathcal{R} \rangle$. C is satisfiable if and only if the configuration $\mathcal{C} = \{\mathsf{C} : \lambda\}$ is satisfiable.

Proof. ($\Rightarrow$) Assume the $\mathcal{ALC}$ concept C is satisfiable. That means that there exists an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ such that $\mathsf{C}^{\mathcal{I}} \neq \emptyset$. The main idea is to use the $\mathcal{ALC}$ interpretation $\mathcal{I}$ in order to construct a first-order structure $\mathcal{M}$ that satisfies the initial configuration and is a semantic structure of the CLDS (i.e. it satisfies the axioms of the extended labelling algebra).

Generally $\mathcal{M} = \langle D, I \rangle$. We define the domain of $\mathcal{M}$ to be the domain of the $\mathcal{ALC}$ interpretation:

$$D = \Delta^{\mathcal{I}}$$

and the interpretation function I to agree with $\cdot^{\mathcal{I}}$ on the interpretation of concepts (unary predicates), roles (binary predicates), and individuals (constants):

$$[[\mathsf{C}]^*]^I = \mathsf{C}^{\mathcal{I}} \qquad R_r^I = r^{\mathcal{I}} \qquad c_a^I = a^{\mathcal{I}}.$$

$\mathcal{M} \models_{CLDS} (\mathsf{C} : \lambda)$ iff $\mathcal{M} \models_{FOL} [\mathsf{C}]^*(\lambda)$ or equivalently, $\lambda^I \in [[\mathsf{C}]^*]^I$. Therefore, the initial configuration is satisfiable iff λ is interpreted as an element of the domain that belongs to the interpretation of the unary predicate $[\mathsf{C}]^*$. Such an element exists because we have defined $[[\mathsf{C}]^*]^I = \mathsf{C}^{\mathcal{I}}$ which we have assumed is non-empty. Hence I is defined to assign to constant λ an element of the domain that belongs to $[[\mathsf{C}]^*]^I$.

It remains to show that $\mathcal{M}$ is a semantic structure for the CLDS, i.e. that it satisfies the axioms of the extended labelling algebra (table 1).

- Ax-$\sqcap$:

$$\mathcal{M} \models_{FOL} \forall x \left([\mathsf{C}_1 \sqcap \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \wedge [\mathsf{C}_2]^* (x)) \right) \text{ iff}$$
$$\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \wedge [\mathsf{C}_2]^* (x)) \text{ for all } d \in D.$$

Assume that $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(x)$ for an arbitrary element d of the domain. We have to show that $\mathcal{M}, [x \mapsto d] \models_{FOL} ([\mathsf{C}_1]^*(x) \wedge [\mathsf{C}_2]^*(x))$ or equivalently, that $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_1]^*(x)$ and $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_2]^*(x)$.

By definition, $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(x)$ iff $d \in [[\mathsf{C}_1 \sqcap \mathsf{C}_2]^*]^I$. However, from the way we have defined I, $[[\mathsf{C}_1 \sqcap \mathsf{C}_2]^*]^I = [\mathsf{C}_1 \sqcap \mathsf{C}_2]^{\mathcal{I}} = \mathsf{C}_1^{\mathcal{I}} \cap \mathsf{C}_2^{\mathcal{I}}$. Therefore, $d \in \mathsf{C}_1^{\mathcal{I}} \cap \mathsf{C}_2^{\mathcal{I}}$ which is equivalent to $d \in \mathsf{C}_1^{\mathcal{I}}$ and $d \in \mathsf{C}_2^{\mathcal{I}}$. However $d \in \mathsf{C}_1^{\mathcal{I}}$ iff $d \in [[\mathsf{C}_1]^*]^I$ and $d \in \mathsf{C}_2^{\mathcal{I}}$ iff $d \in [[\mathsf{C}_2]^*]^I$. Because $d \in [\mathsf{C}_1]^*$ it follows that $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_1]^*(x)$ and similarly, $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_2]^*(x)$.

• Ax-$\sqcup$: The proof is similar to the one for Ax-$\sqcap$ and the details will be omitted.

• Ax-$\neg$:

$$\mathcal{M} \models_{FOL} \forall x \left([\neg \mathsf{C}]^* (x) \leftrightarrow \neg [\mathsf{C}]^* (x)\right) \text{ iff}$$
$$\mathcal{M}, [x \mapsto d] \models_{FOL} \left([\neg \mathsf{C}]^* (x) \leftrightarrow \neg [\mathsf{C}]^* (x)\right) \text{ for all elements } d \text{ of the domain.}$$

$\mathcal{M}, [x \mapsto d] \models_{FOL} [\neg \mathsf{C}]^*(x)$ iff $d \in [[\neg \mathsf{C}]^*]^I$ or equivalently $d \in [\neg \mathsf{C}]^{\mathcal{I}}$. That holds iff $d \notin \mathsf{C}^{\mathcal{I}}$ which is equivalent to $d \notin [[\mathsf{C}]^*]^I$. Equivalently, $\mathcal{M}, [x \mapsto d] \not\models_{FOL} [\mathsf{C}]^*(x)$.

• Ax-$\forall$:

$$\mathcal{M} \models_{FOL} \forall x \left([\forall r.\mathsf{C}]^* (x) \rightarrow \forall y \left([r]^* (x, y) \rightarrow [\mathsf{C}]^* (y)\right)\right) \text{ iff}$$
$$\mathcal{M}, [x \mapsto d] \models_{FOL} \left([\forall r.\mathsf{C}]^* (x) \rightarrow \forall y \left([r]^* (x, y) \rightarrow [\mathsf{C}]^* (y)\right)\right) \text{ for all } d \in D.$$

Assume $\mathcal{M}, [x \mapsto d] \models_{FOL} [\forall r.\mathsf{C}]^* (x)$ for an arbitrary element d of the domain. This holds if and only if $d \in [[\forall r.\mathsf{C}]^*]^I$ or equivalently $d \in [\forall r.\mathsf{C}]^{\mathcal{I}}$. This is equivalent to

$$d \in \{a \in \Delta^{\mathcal{I}} \mid \text{ for all } b \text{ such that } (a, b) \in r^{\mathcal{I}} \text{ implies } b \in r^{\mathcal{I}}\}. \tag{8.1}$$

We have to show that

$$\mathcal{M}, [x \mapsto d] \models_{FOL} \forall y \left([r]^* (x, y) \rightarrow [\mathsf{C}]^* (y)\right) \text{ or equivalently that}$$
$$\mathcal{M}, [x \mapsto d, y \mapsto d'] \models_{FOL} \left([r]^* (x, y) \rightarrow [\mathsf{C}]^* (y)\right) \text{ for all elements } d' \text{ of the domain.}$$

Assume $\mathcal{M}, [x \mapsto d, y \mapsto d'] \models_{FOL} [r]^* (x, y)$ for an arbitrary element d' of the domain. That means that $(d, d') \in [[r]^*]^I$ or equivalently that

$$(d, d') \in r^{\mathcal{I}}. \tag{8.2}$$

We now have to show that $\mathcal{M}, [x \mapsto d, y \mapsto d'] \models_{FOL} [\mathsf{C}]^*(y)$. That holds iff $d \in [[\mathsf{C}]^*]^I$ which is equivalent to $d \in \mathsf{C}^{\mathcal{I}}$. However, this holds from 8.1 and 8.2.

• Ax-$\exists$: The proof is similar to the one for axiom Ax-$\forall$.

($\Leftarrow$) Assume the configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$ is satisfiable. That means that there exists a semantic structure $\mathcal{M}$ that satisfies the axioms of the extended labelling algebra, and moreover $\mathcal{M} \models_{CLDS} (\mathsf{C} : \lambda)$ or equivalently $\mathcal{M} \models_{FOL} [\mathsf{C}]^*(\lambda)$. That holds iff $\lambda^I \in [[\mathsf{C}]^*]^I$. We have to show that the $\mathcal{ALC}$-concept is satisfiable i.e. that there exists and interpretation $\mathcal{I}$ such that $\mathsf{C}^{\mathcal{I}} \neq \emptyset$. We construct the $\mathcal{ALC}$ interpretation $\mathcal{I}$ in relation with $\mathcal{M}$ exactly as before:

$$\Delta^{\mathcal{I}} = D \qquad \mathsf{C}^{\mathcal{I}} = [[\mathsf{C}]^*]^I \qquad r^{\mathcal{I}} = R_r^I \qquad a^{\mathcal{I}} = c_a^I.$$

According to this construction $\mathsf{C}^{\mathcal{I}} = [[\mathsf{C}]^*]^I \neq \emptyset$ since we know that there exists an element of the domain $\lambda^I \in [[\mathsf{C}]^*]^I$. □

9. $\mathcal{ALC}$-CLDS Tableau for Reasoning under an Empty Knowledge Base

The first step is to present the simplest tableau calculus for $\mathcal{ALC}$-CLDS, i.e. in the case when the knowledge base is empty. In this case, since the ABox is empty, the initial configuration is the empty configuration: $\mathcal{C}_0 = \langle \emptyset, \emptyset \rangle$. Moreover, the labelling algebra is considered to be empty.

The only available reasoning tasks when the ABox is empty are concept satisfiability and concept subsumption. In order to check whether a concept C is satisfiable, the CLDS-formula $(\mathsf{C} : c_0)$, where $c_0 \in \mathcal{C}_{gen}$ is the minimal element of the set of labels w.r.t. $\prec_{gen}$, is added to the (previously empty) initial configuration: $\mathcal{C}'_0 = \langle \emptyset, \{(\mathsf{C} : c_0)\} \rangle$. If the tableau for $\mathcal{C}'_0$ closes, then C is not satisfiable, while if the complete tableau is open, C is satisfiable.

Subsumption can also be reduced to (un)satisfiability of a concept: a concept C is subsumed by D iff the concept $\mathsf{C} \sqcap \neg \mathsf{D}$ is unsatisfiable. In this case, the initial configuration will be $\mathcal{C}'_0 = \langle \emptyset, \{((\mathsf{C} \sqcap \neg \mathsf{D}) : c_0)\} \rangle$. If the tableau for $\mathcal{C}'_0$ closes, then $\mathsf{C} \sqcap \neg \mathsf{D}$ is not satisfiable, therefore C is subsumed by D. If the complete tableau for $\mathcal{C}'_0$ is open, then $\mathsf{C} \sqcap \neg \mathsf{D}$ is satisfiable and C is not subsumed by D.

Example 65. Suppose we want to check whether the $\mathcal{ALC}$-formula

$$\exists r_1.\exists r_2.(\mathsf{A}_1 \sqcap \neg \mathsf{A}_2) \sqcap \forall r_1.\forall r_2.(\neg \mathsf{A}_1 \sqcup \mathsf{A}_2)$$

is satisfiable w.r.t. the empty knowledge base. That corresponds to checking the satisfiability of the $\mathcal{ALC}$-CLDS formula

$$(\exists r_1.\exists r_2.(\mathsf{A}_1 \sqcap \neg \mathsf{A}_2) \sqcap \forall r_1.\forall r_2.(\neg \mathsf{A}_1 \sqcup \mathsf{A}_2) : c_0)$$

where c_0 is the minimal label w.r.t. $\prec_{gen}$. We start a tableau with the initial configuration $\mathcal{C} = \{(\exists r_1.\exists r_2.(\mathsf{A}_1 \sqcap \neg \mathsf{A}_2) \sqcap \forall r_1.\forall r_2.(\neg \mathsf{A}_1 \sqcup \mathsf{A}_2) : c_0)\}$. A complete tableau is presented in figure 5.

We will now give a detailed description of how this tableau was constructed. Node ϵ is the initial tableau for $\mathcal{C}$, i.e. it is labelled with the only formula of $\mathcal{C}$. Nodes 0^1 and 0^2 are derived by application of the $\sqcap$-rule to node ϵ. Nodes 0^3 and 0^4 result from node 0^2 by applying the $\exists$-rule. Nodes 0^2 and 0^4 trigger the application of the $\forall$-rule which results to node 0^5. Nodes 0^6 and 0^7 result from the application of the $\exists$-rule to node 0^4, and node 0^8 from the application of the $\forall$-rule to nodes 0^5 and 0^6. Finally, nodes 0^9 and 0^{10} come from the application of the $\sqcap$-rule to node 0^7, and nodes 0^{11} and $0^{10}1$ from the application of $\sqcup$-rule to node 0^8. At this point the branch whose leaf is node 0^{11} closes because of nodes 0^{11} and 0^9, while the branch with leaf node $0^{10}1$ closes because of nodes $0^{10}1$ and 0^{10}. Since the tableau closes, the initial formula is not satisfiable.

9.1. Soundness.

Soundness in the case of the tableau proof procedure means that if a formula X is satisfiable, then every (complete) tableau for X is open.

Lemma 66 (Tableau rules preserve satisfiability)**.** Let $\mathcal{C}$ be a satisfiable configuration and $\mathcal{T}_{AC} = B_{co}(\mathcal{C}) = \langle b, f \rangle$ the initial tableau for $\mathcal{C}$. If a tableau expansion rule can be applied to $\mathcal{T}_{AC}$, then the resulting tableau $\mathcal{T}'_{AC}$ will also be open.

Proof. We distinguish the following cases:

- The applied rule is $\sqcap$-rule:

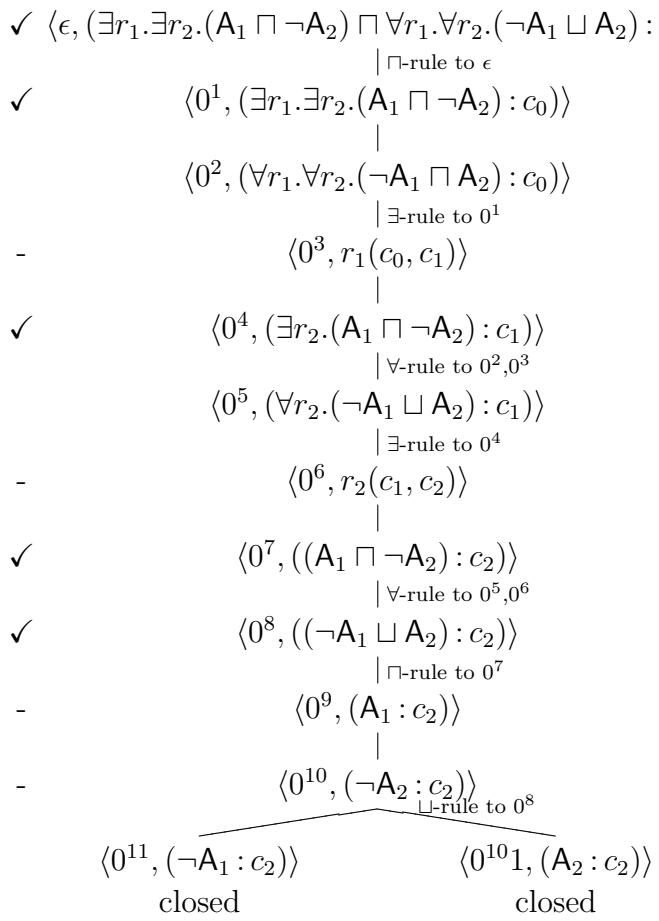


FIGURE 5. A tableau for checking the satisfiability of the CLDS-formula $(\exists r_1.\exists r_2.(\mathsf{A}_1 \sqcap \neg\mathsf{A}_2) \sqcap \forall r_1.\forall r_2.(\neg\mathsf{A}_1 \sqcup \mathsf{A}_2) : c_0)$.

That means that there exists a node of $B_{co}(\mathcal{C})$ that is labelled with a formula of the form $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0)$, otherwise, the $\sqcap$-rule would not be applied. From proposition 55 $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0) \in B_{co}(\mathcal{C})$ iff $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0) \in \mathcal{C}$, therefore $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0) \in \mathcal{C}$. $\mathcal{C}$ is satisfiable by assumption, thus there exists a semantic structure $\mathcal{M}$ that satisfies all formulas of $\mathcal{C}$. Since $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0) \in \mathcal{C}$, $\mathcal{M} \models_{CLDS} ((\mathsf{C}_1 \sqcap \mathsf{C}_2) : c_0)$. It follows that $\mathcal{M} \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(c_0)$.

$\mathcal{M}$ satisfies all the axioms of the extended labelling algebra, including axiom Ax-$\sqcap$:

$$\mathcal{M} \models_{FOL} \forall x \left([\mathsf{C}_1 \sqcap \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \wedge [\mathsf{C}_2]^* (x)) \right)$$

and therefore $\mathcal{M}, [x \mapsto d] \models_{FOL} \left([\mathsf{C}_1 \sqcap \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \wedge [\mathsf{C}_2]^* (x)) \right)$ for all elements d of the domain. It follows that for c_0^I, which is an element of the domain, it holds that $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\mathsf{C}_1]^*(x) \wedge [\mathsf{C}_2]^*(x)$ or equivalently

$$\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\mathsf{C}_1]^*(x) \text{ and } \mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\mathsf{C}_2]^*(x).$$

It follows that

$$\mathcal{M} \models_{FOL} [\mathsf{C}_1]^*(c_0) \text{ and } \mathcal{M} \models_{FOL} [\mathsf{C}_2]^*(c_0).$$

The resulting tableau $\mathcal{T}'_{AC}$ will consist of a single branch B' which is the branch $B_{co}(\mathcal{C})$ together with two new nodes (see tableau expansion rules), one labelled $(\mathsf{C}_1 : c_0)$, and one labelled $(\mathsf{C}_2 : c_0)$. The corresponding configuration is $\mathcal{C}_{co}(B') = \mathcal{C}' = \mathcal{C} \cup \{(\mathsf{C}_1 : c_0), (\mathsf{C}_2 : c_0)\}$. But $\mathcal{M}$ satisfies $\mathcal{C}'$ as well. Therefore, from lemma 57 we can conclude that since $\mathcal{C}'$ is satisfiable, the branch $B_{co}(\mathcal{C}')$ will be open. Since $B_{co}(\mathcal{C}')$ and B' contain exactly the same formulas (those of configuration $\mathcal{C}'$) and $B_{co}(\mathcal{C}')$ is open, B' will also be open. Hence $\mathcal{T}'_{AC}$ (which contains only branch B') will be open.

- If the applied rule is $\sqcup$-rule: It follows that there exists a node of $B_{co}(\mathcal{C})$ labelled with a formula of the form $(\mathsf{C}_1 \sqcup \mathsf{C}_2 : c_0)$ therefore, from proposition 55, $(\mathsf{C}_1 \sqcup \mathsf{C}_2 : c_0) \in \mathcal{C}$. Because $\mathcal{C}$ is satisfiable, it follows that there exists a semantic structure $\mathcal{M}$ that satisfies all the formulas of $\mathcal{C}$. That means that $\mathcal{M} \models_{CLDS} (\mathsf{C}_1 \sqcup \mathsf{C}_2 : c_0)$ or equivalently $\mathcal{M} \models_{FOL} [\mathsf{C}_1 \sqcup \mathsf{C}_2]^*(c_0)$.

  $\mathcal{M}$ satisfies Ax-$\sqcup$ of the extended labelling algebra:

$$\mathcal{M} \models_{FOL} \forall x \left([\mathsf{C}_1 \sqcup \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \vee [\mathsf{C}_2]^* (x)) \right)$$

  and therefore $\mathcal{M}, [x \mapsto d] \models_{FOL} \left([\mathsf{C}_1 \sqcup \mathsf{C}_2]^* (x) \rightarrow ([\mathsf{C}_1]^* (x) \vee [\mathsf{C}_2]^* (x)) \right)$ for all elements d of the domain. It follows that

$$\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\mathsf{C}_1]^*(x) \vee [\mathsf{C}_2]^*(x)$$

  which is equivalent to

$$\mathcal{M} \models_{FOL} [\mathsf{C}_1]^*(c_0) \text{ or } \mathcal{M} \models_{FOL} [\mathsf{C}_2]^*(c_0). \tag{9.1}$$

  The resulting tableau $\mathcal{T}'_{AC}$ will consist of two branches. The left one $(B_{left} = \langle b_{l0}, (\mathsf{C}_1 : c_0) \rangle)$ is the original branch extended with one node labelled $(\mathsf{C}_1 : c_0)$ and the right branch $(B_{right} = \langle b_{l1}, (\mathsf{C}_2 : c_0) \rangle)$ is the original one extended with a node labelled $(\mathsf{C}_2 : c_0)$. Regarding the corresponding configurations of the branches, obviously $\mathcal{C}_{co}(B_{left}) = \mathcal{C} \cup (\mathsf{C}_1 : c_0)$ and $\mathcal{C}_{co}(B_{right}) = \mathcal{C} \cup (\mathsf{C}_2 : c_0)$. From 9.1, it follows that $\mathcal{M} \models_{CLDS} \mathcal{C}_{co}(B_{left})$ or $\mathcal{M} \models_{CLDS} \mathcal{C}_{co}(B_{right})$ and consequently either B_{left} is open or B_{right} is open. In both cases, $\mathcal{T}'_{AC}$ is open since it has at least on open branch.
- If the applied rule is $\forall$-rule: It follows that there exists a node in $B_{co}(\mathcal{C})$ labelled with a formula of the form $(\forall r.\mathsf{C} : c_0)$ and a node labelled with a Relation-predicate $R_r(c_0, c')$ $(c' \in \mathcal{C})$. Therefore, $(\forall r.\mathsf{C} : c_0) \in \mathcal{C}$ and $R_r \in \mathcal{C}$. Since $\mathcal{C}$ is satisfiable, there exists a semantic structure $\mathcal{M}$ such that $\mathcal{M} \models_{CLDS} \mathcal{C}$ or equivalently $\mathcal{M} \models_{CLDS} (\forall r.\mathsf{C} : c_0)$. It follows that $\mathcal{M} \models_{FOL} [\forall r.\mathsf{C}]^*(c_0)$. Moreover, $\mathcal{M} \models_{FOL} R_r(c_0, c')$.

  $\mathcal{M}$ satisfies the axioms of the extended labelling algebra, thus it satisfies axiom Ax-$\forall$:

$$\mathcal{M} \models_{FOL} \forall x \left([\forall r.\mathsf{C}]^* (x) \rightarrow \forall y \left(R_r (x, y) \rightarrow [\mathsf{C}]^* (y) \right) \right)$$

  and therefore $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} \left([\forall r.\mathsf{C}]^* (x) \rightarrow \forall y \left(R_r (x, y) \rightarrow [\mathsf{C}]^* (y) \right) \right)$. Note that $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\forall r.\mathsf{C}]^* (x)$ is equivalent to $\mathcal{M} \models_{FOL} [\forall r.\mathsf{C}]^* (c_0)$ which holds. It follows that $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} \forall y \left(R_r (x, y) \rightarrow [\mathsf{C}]^*(y) \right)$ or equivalently $\mathcal{M}, [x \mapsto c_0^I, y \mapsto d] \models_{FOL} \left(R_r (x, y) \rightarrow [\mathsf{C}]^*(y) \right)$ for all elements d of the domain. Since c'^I is an element of the domain,

$$\mathcal{M}, [x \mapsto c_0^I, y \mapsto c'^I] \models_{FOL} \left(R_r (x, y) \rightarrow [\mathsf{C}]^*(y) \right).$$

This is equivalent to

$$\mathcal{M} \models_{FOL} R_r(c_0, c') \rightarrow [\mathsf{C}]^*(c').$$

Because $\mathcal{M} \models_{FOL} R_r(c_0, c')$ it follows that $\mathcal{M} \models_{FOL} [\mathsf{C}]^*(c')$ which is equivalent to $\mathcal{M} \models_{CLDS} (\mathsf{C} : c')$.

The application of the $\forall$-rule to $\mathcal{T}_{AC}$ results in a new tableau $\mathcal{T}'_{AC}$ which consists of the original branch (i.e. $\mathcal{T}_{AC}$) extended with one node labelled $(\mathsf{C} : c')$. The corresponding configuration of this branch is $\mathcal{C} \cup \{(\mathsf{C} : c')\}$. Obviously, $\mathcal{M}$ satisfies this configuration since it satisfies $\mathcal{C}$ and $(\mathsf{C} : c')$. Therefore, the corresponding branch (i.e. $\mathcal{T}_{AC}$) will be open.

- If the applied rule is $\exists$-rule: That means that there exists a node in $B_{co}(\mathcal{C})$ labelled $(\exists r.\mathsf{C} : c_0)$. Since $\mathcal{C}$ is satisfiable, there exists a semantic structure such that $\mathcal{M} \models_{CLDS} \mathcal{C}$. It follows that $\mathcal{M} \models_{FOL} [\exists r.\mathsf{C}]^*(c_0)$.

  Moreover, $\mathcal{M}$ satisfies axiom Ax-$\exists$ of the extended labelling algebra

  $$\mathcal{M} \models_{FOL} \forall x \left([\exists r.\mathsf{C}]^* (x) \rightarrow \exists y \left(R_r (x, y) \wedge [\mathsf{C}]^* (y)\right)\right)$$

  therefore $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} \left([\exists r.\mathsf{C}]^* (x) \rightarrow \exists y \left(R_r (x, y) \wedge [\mathsf{C}]^* (y)\right)\right)$.

  $\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} [\exists r.\mathsf{C}]^*(x)$ is equivalent to $\mathcal{M} \models_{FOL} [\exists r.\mathsf{C}]^*(c_0)$. It follows that

  $$\mathcal{M}, [x \mapsto c_0^I] \models_{FOL} \exists y \left(R_r (c_0, y) \wedge [\mathsf{C}]^* (y)\right)$$

  or equivalently $\mathcal{M}, [x \mapsto c_0^I, y \mapsto d] \models_{FOL} R_r(c_0, d) \wedge [\mathsf{C}]^*(d)$ for some element d of the domain. It follows that $\mathcal{M}, [x \mapsto c_0^I, y \mapsto d] \models_{FOL} R_r(x, y)$ and $\mathcal{M}, [x \mapsto c_0^I, y \mapsto d] \models_{FOL} [\mathsf{C}]^*(y)$ for some element d of the domain.

  The tableau that results from the application of the $\exists$-rule consists of the original branch extended with two nodes labelled $R_r(c_0, c')$ and $(\mathsf{C} : c')$, where λ' is a label that is new to the branch. The corresponding configuration of the new branch is $\mathcal{C} \cup \{R_r(c_0, c'), (\mathsf{C} : c')\}$. Since c' is new to the branch, it has not been already interpreted as an element of the domain. Therefore, we can define the interpretation of c' to be the element d of the domain that we have mentioned above. Thus $\mathcal{M}$ satisfies the new configuration and the corresponding branch is open. □

Theorem 67 (Soundness of the $\mathcal{ALC}$-CLDS tableau w.r.t. the empty knowledge base)**.** If $\mathcal{C}$ is a satisfiable configuration, every tableau for $\mathcal{C}$ which is open.

Proof. The main idea of the proof is to show that (i) the initial tableau of a satisfiable configuration is open, and (ii) that every application of a tableau rule to an open tableau, results in another open tableau.

(i) The fact that the initial tableau for a satisfiable configuration is open can be derived directly from the definition of the initial tableau (definition 60) and lemma 57.

(ii) Let $\mathcal{T}_{\mathcal{L}}$ be an open tableau for the configuration $\mathcal{C}$. We need to show that if we apply a tableau-expansion rule, the resulting tableau $\mathcal{T}'_{\mathcal{L}}$ will also be open.

Since $\mathcal{T}_{\mathcal{L}}$ is open, it has at least one open branch. Let b_{open} be an open branch of $\mathcal{T}_{\mathcal{L}}$.

- If an expansion rule is applied to a branch $b \neq b_{open}$, then obviously $\mathcal{T}'_{\mathcal{L}}$ will still be open.
- Assume that an expansion rule is applied to branch $b = b_{open}$. From lemma 66 it can be derived that the resulting branch will also be open. □

9.2. **Completeness.** The completeness of the tableau calculus can be summarized in the following sentence: If a complete tableau for a formula X does not close, then X is satisfiable.

Definition 68 (Downward saturated configuration)**.** Let $\mathcal{C}$ be a CLDS configuration. We say that $\mathcal{C}$ is a *downward saturated* set if the following hold:

- If $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda) \in \mathcal{C}$, then $(\mathsf{C}_1 : \lambda) \in \mathcal{C}$ and $(\mathsf{C}_2 : \lambda) \in \mathcal{C}$.
- If $((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda) \in \mathcal{C}$, then $(\mathsf{C}_1 : \lambda) \in \mathcal{C}$ or $(\mathsf{C}_2 : \lambda) \in \mathcal{C}$.
- If $((\forall r.\mathsf{C}) : \lambda_1) \in \mathcal{C}$ and $R_r(\lambda_1, \lambda_2) \in \mathcal{C}$, then $(\mathsf{C} : \lambda_2) \in \mathcal{C}$.
- If $((\exists r.\mathsf{C}) : \lambda_1) \in \mathcal{C}$, then there exists at least one CLDS-label λ_2 such that $R_r(\lambda_1, \lambda_2) \in \mathcal{C}$ and $(\mathsf{C} : \lambda_2) \in \mathcal{C}$.

Definition 69 ("Hintikka" configuration)**.** Let $\mathcal{C}$ be a CLDS configuration. We say that $\mathcal{C}$ is a Hintikka configuration if it is a downward saturated set, and moreover, it does not contain a formula and its negation.

Lemma 70. Every Hintikka configuration is satisfiable.

Proof. Let $\mathcal{C}$ be a Hintikka configuration. We show that it is satisfiable, i.e. that there exists a CLDS semantic structure such that $\mathcal{M} \models_{CLDS} \mathcal{C}$. $\mathcal{M}$ must satisfy the axioms of the extended algebra and the first-order translation of every formula in $\mathcal{C}$.

Notice that by definition every formula in $\mathrm{FOT}(\mathcal{C})$ is atomic and grounded thus $\mathrm{FOT}(\mathcal{C})$ defines a Herbrand interpretation:

$$[[\mathsf{C}]^*]^{\mathcal{H}} = \{d \mid d \in \mathcal{U} \text{ and } [\mathsf{C}]^*(d) \in \mathrm{FOT}(\mathcal{C})\}$$
$$R_r{}^{\mathcal{H}} = \{(d, d') \mid d, d' \in \mathcal{U} \text{ and } R_r(d, d') \in \mathrm{FOT}(\mathcal{C})\}$$

where $\mathcal{U}$ is the Herbrand universe of $\mathrm{FOT}(\mathcal{C})$. In other words, $\mathcal{M}_H \models_{FOL} [\mathsf{C}]^*(d)$ iff $d \in [[\mathsf{C}]^*]^{\mathcal{H}}$ which is equivalent to $[\mathsf{C}]^*(d) \in \mathrm{FOT}(\mathcal{C})$. In a similar way, $\mathcal{M}_H \models_{FOL} R_r(d, d')$ iff $R_r \in \mathrm{FOT}(\mathcal{C})$. Notice that $\mathcal{U}$ is the Herbrand universe of the configuration's translation. To be more specific, the only constants that appear in $\mathrm{FOT}(\mathcal{C})$ are the constants of the semi-labelling language that appear in $\mathcal{C}$, i.e. the labels that appear in the configuration. Since the semi-extended labelling language has no function symbols, it follows that $\mathcal{U}$ is exactly the set of labels that appear in $\mathcal{C}$.

Let $\mathcal{M}_H = \langle \mathcal{U}, \mathcal{H} \rangle$ be the Herbrand structure (for $\mathrm{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$) with $\mathcal{H}$ as defined above. Obviously it satisfies every formula of the configuration $\mathcal{C}$. We have to show that it is a semantic structure of the CLDS, i.e. that it satisfies the axioms of $\mathcal{A}^+$.

- Ax-$\sqcap$:

$$\mathcal{M}_H \models_{FOL} \forall x\, ([\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(x) \rightarrow [\mathsf{C}_1]^*(x) \wedge [\mathsf{C}_2]^*(x)) \text{ iff}$$
$$\mathcal{M}_H \models_{FOL} ([\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(d) \rightarrow [\mathsf{C}_1]^*(d) \wedge [\mathsf{C}_2]^*(d)) \text{ for all elements } d \in \mathcal{U}.$$

Assume $\mathcal{M}_H \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(d)$ for an arbitrary $d \in \mathcal{U}$. We have to show that $\mathcal{M}_H \models_{FOL} [\mathsf{C}_1]^*(d) \wedge [\mathsf{C}_2]^*(d)$ or equivalently that $\mathcal{M}_H \models_{FOL} [\mathsf{C}_1]^*(d)$ and $\mathcal{M}_H \models_{FOL} [\mathsf{C}_2]^*(d)$.

$\mathcal{M}_H \models_{FOL} [\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(d)$ iff $[\mathsf{C}_1 \sqcap \mathsf{C}_2]^*(d) \in \mathrm{FOT}(\mathcal{C})$. It follows that $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : d) \in \mathcal{C}$. That together with the fact that $\mathcal{C}$ is a Hintikka configuration implies that $(\mathsf{C}_1 : d) \in \mathcal{C}$ and $(\mathsf{C}_2 : d) \in \mathcal{C}$. Therefore, $[\mathsf{C}_1]^*(d) \in \mathrm{FOT}(\mathcal{C})$ and $[\mathsf{C}_2]^*(d) \in$

FOT($\mathcal{C}$) and from the way $\mathcal{H}$ has been defined it follows that $\mathcal{M}_H \models_{FOL} [\mathsf{C}_1]^*(d)$ and $\mathcal{M}_H \models_{FOL} [\mathsf{C}_2]^*(d)$.

It has to be noted that the Herbrand structure does not satisfy the converse of this axiom. This is because $\mathcal{C}$ is a Hintikka configuration and according to the definition if $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : d)$ belongs to $\mathcal{C}$, then both $(\mathsf{C}_1 : d)$ and $(\mathsf{C}_2 : d)$ belong to $\mathcal{C}$. However, if both $(\mathsf{C}_1 : d)$ and $(\mathsf{C}_2 : d)$ belong in $\mathcal{C}$, it does not follow that necessarily $(\mathsf{C}_1 \sqcap \mathsf{C}_2 : d)$ is in $\mathcal{C}$. This is the only step in the proof given above that does not "work in both ways".

- Ax-$\sqcup$: Similar to the proof for Ax-$\sqcap$.
- Ax-$\neg$:

$\mathcal{M}_H \models_{FOL} \forall x\, ([\neg \mathsf{C}]^*(x) \leftrightarrow \neg [\mathsf{C}]^*(x))$ iff

$\mathcal{M}_H \models_{FOL} ([\neg \mathsf{C}]^*(d) \leftrightarrow \neg [\mathsf{C}]^*(d))$ for all elements d of the Herbrand universe.

Assume that $\mathcal{M}_H \models_{FOL} [\neg \mathsf{C}]^*(d)$ for an arbitrary $d \in \mathcal{U}$. We have to show that $\mathcal{M}_H \models_{FOL} \neg [\mathsf{C}]^*(d)$.

$\mathcal{M}_H \models_{FOL} [\neg \mathsf{C}]^*(d)$ iff $[\neg \mathsf{C}]^*(d) \in$ FOT($\mathcal{C}$). That means that $(\neg \mathsf{C} : d) \in \mathcal{C}$ and since $\mathcal{C}$ is a Hintikka configuration, $(\mathsf{C} : d) \notin \mathcal{C}$ or equivalently $[\mathsf{C}]^*(d) \notin$ FOT($\mathcal{C}$). This is equivalent to $\mathcal{M}_H \models_{FOL} \neg [\mathsf{C}]^*(d)$.

Notice that in every step of the proof we move using equivalence, not implication. Therefore, both directions have been proved. The difference from the other axioms is that a formula belongs to a Hintikka configuration *if and only if* its negation does not.

- Ax-$\forall$:

$\mathcal{M}_H \models_{FOL} \forall x\, ([\forall r.\mathsf{C}]^*(x) \rightarrow \forall y\, (R_r(x, y) \rightarrow [\mathsf{C}]^*(y)))$ iff

$\mathcal{M}_H \models_{FOL} ([\forall r.\mathsf{C}]^*(d) \rightarrow \forall y\, (R_r(d, y) \rightarrow [\mathsf{C}]^*(y)))$ for all elements $d \in \mathcal{U}$.

Assume that $\mathcal{M}_H \models_{FOL} [\forall r.\mathsf{C}]^*(d)$ for an arbitrary $d \in \mathcal{U}$. That means that $[\forall r.\mathsf{C}]^*(d) \in$ FOT($\mathcal{C}$) or equivalently that

$$((\forall r.\mathsf{C}) : d) \in \mathcal{C}. \tag{9.2}$$

We have to show that $\mathcal{M}_H \models_{FOL} \forall y\, (R_r(d, y) \rightarrow [\mathsf{C}]^*(y))$ or equivalently that

$$\mathcal{M}_H \models_{FOL} (R_r(d, d') \rightarrow [\mathsf{C}]^*(d'))$$

for all elements d' of the domain. Assume that $\mathcal{M}_H \models_{FOL} R_r(d, d')$ for an arbitrary element d' of the domain. We have to show that $\mathcal{M}_H \models_{FOL} [\mathsf{C}]^*(d')$ or equivalently that $[\mathsf{C}]^*(d') \in$ FOT($\mathcal{C}$). $\mathcal{M}_H \models_{FOL} R_r(d, d')$ means that $(d, d') \in {R_r}^{\mathcal{H}}$. It follows that

$$R_r(d, d') \in \mathcal{C}. \tag{9.3}$$

From 9.2 and 9.3 together with the fact that $\mathcal{C}$ is a Hintikka configuration, it can be derived that $(\mathsf{C} : d') \in \mathcal{C}$. This means that $[\mathsf{C}]^*(d') \in$ FOT($\mathcal{C}$).

- Ax-$\exists$:

$\mathcal{M}_H \models_{FOL} \forall x\, ([\exists r.\mathsf{C}]^*(x) \rightarrow \exists y\, (R_r(x, y) \wedge [\mathsf{C}]^*(y)))$ iff

$\mathcal{M}_H \models_{FOL} ([\exists r.\mathsf{C}]^*(d) \rightarrow \exists y\, (R_r(d, y) \wedge [\mathsf{C}]^*(y)))$ for some element $d \in \mathcal{U}$.

Assume $\mathcal{M}_H \models_{FOL} [\exists r.\mathsf{C}]^*(d)$ for an element $d \in \mathcal{U}$. That means that $[\exists r.\mathsf{C}]^*(d) \in$ FOT($\mathcal{C}$) or equivalently that

$$((\exists r.\mathsf{C}) : d) \in \mathcal{C}. \tag{9.4}$$

We have to show that $\mathcal{M}_H \models_{FOL} \exists y\,(R_r\,(d, y) \wedge [\mathsf{C}]^*\,(y))$ or equivalently that

$$\mathcal{M}_H \models_{FOL} (R_r\,(d, d') \wedge [\mathsf{C}]^*\,(d'))$$

for some element d' of the domain. However, $\mathcal{C}$ is a Hintikka configuration therefore, because of 9.4, it follows that there exists a CLDS-label λ such that $R_r(d, \lambda) \in \mathcal{C}$ and $(\mathsf{C} : \lambda) \in \mathcal{C}$. This means that $R_r(d, \lambda) \in \mathrm{FOT}(\mathcal{C})$ and $[\mathsf{C}]^*(\lambda) \in \mathrm{FOT}(\mathcal{C})$. □

Definition 71 (Complete Branch)**.** A branch of a tableau is said to be *complete* when no tableau expansion rule can be applied to any node of the branch.

We say that a tableau is *complete* if all of its branches are complete.

Proposition 72. Every open and complete branch of an $\mathcal{ALC}$-CLDS tableau corresponds to a Hintikka configuration.

In other words, if B is an open and complete branch, then $\mathcal{C}_{co}(B)$ is a Hintikka configuration.

Proof. Let B be an open and complete branch of an $\mathcal{ALC}$-CLDS tableau and assume that $\mathcal{C}_{co}(B)$ is not a Hintikka configuration. We distinguish the following cases:

- $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda) \in \mathcal{C}_{co}(B)$ and $(\mathsf{C}_1 : \lambda) \notin \mathcal{C}(B)$ or $(\mathsf{C}_2 : \lambda) \notin \mathcal{C}(B)$. From proposition 55 it follows that $((\mathsf{C}_1 \sqcap \mathsf{C}_2) : \lambda) \in B$ and $(\mathsf{C}_1 : \lambda) \notin B$ or $(\mathsf{C}_2 : \lambda) \notin B$. That means that the $\sqcap$-rule could be applied, therefore the branch is not complete. Contradiction!
- $((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda) \in \mathcal{C}_{co}(B)$ and $(\mathsf{C}_1 : \lambda) \notin \mathcal{C}(B)$ and $(\mathsf{C}_2 : \lambda) \notin \mathcal{C}(B)$. Similarly to the previous case, from proposition 55 it follows that $((\mathsf{C}_1 \sqcup \mathsf{C}_2) : \lambda) \in B$ and $(\mathsf{C}_1 : \lambda) \notin B$ and $(\mathsf{C}_2 : \lambda) \notin B$. That means that the $\sqcup$-rule could be applied, therefore the branch is not complete. Contradiction!
- $((\forall r.\mathsf{C}) : \lambda_1) \in \mathcal{C}_{co}(B)$ and $r(\lambda_1, \lambda_2) \in \mathcal{C}_{co}(B)$, but $(\mathsf{C} : \lambda_2) \notin \mathcal{C}_{co}(B)$. Like in the previous cases, the $\forall$-rule could be applied to B and we reach a contradiction.
- $((\exists r.\mathsf{C}) : \lambda_1) \in \mathcal{C}_{co}(B)$, however there exists no label λ_2 such that $r(\lambda_1, \lambda_2) \notin \mathcal{C}_{co}(B)$ or $(\mathsf{C} : \lambda_2) \notin \mathcal{C}_{co}(B)$. In this case the $\exists$-rule could be applied. Contradiction!
- A formula and its negation appear in $\mathcal{C}B$. It follows that they would also appear in B and thus it would be closed. Contradiction!

Therefore $\mathcal{C}_{co}(B)$ is a Hintikka configuration. □

Theorem 73 (Completeness of $\mathcal{ALC}$-CLDS tableau w.r.t. the empty knowledge base)**.** If a configuration $\mathcal{C}$ is not satisfiable, then every complete tableau for $\mathcal{C}$ is closed.

Proof. Let $\mathcal{C}$ be an unsatisfiable configuration and $\mathcal{T}_{AC}$ a complete tableau for $\mathcal{C}$. Assume that $\mathcal{T}_{AC}$ is open. That means that there exists a branch of $\mathcal{T}_{AC}$, say B, that is open and complete. From proposition 72 it follows that the corresponding configuration $\mathcal{C}_{co}(B)$ is a Hintikka configuration, and therefore it is satisfiable (lemma 70). Configuration $\mathcal{C}$ is a subset of $\mathcal{C}_{co}(B)$ (see remark 62), therefore $\mathcal{C}$ is also satisfiable. Contradiction! □

9.3. **Termination.** It remains to show that the algorithm for the construction of an $\mathcal{ALC}$-CLDS tableau terminates. In other words, every complete $\mathcal{ALC}$-CLDS

tableau for a CLDS with an empty labelling algebra and when reasoning w.r.t. an empty knowledge base, is finite.

First of all, remember that $\mathcal{ALC}$ concepts are strings over the alphabet of $\mathcal{ALC}$ It is obvious that the number of atomic concept symbols and the number of atomic role symbols that appear in a concept C is bounded by the length of C.

We will now define the notion of subconcept. Informally, a subconcept of a given concept is a "smaller" concept contained in the given one. Subconcepts are the equivalent of subformulas. However, since $\mathcal{ALC}$-formulas are commonly referred to as concepts, the name subconcept is more appropriate. In order to give a formal definition for subconcepts, we first have to define the notion of substrings.

Definition 74 (Subconcept)**.** Let C be an $\mathcal{ALC}$ concept. We say that C' is a *subconcept* of C if it is an $\mathcal{ALC}$ concept and moreover, it is a substring of C.

Given an $\mathcal{ALC}$-concept C we will use the notation $sub(\mathsf{C})$ to denote the set of all the subconcepts of C:

$$sub(\mathsf{C}) = \{\mathsf{C}' \mid \mathsf{C}' \text{ is a subconcept of } \mathsf{C}\}.$$

Lemma 75. The number of subconcepts of an $\mathcal{ALC}$ concept is finite.

Proof. Only a sketch of the proof will be presented here since it is a quite common result. Fix an $\mathcal{ALC}$ concept C. The proof is by structural induction on C.

• (Base case) If C is an atomic concept then obviously, $sub(\mathsf{C}) = \{\mathsf{C}\}$ which is a finite set.

• Assume that the proposition holds for two arbitrary $\mathcal{ALC}$ concepts C_1 and C_2. If $\mathsf{C} = \neg\mathsf{C}_1$, then $sub(\mathsf{C}) = sub(\mathsf{C}_1) \cup \{\neg\mathsf{C}_1\}$ which is finite. Similarly, it can be proved that the set $sub(\mathsf{C})$ is finite if C is of the form $\mathsf{C}_1 \sqcap \mathsf{C}_2$, $\mathsf{C}_1 \sqcup \mathsf{C}_2$, $\exists r.\mathsf{C}_1$, or $\forall r.\mathsf{C}_1$. Details will be omitted. □

Since $sub(\mathsf{C})$ is a finite set, its powerset is also finite. To be more precise, the size of $\wp(sub(\mathsf{C}))$ is equal to $2^{|sub(\mathsf{C})|}$.

Consider the case of an $\mathcal{ALC}$-CLDS tableau $\mathcal{T}_{AC}$ for the configuration $\mathcal{C} = \{(\mathsf{C} : \lambda_0)\}$. The labelling algebra of the CLDS as well as the knowledge base are considered to be empty. It is easy to see that for every node of $\mathcal{T}_{AC}$ which is labelled with a declarative unit of the form $(a : \lambda)$, a is a subconcept of C. This can be proved using the tableau rules and the observation that a subconcept of a subconcept of C is also a subconcept of C (in other words, the subconcept relation on a set of concepts is transitive). The details of the proof will be omitted. From the above it can be derived that every set $S(\lambda)$, where λ is an arbitrary CLDS-label that appears in the tableau, is a set of subconcepts of C:

$$S(\lambda) \in \wp(sub(\mathsf{C})).$$

It follows that there may exist only $2^{|sub(\mathsf{C})|}$ different such sets and that every such set is itself finite.

Lemma 76. Every CLDS-label is associated with a finite set of concepts.

Proof. From the above analysis, for an arbitrary CLDS-label λ, $S(\lambda) \in \wp(sub(\mathsf{C}))$ and every set in $\wp(sub(\mathsf{C}))$ is finite. □

Lemma 77. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau and B a branch of $\mathcal{T}_{AC}$ for the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$. If the number of CLDS-labels that appear on B is finite, then B contains a finite number of nodes labelled with declarative units.

Proof. Assume that B contains finitely many different CLDS-labels: $\lambda_0 \prec \lambda_1 \prec \ldots \prec \lambda_n$. From lemma 76 it follows that every CLDS-label λ_i $(1 \leq i \leq n)$ is associated with a finite set of subconcepts of C, namely $S(\lambda_i)$. Moreover, there exist no two nodes in the tableau labelled with the same declarative unit (the tableau rules add nodes only if they don't already appear in the tableau). It follows that there exists a finite number of nodes that are labelled with a declarative unit that contains λ_i. Since by assumption there are finitely many CLDS-labels, and we have shown that each one appears in finitely many nodes, it follows that there exist finitely many nodes labelled with declarative units. □

Proposition 78. The number of nodes labelled with Relation-literals that are added by the application of a tableau rule to a branch is smaller than or equal to the number of nodes labelled with declarative units that are added to the branch by the same application.

Proof. This can be easily proved by examining the tableau rules. Assume that we apply the:

- $\sqcap$-rule. In this case two nodes labelled with declarative units are created while no nodes labelled with Relation-literals are added.
- $\sqcup$-rule. As before, only nodes labelled with declarative units are created.
- $\forall$-rule. Only one node labelled with a declarative unit is created.
- $\exists$-rule. Two nodes are created, one labelled with a declarative unit and one labelled with a Relation-literal.

In any case, the number of introduced nodes labelled with Relation-literals is smaller than or equal to the number of introduced nodes labelled with declarative units. □

Proposition 79. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$. The number of nodes of $\mathcal{T}_{AC}$ that are labelled with Relation-literals is smaller than the number of nodes that are labelled with declarative units.

Proof. This can be easily derived. It obviously holds for the initial tableau for $\mathcal{C}$, which consists of a single node labelled $(\mathsf{C} : \lambda)$. Assuming that it holds for a tableau $\mathcal{T}'_{AC}$ for $\mathcal{C}$ it can be shown that it holds for the tableau $\mathcal{T}''_{AC}$ that results from $\mathcal{T}_{AC}$ by the application of a tableau rule, using proposition 78. The details of the proof will be omitted. □

Corollary 80. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$ and B a labelled branch of $\mathcal{T}_{AC}$. If B contains finitely many CLDS-labels, then it is finite.

Proof. Assume that B contains a finite number of (different) CLDS-labels. From lemma 77 it follows that B contains a finite number of nodes labelled with declarative units. From proposition 79 it can be derived that the number of nodes that are labelled with Relation-literals is also finite. Therefore, in total B contains finitely many nodes. □

The following definition will be very useful in the proof of termination.

Definition 81 (∃-nesting degree)**.** Let C be an $\mathcal{ALC}$-concept. The ∃-*nesting degree* of C is equal to the maximum number of ∃ symbols that appear in those subconcepts of C which are of the form $\exists r.\mathsf{D}$ (where r is an arbitrary $\mathcal{ALC}$-role and D an arbitrary $\mathcal{ALC}$-concept).

Example 82. Assume that C is the concept $\exists r_1.\mathsf{A}_1 \sqcap \exists r_2.\exists r_1.\exists r_2.\mathsf{A}_2$. In this case, C has the following subconcepts of the form $\exists r.\mathsf{D}$: $\exists r_1.\mathsf{A}_1$, $\exists r_2.\exists r_1.\exists r_2.\mathsf{A}_2$, $\exists r_1.\exists r_2.\mathsf{A}_2$, and $\exists r_2.\mathsf{A}_2$. The number of ∃ symbols that appear in these concepts is 1, 3, 2, and 1 respectively. Therefore, the ∃-nesting degree of C is equal to 3 because of the subconcept $\exists r_2.\exists r_1.\exists r_2.\mathsf{A}_2$.

Obviously, a concept cannot have more ∃-symbols than the length of the concept. Therefore, the ∃-nesting degree of a concept is bounded by the length of the concept. This is of course a very loose bound.

As a final step before the actual proof of termination, we consider König's lemma, a standard result of graph theory. We will state here without proof a special case of this lemma that refers to trees (and not generally to graphs), as we intend to use it with tableaux (i.e. labelled trees).

Lemma 83 (König's Lemma)**.** A finitely branching tree (namely a tree where every node has finitely many children) is infinite (i.e. it has infinitely many nodes) if and only if it has an infinite branch.

We can now move on to prove termination.

Theorem 84 (Termination)**.** Every $\mathcal{ALC}$-CLDS tableau for checking the satisfiability of the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda_0)\}$ w.r.t. the empty knowledge base.

Proof. Let $\mathcal{T}_{AC}$ be such a tableau and assume that it is not finite. Since every $\mathcal{ALC}$-CLDS tableau is a binary tree (therefore every node has at most two children), it follows from König's Lemma that $\mathcal{T}_{AC}$ contains at least one infinite branch. Let B be an infinite branch of the tableau. If B contained finitely many CLDS-labels, then according to lemma 80, B would be finite. Therefore, B contains infinitely many CLDS-labels.

The number of R-successors of a label is bounded by the maximum nesting degree of ∃-concepts in C. Since B has infinitely many CLDS-labels, it follows that there exists a subconcept of C with infinite nesting degree of ∃-concepts. This leads to a contradiction, since the nesting degree of C is bounded by the length of C which is assumed to be finite. □

10. $\mathcal{ALC}$-CLDS with a Non-Empty TBox

We will consider the most general case, namely the case of a free TBox (that may contain cycles). It is well known in the field of description logics that a TBox that contains several general inclusion axioms can be reduced to a TBox that contains only one general inclusion axiom [4]. To be more specific, let

$$\mathcal{T} = \{\mathsf{C}_1 \dot{\sqsubseteq} \mathsf{D}_1, \ldots, \mathsf{C}_n \dot{\sqsubseteq} \mathsf{D}_n\}$$

be a free TBox. This is equivalent to the TBox $\mathcal{T}'$ that contains only one axiom:

$$\mathcal{T}' = \{\top \dot{\sqsubseteq} ((\neg \mathsf{C}_1 \sqcup \mathsf{D}_1) \sqcap \ldots \sqcap (\neg \mathsf{C}_n \sqcup \mathsf{D}_n))\}.$$

From now on, we will use the symbol $\mathsf{C}_{\mathcal{T}}$ to denote the concept that appears in a free TBox. Namely $\mathsf{C}_{\mathcal{T}} = ((\neg \mathsf{C}_1 \sqcup \mathsf{D}_1) \sqcap \ldots \sqcap (\neg \mathsf{C}_n \sqcup \mathsf{D}_n))$ and a free TBox will be of the form

$$\mathcal{T} = \{\top \dot{\sqsubseteq} \mathsf{C}_{\mathcal{T}}\}.$$

The information of the TBox has to be incorporated in the CLDS framework. We have considered the following three alternatives. The first option is to create a new part in the CLDS framework that would contain the information of the TBox. However, this would require changes in the structure of the CLDS framework as presented in [10]. Since CLDS is destined to be a unifying framework, this alternative was abandoned.

The information of the TBox could also be incorporated in the initial configuration of the $\mathcal{ALC}$-CLDS. However, the axioms of the TBox are not $\mathcal{ALC}$-formulas and therefore, they cannot be directly expressed in the CLDS-language. For example, $\mathsf{C} \dot{\sqsubseteq} \mathsf{D}$ is (syntactically) a correct axiom for the TBox, however the expression $((\mathsf{C} \dot{\sqsubseteq} \mathsf{D}) : \lambda)$ (where λ is a label of the labelling language) is not a CLDS-formula!

The third approach, and the one we opted for, is to include the axiom of the TBox in the extended labelling algebra $\mathrm{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$. This approach provides a unifying representation, since the extended labelling algebra contains all the general information for a specific problem: axioms for the roles and for the concepts. However, recall that the labelling algebra is a *first-order* theory. Therefore, the axiom of the TBox has to be translated into first-order logic. To be more specific, the following axiom is added to the extended labelling algebra:

$$\text{Ax-}\dot{\sqsubseteq} \qquad \forall x [\mathsf{C}_{\mathcal{T}}]^*(x)$$

First of all we will show that checking whether a configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$ is satisfiable w.r.t. a TBox $\mathcal{T} = \{\top \dot{\sqsubseteq} \mathsf{C}_{\mathcal{T}}\}$ is equivalent with checking the satisfiability of C w.r.t. $\mathcal{T}$. The following proposition is the analogous of proposition 64 that established the correspondence between $\mathcal{ALC}$ and $\mathcal{ALC}$-CLDS in the case of the empty TBox.

Proposition 85 (Correspondence between concept satisfiability w.r.t a free TBox in $\mathcal{ALC}$ and in $\mathcal{ALC}$-CLDS)**.** Let C be a concept of $\mathcal{ALC}$, $\langle \langle \mathcal{ALC}, \mathcal{L}_L^{\mathcal{ALC}} \rangle, \emptyset, \mathcal{R} \rangle$ an $\mathcal{ALC}$ CLDS system, and $\mathcal{T} = \{\top \dot{\sqsubseteq} \mathsf{C}_{\mathcal{T}}\}$ a free TBox. C is satisfiable w.r.t. $\mathcal{T}$ if and only if the configuration $\mathcal{C} = \{\mathsf{C} : \lambda\}$ is satisfiable.

Proof. ($\Rightarrow$) Assume that C is satisfiable w.r.t. $\mathcal{T}$. That means that there exists an interpretation $\mathcal{I}$ such that $\mathsf{C}^{\mathcal{I}} \neq \emptyset$ and $\mathcal{I}$ is a model of $\mathcal{T}$, i.e. it satisfies the single axiom that this TBox contains: $\Delta^{\mathcal{I}} \subseteq \mathsf{C}_{\mathcal{T}}^{\mathcal{I}}$. We have to show that the

configuration $\mathcal{C}$ is also satisfiable. In other words, we want to show that there exists a semantic structure $\mathcal{M} = \langle D, I\rangle$ that satisfies the axioms of the extended labelling algebra and $\mathcal{M} \models_{CLDS} \mathcal{C}$. We define $\mathcal{M}$ as in the proof of proposition 64. The domain of $\mathcal{M}$ is the domain of the $\mathcal{ALC}$ interpretation:

$$D = \Delta^{\mathcal{I}}$$

and the interpretation function I agrees with $\cdot^{\mathcal{I}}$ on the interpretation of concepts (unary predicates), roles (binary predicates), and individuals (constants):

$$[[\mathsf{C}]^*]^I = \mathsf{C}^{\mathcal{I}} \qquad R_r^I = r^{\mathcal{I}} \qquad c_a^I = a^{\mathcal{I}}.$$

Obviously, $\mathcal{M}$ satisfies configuration $\mathcal{C}$. It remains to show that it satisfies the axioms of the extended labelling algebra. The case of axioms Ax-$\sqcap$, Ax-$\sqcup$, Ax-$\neg$, Ax-$\forall$, and Ax-$\exists$ is identical to that in the proof of proposition 64 and we will not repeat them here. However, $\mathcal{M}$ has now to satisfy axiom Ax-$\dot{\sqsubseteq}$.

$$\mathcal{M} \models_{FOL} \forall x[\mathsf{C}_{\mathcal{T}}]^*(x) \text{ iff}$$
$$\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_{\mathcal{T}}]^*(x) \text{ for all elements } d \text{ of the domain.}$$

This is equivalent to $d \in [[\mathsf{C}_{\mathcal{T}}]^*]^I$. However, the $\mathcal{ALC}$-interpretation $\mathcal{I}$ satisfies the TBox axiom: $\Delta^{\mathcal{I}} \subseteq \mathsf{C}_{\mathcal{T}}^{\mathcal{I}}$. From the way $\mathcal{M}$ has been defined this is equivalent to $D \subseteq [[\mathsf{C}_{\mathcal{T}}]^*]^I$ and since $d \in D$ it follows that $d \in [[\mathsf{C}_{\mathcal{T}}]^*]^I$.

($\Leftarrow$) Assume configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$ is satisfiable. That means that there exists a semantic structure $\mathcal{M}$ that satisfies the axioms of the extended labelling algebra, and moreover $\mathcal{M} \models_{CLDS} (\mathsf{C} : \lambda)$ or equivalently $\mathcal{M} \models_{FOL} [\mathsf{C}]^*(\lambda)$. This holds if and only if $\lambda \in [[\mathsf{C}]^*]^I$. We have to show that the $\mathcal{ALC}$ concept C is satisfiable w.r.t. the TBox $\mathcal{T}$. We construct an $\mathcal{ALC}$-interpretation $\mathcal{I}$ based on $\mathcal{M}$ exactly as before.

$$\Delta^{\mathcal{I}} = D \qquad \mathsf{C}^{\mathcal{I}} = [[\mathsf{C}]^*]^I \qquad r^{\mathcal{I}} = R_r^I \qquad a^{\mathcal{I}} = c_a^I.$$

According to this construction $\mathsf{C}^{\mathcal{I}} = [[\mathsf{C}]^*]^I \neq \emptyset$ since we know that there exists an $\lambda \in [[\mathsf{C}]^*]^I$. It remains to show that $\mathcal{I}$ is a model of $\mathcal{T}$, i.e. that $\Delta^{\mathcal{I}} \subseteq \mathsf{C}_{\mathcal{T}}^{\mathcal{I}}$. $\mathcal{M}$ satisfies the axiom Ax-$\dot{\sqsubseteq}$ of the extended labelling algebra $\mathcal{M} \models_{FOL} \forall x[\mathsf{C}_{\mathcal{T}}]^*(x)$ or equivalently $d \in [[\mathsf{C}_{\mathcal{T}}]^*]^I$ for every element d of the domain D. From the way $\mathcal{I}$ has been defined, it follows that $d \in \mathsf{C}_{\mathcal{T}}^{\mathcal{I}}$ for every element $d \in \Delta^{\mathcal{I}}$. $\square$

11. $\mathcal{ALC}$-CLDS Tableau for Reasoning in the Presence of a Free TBox

It is also necessary to extend the tableau calculus that we have presented with one new rule that will handle the existence of an axiom in the TBox. Recall that the axioms of the TBox are in a sense a "universal modality" and every individual has to satisfy them. This gives rise to the tableau rule described below.

- The $\dot{\sqsubseteq}$-rule:
  Informally, if the TBox contains the axiom $\top \dot{\sqsubseteq} \mathsf{C}_{\mathcal{T}}$, label λ appears in a branch B and $(\mathsf{C}_{\mathcal{T}} : \lambda)$ does not, then extend B with a new node labelled with this declarative unit.
  Formally, we say that a label λ appears in a branch B if there exists an $\mathcal{ALC}$ concept $\mathsf{C}_{\mathcal{T}}$ such that $(\mathsf{C}_{\mathcal{T}} : \lambda) \in B$ or if there exists a Relation-predicate R_r such that either $R_r(\lambda, \lambda') \in B$ or $R_r(\lambda', \lambda) \in B$. If a label λ appears in B, then add one new node to the end of B $l0$ labelled $(\mathsf{C}_{\mathcal{T}} : \lambda)$. The rule is represented as follows:

$$\frac{(\lambda \text{ appears in the branch})}{\langle l0, (\mathsf{C}_{\mathcal{T}} : \lambda)\rangle}$$

It can be shown that the $\mathcal{ALC}$-CLDS tableau with this additional rule is sound and complete. To prove soundness it is necessary to extend the proof of lemma 66 to include the new tableau rule. In other words, it is enough to show that the $\dot{\sqsubseteq}$-rule preserves satisfiability:

Lemma 86. The $\dot{\sqsubseteq}$-rule preserves satisfiability.

Proof. Let $\mathcal{C}$ be a satisfiable configuration and $\mathcal{T}_{AC} = B_{co}(\mathcal{C})$ the initial tableau for $\mathcal{C}$. We have to show that if $B_{co}(\mathcal{C})$ is open, then the resulting tableau after the application of this rule will also be open.

If we apply the rule to the initial tableau of $\mathcal{C}$, the resulting tableau will have a single branch $\mathcal{T}'_{AC} = B'$ where $\mathcal{C}_{co}(B') = \mathcal{C} \cup \{(\mathsf{C}_{\mathcal{T}} : \lambda)\}$. Since $\mathcal{C}$ is satisfiable, there exists a semantic structure $\mathcal{M}$ that satisfies all the axioms of the extended labelling algebra, therefore it satisfies Ax-$\dot{\sqsubseteq}$: $\mathcal{M} \models_{FOL} \forall x [\mathsf{C}_{\mathcal{T}}]^*(\lambda)$ or equivalently $\mathcal{M}, [x \mapsto d] \models_{FOL} [\mathsf{C}_{\mathcal{T}}]^*(x)$ for all elements d of the domain. It follows that $\mathcal{M}, [x \mapsto \lambda^I] \models_{FOL} [[\mathsf{C}_{\mathcal{T}}]^*]^I(x)$ or in other words, $\lambda^I \in [\mathsf{C}_{\mathcal{T}}]^*$. and therefore, $\mathcal{M} \models_{FOL} [\mathsf{C}_{\mathcal{T}}]^*(\lambda)$ or equivalently $\mathcal{M} \models_{CLDS} (\mathsf{C}_{\mathcal{T}} : \lambda)$. Since $\mathcal{M} \models_{CLDS} \mathcal{C}$ and $\mathcal{C}' = \mathcal{C} \cup \{(\mathsf{C}_{\mathcal{T}} : \lambda)\}$ we conclude that $\mathcal{M} \models_{CLDS} \mathcal{C}'$. That means that $\mathcal{C}'$ is satisfiable and thus the corresponding branch is open. □

In order to prove the completeness we have to extend the definition of a downward saturated configuration (definition 68).

Definition 87 (Extension of definition 68)**.** We add the following case to the definition for downward saturated configurations:

- If a CLDS-label λ appears in $\mathcal{C}$, then $(\mathsf{C}_{\mathcal{T}} : \lambda) \in \mathcal{C}$.

It is also necessary to modify the proof lemma 70, which states that every Hintikka configuration is satisfiable, in order to include the new axiom of the extended labelling algebra.

Proof. (Extension of the proof for lemma 70.)

- Ax-$\dot{\sqsubseteq}$: $\mathcal{M}_H \models_{FOL} \forall x[\mathsf{C}_\mathcal{T}]^*(x)$ if and only if $\mathcal{M}_H \models_{FOL} [\mathsf{C}_\mathcal{T}]^*(d)$ for all elements d of the Herbrand universe of FOT($\mathcal{C}$). Assume that there exists an element $d \in \mathcal{U}$ such that $\mathcal{M}_H \not\models_{FOL} [\mathsf{C}_\mathcal{T}]^*(d)$. That means that $[\mathsf{C}_\mathcal{T}]^*(d) \notin \text{FOT}(\mathcal{C})$. Equivalently, $(\mathsf{C}_\mathcal{T} : d) \notin \mathcal{C}$. However, d is in the Herbrand universe of FOT($\mathcal{C}$) which means that d appears in $\mathcal{C}$. Since $\mathcal{C}$ is a Hintikka configuration, according to definition 87, $(\mathsf{C}_\mathcal{T} : d) \in \mathcal{C}$. Contradiction! □

Finally, in order to prove completeness it is necessary to extend the proof of proposition 72 that every open and complete branch corresponds to a satisfiable configuration.

Proof. (Extension of proof for proposition 72)

- Assume that λ appears in B and $(\mathsf{C}_\mathcal{T} : \lambda) \notin \mathcal{C}$. The $\dot{\sqsubseteq}$-rule can be applied, therefore the branch is not complete. Contradiction! □

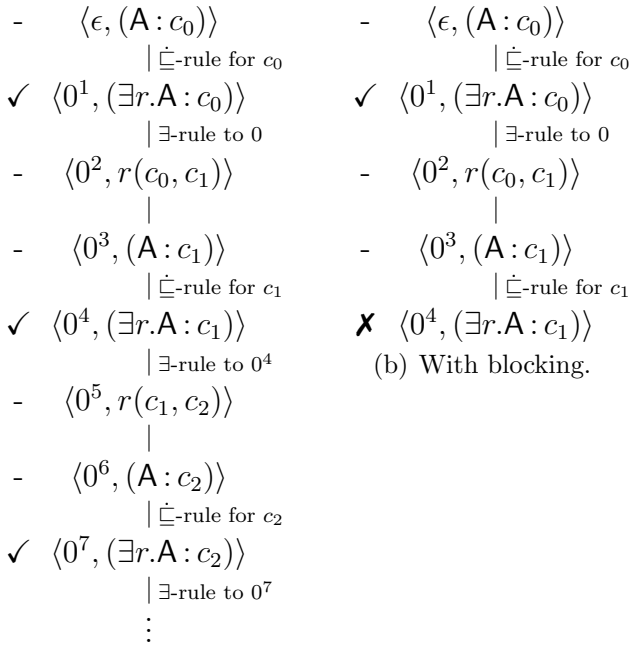
12. Blocking

Although the extra tableau rule and the new axiom in the extended labelling algebra result in a sound and complete tableau procedure for the $\mathcal{ALC}$-CLDS when reasoning w.r.t. a free TBox, they are not enough to guarantee that this is a decision procedure, namely that a tableau can be constructed with an algorithm that always terminates. Consider the following example:

Example 88. Let $\mathcal{T} = \{\top \dot{\sqsubseteq} \exists r.\mathsf{A}\}$ be a (free) TBox and assume that we want to check whether the concept A is satisfiable w.r.t. $\mathcal{T}$. The initial configuration is $\mathcal{C} = \{(\mathsf{A} : c_0)\}$. The corresponding tableau is shown in figure 6(a). Node 0^1 results from the application of the $\dot{\sqsubseteq}$-rule to node ϵ. Nodes 0^2 and 0^3 come from the application of $\exists$-rule to node 0^1. Node 0^4 results from applying the $\dot{\sqsubseteq}$-rule to node 0^3 and so on. This procedure will not terminate. However, notice that the labels of nodes 0^3, 0^4, and 0^5 are similar to those of nodes ϵ, 0^1, and 0^2 respectively. To be more specific, they are the same if we replace c_1 by c_2, and c_0 by c_1 in the labels of the first three nodes. This repetition is observed in the rest of the nodes as well.

Although we have not defined yet the notion of blocking, it is not hard to understand intuitively how the tableau would be if we used blocking. As shown in figure 6(b)), the application of tableau rules stops at node 0^4. Notice that the symbol ✗ is not part of the tableau. It is used in order to denote that the corresponding node is blocked.

To prevent these loops in tableaux, a technique known as *blocking* or *loop checking* is employed. This is commonly used in tableaux for description logics



(a) Without blocking.

(b) With blocking.

Figure 6. An $\mathcal{ALC}$-CLDS tableau for checking the satisfiability of concept A w.r.t. to the TBox $\mathcal{T} = \{\top \dot{\sqsubseteq} \exists r.\mathsf{A}\}$ (example 88).

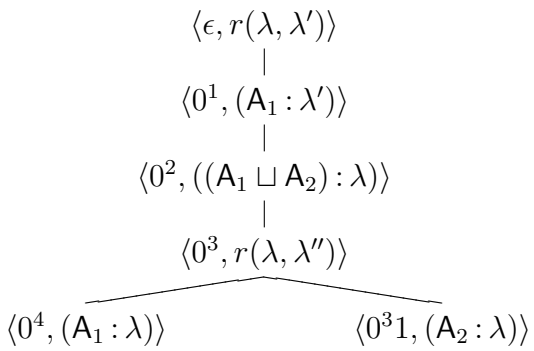


FIGURE 7. An $\mathcal{ALC}$-CLDS tableau for example 89.

(see for example [3] and [4]) as well as in tableaux for hybrid logics [7, 6], and certain modal logics [5].

The main idea behind blocking is to try and detect where a cycle will occur and block the application of generating rules (i.e. of the $\exists$-rule). Informally, if a node n is blocked by a node n', then the application of a generating rule to n will create a cycle. In other words, there is no need to add new successors to node n as it could use the successors of n'. In the remaining of this section we will formally describe how blocking works in a CLDS-tableau. Notice that we won't restrict the description to $\mathcal{ALC}$-CLDS tableaux. It will be a general approach that can be applied to any CLDS-tableau.

Fix a CLDS $\mathfrak{C} = \langle \langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle, \emptyset, \mathcal{R} \rangle$ for the language $\mathcal{L}_{\#}$ and a CLDS-tableau $\mathcal{T}_C = \langle \mathcal{T}, f \rangle$. Every node of $\mathcal{T}_C$ is labelled with either a declarative unit, or with a Relation-literal. A CLDS-label may appear in several declarative units, each time is used as a label for a different $\mathcal{L}_{\#}$-formula. As a first step, we define a function S that maps every CLDS-label that appears in $\mathcal{T}_C$ to the set of $\mathcal{L}_{\#}$-formulas that it labels ($S : \mathcal{L} \to \wp(\mathcal{L}_{\#})$, where $\mathcal{L}$ is the set of labels that appear in $\mathcal{T}_C$):

$$S(\lambda) = \{a \mid n \in \mathcal{T} \text{ and } f(n) = (a : \lambda)\}.$$

Obviously, a is a formula of $\mathcal{L}_{\#}$.

Example 89. Assume that $\mathcal{L}_{\#}$ is $\mathcal{ALC}$ and $\mathcal{T}_{AC}$ is the $\mathcal{ALC}$-CLDS tableau shown in figure 7. Only three different CLDS-labels appear in $\mathcal{T}_{AC}$: λ, λ', and λ''. The value of S for each CLDS-label is:

$$S(\lambda) = \{(\mathsf{A}_1 \sqcup \mathsf{A}_2), \mathsf{A}_1, \mathsf{A}_2\} \qquad S(\lambda') = \{\mathsf{A}_1\} \qquad S(\lambda'') = \emptyset$$

Notice that since λ'' does not appear in any declarative unit but only in Relation-literals, $S(\lambda'')$ is the empty set.

This function S will be used to define what it means for a CLDS-label to be blocked.

Definition 90 (Blocked CLDS-label)**.** A CLDS-label λ is said to be *blocked* if there exists a CLDS-label λ' such that

(i) $S(\lambda) \subset S(\lambda')$, or
(ii) $S(\lambda) = S(\lambda')$ and $\lambda' \prec \lambda$.

What this definition says is that a CLDS-label is blocked if it carries less "$\mathcal{L}_{\#}$-information" than another CLDS-label (case (i) of the above definition).

However, if two CLDS-labels contain the same "$\mathcal{L}_{\#}$-information" (namely $S(\lambda) = S(\lambda')$) we assume that the one that appears earlier in the tableau blocks the one that appears later (case (ii)). This assumption is necessary in order to avoid cyclic blocking: λ would block λ' and λ' would block λ. An example is necessary to better understand how this definition works.

Example 91. Consider again the tableau of figure 6(a). In this case the following hold:

$$S(c_0) = \{\mathsf{A}, \exists r.\mathsf{A}\} \qquad S(c_1) = \{\mathsf{A}, \exists r.\mathsf{A}\} \qquad S(c_2) = \{\mathsf{A}, \exists r.\mathsf{A}\}$$

Obviously, $S(c_0) = S(c_1) = S(c_2)$. Moreover, $c_0 \prec c_1 \prec c_2$. Therefore, according to definition 90, label c_0 blocks c_1 and c_2, and label c_1 blocks c_2.

The minimal label w.r.t. $\prec$ that blocks λ is said to be the *main blocking label* of λ. In the above example, the main blocking label of c_2 is c_0.

Now we can give a definition for blocked nodes. Notice that in essence, blocking forbids the application of a generating rule to a node. Therefore, it makes sense to talk about blocking a node only when a generating rule can be applied to this node. For example, there is no point in saying that a node labelled $(\mathsf{C}_1 \sqcup \mathsf{C}_2 : \lambda)$ is blocked since the only tableau rule that can be applied to this node is the $\sqcap$-rule. In other words, just because the CLDS-label that appears in a node is blocked, this does not necessarily mean that the node is blocked.

Definition 92 (Blocked node)**.** A node is said to be *blocked* if the following conditions are met:

(i) A generating rule can be applied to the node.
(ii) The CLDS-label that appears in the node, λ, is blocked.

12.1. **$\mathcal{ALC}$-CLDS tableau with blocking.** As far as the tableau calculus for $\mathcal{ALC}$-CLDS is concerned, in order to take blocking into consideration we have to make sure that generating rules can be applied to a node only when the node is not blocked. In the case of the $\mathcal{ALC}$-CLDS tableau rules that we have presented, it is necessary to change the preconditions only of the $\exists$-rule.

- The (blocking) $\exists$-rule:

  Informally, if the CLDS-formula $(\exists r.\mathsf{C} : \lambda_1)$ appears in the branch B and the node labelled with this formula is not blocked, then extend B with two nodes labelled $R_r(\lambda_1, \lambda_2)$ and $(\mathsf{C} : \lambda_2)$, where λ_2 is a CLDS label that does not appear already in the branch B.

  Formally, if $\langle n, (\exists r.\mathsf{C} : \lambda_1)\rangle \in B$, n is not blocked and there exists no label λ such that $R_r(\lambda_1, \lambda) \in B$ and $(\mathsf{C} : \lambda) \in B$ then add two nodes to the end of B: $l0$ labelled $R_r(\lambda_1, \lambda_2)$ and $l00$ labelled $(\mathsf{C} : \lambda_2)$, where λ_2 is *new* in the branch B.

  This rule is represented as follows:

$$\frac{\langle n, (\exists r.\mathsf{C} : \lambda_1)\rangle \ , \ n \text{ is not blocked}}{\langle l0, R_r(\lambda_1, \lambda_2)\rangle \ , \ \langle l00, (\mathsf{C} : \lambda_2)\rangle} \qquad (\lambda_2 \text{ is "new" to the branch})$$

12.2. **Termination.** We now have to prove that a tableau calculus that contains the blocking $\exists$-rule is sound, complete, and that it terminates. We will first prove termination as it will be used in the proof of completeness. We start with some preliminary definitions and lemmas that will be useful in the proof of termination.

Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the configuration $\mathcal{C} = \{(\mathsf{C} : \lambda_0)\}$ and assume that the knowledge base consists of a non-empty TBox $\mathcal{T} = \{\top \dot{\sqsubseteq} \mathsf{C}_{\mathcal{T}}\}$. It is easy to see that for every node of $\mathcal{T}_{AC}$ which is labelled with a declarative unit of the form $(a : \lambda)$, a is a subconcept of C or a subconcept of $\mathsf{C}_{\mathcal{T}}$. Thus for every CLDS-label λ that appears in the tableau the set $S(\lambda)$ is a set of subconcepts of C and subconcepts of $\mathsf{C}_{\mathcal{T}}$:

$$S(\lambda) \in \wp(sub(\mathsf{C}) \cup sub(\mathsf{C}_{\mathcal{T}})).$$

It follows that there may exist only $2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_{\mathcal{T}})|}$ different such sets and that every such set is itself finite.

Lemma 93. Every CLDS-label is associated with a finite set of concepts.

Proof. From the above analysis, for an arbitrary CLDS-label λ, $S(\lambda) \in \wp(sub(\mathsf{C}) \cup sub(\mathsf{C}_{\mathcal{T}}))$ and every set in $\wp(sub(\mathsf{C}) \cup sub(\mathsf{C}_{\mathcal{T}}))$ is finite. □

Lemma 94. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau and B a branch of $\mathcal{T}_{AC}$ for the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda)\}$. If the number of CLDS-labels that appear on B is finite, then B contains a finite number of nodes labelled with declarative units.

Proof. Assume that B contains finitely many different CLDS-labels: $\lambda_0 \prec \lambda_1 \prec \ldots \prec \lambda_n$. Every CLDS-label λ_i $(1 \leq i \leq n)$ is associated with a finite set of concepts (lemma 93). Moreover, there exist no two nodes in the tableau labelled with the same declarative unit (the tableau rules add nodes only if they don't already appear in the tableau). It follows that there exists a finite number of nodes that are labelled with a declarative unit that contains λ_i. Since by assumption there are finitely many CLDS-labels, and we have shown that each one appears in finitely many nodes, it follows that there exist finitely many nodes labelled with declarative units. □

We will now prove that there exist a maximum number of labels in a tableau branch that can be unblocked. Notice that the following proposition is about unblocked labels and not unblocked nodes.

Proposition 95. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda_0)\}$ and B a branch of $\mathcal{T}_{AC}$. B contains at most $2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_{\mathcal{T}})|}$ labels that are not blocked.

Proof. Assume for contradiction that B contains more CLDS-labels that are not blocked. Suppose that it contains only one more such label, i.e. in total it contains $1 + 2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_{\mathcal{T}})|}$ labels that are not blocked. However, there exist only $2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_{\mathcal{T}})|}$ different sets of $\mathcal{ALC}$ subconcepts that can be associated with a CLDS-label. That means that there exist at least two CLDS-labels, say λ_i, λ_j, that are associated with exactly the same set of $\mathcal{ALC}$ concepts: $S(\lambda_i) = S(\lambda_j)$. Since we have assumed that labels are introduced to a tableau branch according to the strict total order $\prec$, it follows that either $\lambda_i \prec \lambda_j$ or $\lambda_j \prec \lambda_i$. That means that either λ_i blocks λ_j, or that λ_j blocks λ_i. This is a contradiction since we have assumed that both labels are not blocked. □

We will show that proposition 78 also holds when the TBox is not empty.

Proposition 96. The number of nodes labelled with Relation-literals that are added by the application of a tableau rule to a branch is smaller than or equal to the number of nodes labelled with declarative units that are added to the branch by the same application.

Proof. It is sufficient the proof to proposition 78 with the case of $\dot{\sqsubseteq}$-rule. Assume that we apply the:

- $\dot{\sqsubseteq}$-rule. Only a node labelled with a declarative unit is created.

In any case, the number of introduced nodes labelled with Relation-literals is smaller than or equal to the number of introduced nodes labelled with declarative units. □

Proposition 79 and corollary 80 need no modification.

Lemma 97. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C} = \{(\mathsf{C} : c_0)\}$, and c a CLDS-label that appears in $\mathcal{T}_{AC}$. The number of direct R_r-successors of c is equal to the number of unblocked nodes of the tableau that are labelled with a declarative unit of the form $(\exists r.\mathsf{C} : c)$, where C is an arbitrary $\mathcal{ALC}$-concept.

Proof. By definition, the number of direct R_r-successors of c is equal to the number of Relation-predicates of the form $R_r(c, c')$ that appear in the tableau. Every such predicate has been added to the tableau through the application of the $\exists$-rule to a node labelled $(\exists r.\mathsf{D} : c)$, where D is an arbitrary $\mathcal{ALC}$-concept. However, the $\exists$-rule can only be applied to unblocked nodes. □

We can now move on to prove termination.

Theorem 98 (Termination)**.** Every $\mathcal{ALC}$-CLDS tableau for checking the satisfiability of the initial configuration $\mathcal{C} = \{(\mathsf{C} : \lambda_0)\}$ w.r.t. a free TBox is finite.

Proof. Let $\mathcal{T}_{AC}$ be such a tableau and assume that it is not finite. Since every $\mathcal{ALC}$-CLDS tableau is a binary tree (therefore every node has at most two children), it follows from König's Lemma that $\mathcal{T}_{AC}$ contains at least one infinite branch. Let B be an infinite branch of the tableau. If B contained finitely many CLDS-labels, then according to corollary 80, B would be finite. This cannot be the case, since we have assumed that B is an infinite branch. Therefore, B contains infinitely many CLDS-labels.

We have already proved that a branch contains at most $2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_\mathcal{T})|}$ unblocked labels (see proposition 95). Moreover, every label is associated with a finite set of concepts (lemma 93). Therefore, for a given CLDS-label λ and a given $\mathcal{ALC}$ role r, the number of $\mathcal{ALC}$ concepts of the form $\exists r.\mathsf{D}$ (D is an arbitrary concept) that appear in $S(\lambda)$ is finite. It follows that every unblocked label will have a finite number of direct R_r-successors. Furthermore, the number of Relation-predicate symbols that appear in the tableau is also finite (is the number of Relation-predicate symbols that appear in the concept of the initial configuration plus the number of Relation-predicate symbols that appear in the concept of the TBox). Assume that $R_{r_1}, \ldots, R_{r_n}$ are all the Relation-predicates symbols that appear in the tableau. Since the number of direct R_{r_i}-successors is finite for all $1 \leq i \leq n$, it follows that the number of direct R-successors of λ is finite as well.

Let m denote the maximum number of direct R-successors that an unblocked label has. That means that there exist maximum $m \times 2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_\mathcal{T})|}$ CLDS-labels as direct R-successors of the unblocked labels. On the other hand, a label that is blocked will have no R-successors at all because no generating rules can be applied to a node with a blocked label. That means that in total the maximum number of CLDS-labels is $2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_\mathcal{T})|}$ (the unblocked labels) plus $m \times 2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_\mathcal{T})|}$ (the direct R-successors of the unblocked labels). This is a contradiction, since we have assumed that B contains infinitely many CLDS-labels. □

12.3. **Soundness and Completeness.** To prove soundness it is enough to show that the blocking ∃-rule preserves satisfiability. The proof is identical to the that of the non-blocking ∃-rule in lemma 66 since the only difference between the blocking and the non-blocking rule is in the preconditions of the rules.

On the other hand, the proof of completeness is more complicated. It has to be noted that if it is possible to block nodes, then we cannot use Hintikka configurations to prove completeness as it was the case in the tableau calculus without blocking. This is because if blocking is used in a branch, then the corresponding configuration *is not* a Hintikka configuration. Consider for example the tableau shown in figure 6(b). The corresponding configuration is not a Hintikka one because although it contains the declarative unit $(\exists r.\mathsf{A} : c_1)$ it does not contain a Relation-predicate $R_r(c_1, c_2)$ nor a declarative unit $(\mathsf{A} : c_2)$.

We will instead prove the following proposition.

Proposition 99. If B be an open and complete branch of an $\mathcal{ALC}$-CLDS tableau, then the corresponding configuration $\mathcal{C}_{co}(B)$ is satisfiable.

Proof. It has to be noted that this proposition is the analogous of lemma 70 together with proposition 72 that are used to prove the completeness of the $\mathcal{ALC}$-CLDS tableau when reasoning under the empty TBox. The basic idea of the proof is similar to the proofs of the aforementioned lemma and proposition, however blocking has to be taken under consideration.

In order to show that $\mathcal{C}_{co}(B)$ is satisfiable we have to show that there exists a semantic structure that satisfies all the CLDS-formulas of the configuration. Let $\mathcal{M}_H$ be the Herbrand structure with $\mathcal{H}$ on unary predicates defined as in lemma 70:

$$[[\mathsf{C}]^*]^{\mathcal{H}} = \{d \in \mathcal{U} \mid [\mathsf{C}]^*(d) \in \mathrm{FOT}(\mathcal{C}_{co}(B))\}$$

where $\mathcal{U}$ is the Herbrand universe of $\mathrm{FOT}(\mathcal{C})$. However, the interpretation of binary predicates is defined in a different way in order to take blocking into account:

$$R_r{}^{\mathcal{H}} = \{(d, d') \in \mathcal{U}^2 \mid R_r(d, d') \in \mathrm{FOT}(\mathcal{C}_{co}(B)), \text{ or } d'' \text{ is the main blocking label of } d \text{ and } R_r(d'', d')\}$$

The idea behind this definition is quite simple. If d is blocked by d'' and d' is a direct successor of d'', then do not create new successors for d, but instead "assign" to d the successors of the node that is blocking it. To better understand how this interpretation works see example 101.

Obviously, $\mathcal{M}_H$ satisfies every formula of $\mathcal{C}_{co}(B)$ and it remains to be shown that it satisfies the axioms of the extended labelling algebra. The proof is identical to that of lemma 70 for all the axioms of the extended labelling algebra (including

the extension for Ax-$\dot{\sqsubseteq}$) with the exception of Ax-$\exists$. This case will be presented in detail here.

- Ax-$\exists$: We want to show that $\mathcal{M}_H$ satisfies the Ax-$\exists$:

  $\mathcal{M}_H \models_{FOL} \forall x\,([\exists r.\mathsf{C}]^*(x) \rightarrow \exists y\,(R_r(x,y) \wedge [\mathsf{C}]^*(y)))$ iff
  $\mathcal{M}_H \models_{FOL} ([\exists r.\mathsf{C}]^*(d) \rightarrow \exists y\,(R_r(d,y) \wedge [\mathsf{C}]^*(y)))$ for every $d \in \mathcal{U}$.

  Assume $\mathcal{M}_H \models_{FOL} [\exists r.\mathsf{C}]^*(d)$ for an arbitrary $d \in \mathcal{U}$. We have to show that $\mathcal{M}_H \models_{FOL} \exists y\,(R_r(d,y) \wedge [\mathsf{C}]^*(y))$ or equivalently that $\mathcal{M}_H \models_{FOL} R_r(d,d')$ and $\mathcal{M}_H \models_{FOL} [\mathsf{C}]^*(d')$ for some element $d' \in \mathcal{U}$. That means that there exists a node in B labelled $(\mathsf{C} : d')$ and a node labelled $R_r(d,d')$ for some element $d' \in \mathcal{U}$.

  Since $\mathcal{M}_H \models_{FOL} [\exists r.\mathsf{C}]^*(d)$, it follows that $(\exists r.\mathsf{C} : d) \in \mathcal{C}_{co}(B)$. That means that there exists a node in B labelled $(\exists r.\mathsf{C} : d)$. Let n be this node: $\langle n, (\exists r.\mathsf{C} : d)\rangle \in B$. We distinguish the following two cases:

  – Assume that n is not blocked. Since B is a complete branch, the $\exists$-rule has been applied to the node n labelled $(\exists r.\mathsf{C} : d)$, therefore there exist in B a node labelled $R_r(d,d')$ and a node $\mathsf{C}(d')$, where d' is some element of the domain.

  – Assume that n is blocked. That means that there exists a node, say n' labelled $(a' : \lambda')$, that blocks n. From definition 92 it follows that $S(d) \subseteq S(\lambda')$. Since n is labelled with $(\exists r.\mathsf{C} : d)$ it follows that the $\mathcal{ALC}$-formula $\exists r.\mathsf{C}$ belongs to $S(d)$. Therefore, $\exists r.\mathsf{C}$ also belongs to $S(\lambda')$. That means that there exists a node, say n'' labelled $(\exists r.\mathsf{C} : \lambda')$. If n'' is is not blocked, then the (blocking) $\exists$-rule can be applied to it. Therefore, there exist two nodes labelled $r(\lambda', \lambda'')$ and $\mathsf{C}(\lambda'')$. From the way we have defined the interpretation of binary predicates, it follows that since d is blocked by λ' and $r(\lambda', \lambda'')$, $(d, \lambda'') \in R_r{}^{\mathcal{H}}$. In other words, $\mathcal{M}_H \models_{FOL} R_r(d, \lambda'')$ and $\mathcal{M}_H \models_{FOL} [\mathsf{C}]^*\lambda''$. On the other hand if n'' is blocked then we can repeat the same procedure and find a node that blocks n''. However, since we have prove termination, the number of nodes in B is finite, therefore this procedure cannot "go on forever". At some point we will reach a node that is not blocked. □

Now the completeness of the tableau calculus is straightforward.

Theorem 100 (Completeness of $\mathcal{ALC}$-CLDS tableau w.r.t. a free TBox)**.** If a configuration $\mathcal{C}$ is not satisfiable, then every complete tableau for $\mathcal{C}$ is closed.

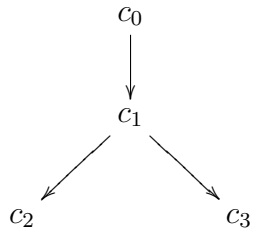
Proof. Assume $\mathcal{C}$ is an unsatisfiable configuration and $\mathcal{T}_{AC}$ a complete tableau for $\mathcal{C}$. Suppose $\mathcal{T}_{AC}$ is open. It follows that there exists a branch B of $\mathcal{T}_{AC}$ that is open and complete. Then according to proposition 99 the corresponding configuration of B is satisfiable. Since $\mathcal{C}$ is contained in $\mathcal{C}_{co}(B)$ it can be derived that $\mathcal{C}$ is satisfiable. Contradiction! □

An example is necessary in order to better understand how the interpretation of Relation-predicates is used to create a model in the case that blocking is available.

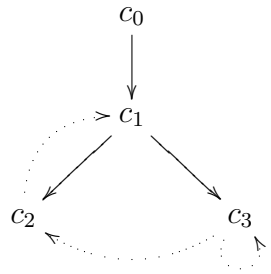
Example 101. Suppose we want to check whether the $\mathcal{ALC}$ concept A is satisfiable w.r.t. to the TBox $\mathcal{T} = \{\top \dot{\sqsubseteq} \exists r.\exists r.\mathsf{A}\}$. A complete tableau for this problem

- $\langle \epsilon, (\mathsf{A} : c_0) \rangle$
  | $\dot{\sqsubseteq}$-rule for c_0
- ✓ $\langle 0, (\exists r.\exists r.\mathsf{A} : c_0) \rangle$
  | $\exists$-rule to 0
- $\langle 0^2, R_r(c_0, c_1) \rangle$
  |
- $\langle 0^3, (\exists r.\mathsf{A} : c_1) \rangle$
  | $\dot{\sqsubseteq}$-rule for c_1
- ✓ $\langle 0^4, (\exists r.\exists r.\mathsf{A} : c_1) \rangle$
  | $\exists$-rule to 0^3
- $\langle 0^5, R_r(c_1, c_2) \rangle$
  |
- $\langle 0^6, (\mathsf{A} : c_2) \rangle$
  | $\dot{\sqsubseteq}$-rule to c_2
- ✗ $\langle 0^7, (\exists r.\exists r.\mathsf{A} : c_2) \rangle$
  | $\exists$-rule to 0^4
- $\langle 0^8, R_r(c_1, c_3) \rangle$
  |
- ✗ $\langle 0^9, (\exists r.\mathsf{A} : c_3) \rangle$
  | $\dot{\sqsubseteq}$-rule to c_3
- ✗ $\langle 0^{10}, (\exists r.\exists r.\mathsf{A} : c_3) \rangle$

(a) A complete tableau for checking the satisfiability of the concept A w.r.t. $\mathcal{T} = \{\top \dot{\sqsubseteq} \exists r.\exists r.\mathsf{A}\}$.



(b) Diagram of the configuration that corresponds to the (single) branch of the tableau. c_2 is blocked by c_0 and c_3 by c_1.



(c) Representation of $R_r{}^{\mathcal{H}}$.

FIGURE 8. Tableau and diagram representation for example 101.

is shown in figure 8(a). The $\mathcal{ALC}$-information that each label carries is:

$$S(c_0) = \{\mathsf{A}, \exists r.\exists r.\mathsf{A}\} \qquad S(c_1) = \{\exists r.\mathsf{A}, \exists r.\exists r.\mathsf{A}\}$$
$$S(c_2) = \{\mathsf{A}, \exists r.\exists r.\mathsf{A}\} \qquad S(c_3) = \{\exists r.\mathsf{A}, \exists r.\exists r.\mathsf{A}\}.$$

Obviously, c_2 is blocked by c_0 and c_3 is blocked by c_1.

The $\mathcal{H}$-interpretation of R_r if we do *not* take blocking into consideration would contain only the following elements: $(c_0, c_1), (c_1, c_2), (c_1, c_3)$. The representation of the corresponding diagram is shown in figure 8(b). However, this interpretation would not satisfy for example the declarative unit $(\exists r.\mathsf{A} : c_3)$.

If we take blocking into consideration, the $\mathcal{H}$-interpretation of R_r would also contain (c_2, c_1), (c_3, c_2), and (c_3, c_3). That is because c_0 is the main blocking label of c_2. Thus c_2 would have as R_r-successors the R_r-successors of c_0, i.e. c_1. Similarly, c_3 would have as R_r-successors the R_r-successors of c_1, i.e. c_2 and c_3. The diagram in this case is shown in figure 8(c). The dotted arrows represent the R_r-successors that are added due to blocking.

13. $\mathcal{ALC}$-CLDS with a Non-Empty ABox

Up to now we have been concerned with knowledge bases whose ABox is empty. In what follows we will examine how to integrate the assertional component of description logics into the CLDS framework.

The statements of the ABox can be represented in the $\mathcal{ALC}$-CLDS system as CLDS-formulas in the initial configuration. To be more specific, every concept assertion $\mathsf{C}(a)$, can be expressed as a declarative unit $(\mathsf{C} : c_a)$ in the initial configuration. Similarly, a role assertion $r(a, b)$ will become a Relation-literal $R_r(c_a, c_b)$ in the initial configuration.

Fix a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. We will use the notation $\mathcal{C}(\mathcal{A})$ to denote the configuration that corresponds to $\mathcal{A}$:

$$\mathcal{C}(\mathcal{A}) = \{(\mathsf{C} : c_a) \mid \mathsf{C}(a) \in \mathcal{A}\} \cup \{R_r(c_a, c_b) \mid r(a, b) \in \mathcal{A}\}$$

First of all we will show the correspondence between ABox consistency in $\mathcal{ALC}$ and configuration satisfiability in $\mathcal{ALC}$-CLDS. In what follows we will consider the (more general) case of ABox consistency w.r.t. a TBox. The case of the empty TBox is obviously a special case of this analysis.

Proposition 102 (Correspondence between ABox consistency in $\mathcal{ALC}$ and configuration satisfiability in $\mathcal{ALC}$-CLDS)**.** Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a knowledge base and $\mathcal{C}(\mathcal{A})$ be the corresponding configuration. $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ if and only if $\mathcal{C}(\mathcal{A})$ is satisfiable.

Proof. ($\Rightarrow$) Assume that $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$. That means that there exists an $\mathcal{ALC}$ interpretation $\mathcal{I}$ that is a model of both the ABox and the TBox. We construct a semantic structure for the $\mathcal{ALC}$-CLDS $\mathcal{M} = \langle D, I \rangle$ exactly like we have done in the proof of correspondence in the case of the empty knowledge base and of the free TBox (propositions 64 and 85 respectively):

$$D = \Delta^{\mathcal{I}} \qquad [[\mathsf{C}]^*]^I = \mathsf{C}^{\mathcal{I}} \qquad R_r^I = r^{\mathcal{I}} \qquad c_a^I = a^{\mathcal{I}}.$$

$\mathcal{I}$ satisfies every assertion of the ABox and every axiom of the TBox. Therefore, for any concept assertion $\mathsf{C}(a) \in \mathcal{A}$ it is the case that $a^{\mathcal{I}} \in \mathsf{C}^{\mathcal{I}}$. From the way $\mathcal{M}$ has been defined, this is equivalent to $c_a \in [[\mathsf{C}]^*]^I$ or equivalently $\mathcal{M} \models_{FOL} [\mathsf{C}]^*(c_a)$. Similarly, for any role assertion $r(a, b) \in \mathcal{A}$ it follows that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ or equivalently $(c_a^I, c_b^I) \in R_r^I$. It follows that $\mathcal{M} \models_{FOL} R_r(c_a, c_b)$.

It remains to show that $\mathcal{M}$ is a semantic structure for the $\mathcal{ALC}$-CLDS, namely that it satisfies the axioms of the extended labelling algebra. The proof is exactly like in the case of proposition 85 and we won't repeat it here.

($\Leftarrow$) Assume that $\mathcal{C}(\mathcal{A})$ is satisfiable. We have to show that $\mathcal{A}$ is consistent. Given the semantic structure for the CLDS $\mathcal{M}$ we define an $\mathcal{ALC}$ interpretation $\mathcal{I}$ exactly as above. Obviously it satisfies all the assertions of the ABox. □

14. $\mathcal{ALC}$-CLDS Tableau for Reasoning in the Presence of a Non-Empty ABox

The presence of a non-empty ABox results in an initial configuration that may contain more than one formulas. That means that the initial tableau will not consist of just one node, as it was the case until now. Instead, it will generally be a single branch with more than one nodes. However, the tableau calculus as defined so far does not at any point make use of the fact that the initial tableau was a single node. Therefore, the generalization to tableau for a non-empty ABox is rather trivial. The tableau rules are the same, as well as the proof of soundness and completeness. However, the proof of termination as presented in the case of a free TBox (with blocking) needs some minor adaptations. We will present these changes here in detail.

14.1. **Termination.** Fix a finite configuration $\mathcal{C}$ and let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for $\mathcal{C}$. It is obvious that for every node of $\mathcal{T}_{AC}$ which is labelled with a declarative unit of the form $(a:\lambda)$, a is a subconcept of one of the concepts that appear in the initial configuration or in the TBox. It follows that for every CLDS-label λ that appears in $\mathcal{T}_{AC}$ the set $S(\lambda)$ consists of subconcepts of the concepts that appear in $\mathcal{C}$ and of the concept $\mathsf{C}_{\mathcal{T}}$. To be more specific, if $\mathsf{C}_1, \ldots, \mathsf{C}_n$ are all the concept names that appear in $\mathcal{C}$, then:

$$S(\lambda) \in \wp(sub(\mathsf{C}_1) \cup \ldots \cup sub(\mathsf{C}_n) \cup sub(\mathsf{C}_{\mathcal{T}})).$$

From lemma 75, each one of the sets $sub(\mathsf{C}_1),\ldots, sub(\mathsf{C}_n)$, $sub(\mathsf{C}_{\mathcal{T}})$ is finite. That means that the union of these sets is also finite as well as the powerset of the union. To be more specific, the size of $\wp(sub(\mathsf{C}_1) \cup \ldots \cup sub(\mathsf{C}_n) \cup sub(\mathsf{C}_{\mathcal{T}}))$ is equal to $2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_{\mathcal{T}})|}$. Therefore, the following holds:

Lemma 103. Every CLDS-label is associated with a finite set of declarative units.

The following lemma is a generalization of lemma 77 that handles an arbitrary initial configuration.

Lemma 104. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C}$ and B a branch of $\mathcal{T}_{AC}$. If the number of CLDS-labels that appear on B is finite, then B contains a finite number of nodes labelled with declarative units.

Proof. Let $\mathsf{C}_1, \ldots, \mathsf{C}_n$ be the concept names that appear in the initial configuration $\mathcal{C}$. Assume that B contains finitely many different CLDS-labels: $\lambda_0 \prec \lambda_1 \prec \ldots \prec \lambda_n$. Every CLDS-label λ_i $(1 \leq i \leq n)$ is associated with a set $S(\lambda_i)$ of subconcepts of the concepts that appear in the initial configuration. We have shown that there exists a finite number of such sets, and each one of them is finite. Therefore, since there exist no two nodes in the tableau labelled with the same declarative unit, it follows that there exists a finite number of nodes that are labelled with a declarative unit that contains λ_i. Since by assumption there are finitely many CLDS-labels, and we have shown that each one appears in finitely many nodes, it follows that there exist finitely many nodes labelled with declarative units. □

We also have to generalize proposition 95 to include the case when the initial configuration does not consist of a single declarative unit.

Proposition 105. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for an initial configuration $\mathcal{C}$ and B a branch of $\mathcal{T}_{AC}$. B contains a finite number of labels that are not blocked.

Proof. The initial configuration is a finite set of CLDS-formulas, therefore it contains a finite number of Relation-literals and a finite number of declarative units, say num_{rp} and num_{du} respectively. Let $\mathsf{C}_1, \ldots, \mathsf{C}_n$ be the concept names that appear in the initial configuration. According to the above analysis, there exist $2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_\mathcal{T})|}$ different sets of concepts that can be associated with a CLDS-label, therefore there can only be $2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_\mathcal{T})|}$ unblocked labels. □

We also need to extend proposition 79.

Proposition 106. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C}$ and B a labelled branch of $\mathcal{T}_{AC}$. If there exists a finite number of nodes in B labelled with declarative units, then the number of nodes in B labelled with Relation-literals is also finite.

Proof. The number of Relation-literals in the tableau is the number of those that exist in the initial configuration and those that have been added through the application of the tableau rules. According to proposition 78 the Relation-literals created from application rules are less than the declarative units created from the rules. Therefore, if the number of declarative units created by the rules is finite, so is the number of Relation-literals created by the rules. Moreover, since the initial configuration is finite, it follows that the number of nodes labelled with declarative units and the number of nodes labelled with Relation-literals (i.e. all the nodes of the tableau) is finite. □

Corollary 107. Let $\mathcal{T}_{AC}$ be an $\mathcal{ALC}$-CLDS tableau for the initial configuration $\mathcal{C}$ and B a labelled branch of $\mathcal{T}_{AC}$. If B contains finitely many CLDS-labels, then it is finite.

Proof. From lemma 104 it follows that B contains a finite number of nodes labelled with declarative units. From the above analysis it can be derived that the number of nodes that are labelled with Relation-predicates is also finite. Therefore, in total B contains finitely many nodes. □

Now we are able to generalize theorem 98.

Theorem 108 (Termination)**.** Every $\mathcal{ALC}$-CLDS tableau for checking the satisfiability of an initial configuration $\mathcal{C}$ w.r.t. a free TBox is finite.

Proof. Let $\mathcal{T}_{AC}$ be such a tableau and assume that it is not finite. Since every $\mathcal{ALC}$-CLDS tableau is a binary tree (therefore every node has at most two children), it follows from König's Lemma that $\mathcal{T}_{AC}$ contains at least one infinite branch. Let B be an infinite branch of the tableau. If B contained finitely many CLDS-labels, then according to corollary 107, B would be finite. This cannot be the case, since we have assumed that B is an infinite branch. Therefore, B contains infinitely many CLDS-labels.

A branch contains at most $2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_\mathcal{T})|}$ unblocked labels (see proposition 105). Moreover, every label is associated with a finite set of concepts (lemma 103). Therefore, for a given CLDS-label λ and a given $\mathcal{ALC}$ role r, the

number of $\mathcal{ALC}$ concepts of the form $\exists r.\mathsf{D}$ (D is an arbitrary concept) that appear in $S(\lambda)$ is finite. It follows that every unblocked label will have a finite number of direct R_r-successors created by the application of the $\exists$ rule. Furthermore, the number of Relation-predicate symbols that appear in the tableau is also finite (is the number of Relation-predicate symbols that appear in the concept of the initial configuration plus the number of Relation-predicate symbols that appear in the concept of the TBox). Assume that $R_{r_1}, \ldots, R_{r_n}$ are all the Relation-predicates symbols that appear in the tableau. Since the number of direct R_{r_i}-successors is finite for all $1 \leq i \leq n$, it follows that the number of direct R-successors of λ created by the application of generating rules is finite as well. Since the number of role assertions that there exist in the ABox is finite, it follows that the number of direct R-successors of λ (those created by the rules and those that are due to role assertions of the ABox) is finite.

Let m denote the maximum number of direct R-successors that an unblocked label has. That means that there exist maximum $m \times 2^{|sub(\mathsf{C})|+|sub(\mathsf{C}_\mathcal{T})|}$ CLDS-labels as direct R-successors of the unblocked labels. On the other hand, a label that is blocked will have no R-successors at all because no generating rules can be applied to a node with a blocked label. That means that in total the maximum number of CLDS-labels is $2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_\mathcal{T})|}$ (the unblocked labels) plus $m \times 2^{|sub(\mathsf{C}_1)|+\ldots+|sub(\mathsf{C}_n)|+|sub(\mathsf{C}_\mathcal{T})|}$ (the direct R-successors of the unblocked labels). This is a contradiction, since we have assumed that B contains infinitely many CLDS-labels. □

15. $\mathcal{ALC}$-CLDS with Non-Empty Labelling Algebra

Until now we have only considered CLDSs whose labelling algebra was the empty set. In this section we will examine an $\mathcal{ALC}$-CLDS with a non-empty labelling algebra. Remember that the labelling algebra contains formulas of the semi-extended labelling language. In the case of the $\mathcal{ALC}$-CLDS, the labelling language is a first-order language with the signature $\langle \mathcal{P}, \emptyset, \mathcal{C} \rangle$, where $\mathcal{P}$ is a set of binary predicates that correspond to the roles of the $\mathcal{ALC}$ language of the CLDS. It follows that the labelling algebra can only contain *information regarding the roles*. In other words, the labelling algebra is the equivalent of the RBox of description logics.

The labelling algebra is assumed to contain only *universally quantified definite Horn clauses*. This is because such formulas can be written as implications and consequently the creation of tableau rules is straightforward.

Example 109. Consider for example that the labelling algebra contains the following universally quantified definite Horn clause that expresses transitivity of R_r:

$$\forall x \forall y \forall z \left(\neg R_r(x, y) \vee \neg R_r(y, z) \vee R_r(x, z) \right).$$

This can be written equivalently as

$$\forall x \forall y \forall z \left((R_r(x, y) \wedge R_r(y, z)) \rightarrow R_r(x, z) \right).$$

Intuitively, we need a tableau rule that whenever $R_r(x, y)$ and $R_r(y, z)$ appear in a branch, the CLDS-formulas $R_r(x, z)$ will be added to the branch.

With universally quantified Horn clauses it is possible to express some commonly used role axioms and role constructors (see tables 2(a) and 2(b) for examples). Notice that in the case of role intersection, the axiom

$$\forall x \forall y \left((R_{r_1}(x, y) \wedge R_{r_2}(x, y)) \leftrightarrow R_{r_1 \cap r_2}(x, y) \right)$$

is not itself a universally quantified Horn clause. However it can be written equivalently as the conjunction of three universally quantified Horn clauses:

$$\begin{gathered} \forall x \forall y ((R_{r_1}(x, y) \wedge R_{r_2}(x, y)) \rightarrow R_{r_1 \cap r_2}(x, y)) \\ \forall x \forall y (R_{r_1 \cap r_2}(x, y) \rightarrow R_{r_1}(x, y)) \\ \forall x \forall y (R_{r_1 \cap r_2}(x, y) \rightarrow R_{r_2}(x, y)) \end{gathered}$$

It is not possible, however, to express all the desired properties as universally quantified Horn clauses. Some examples of constructors that cannot be expressed with such clauses are presented in table 2(c).

Notice that if the labelling algebra is not empty, then the corresponding description language that the CLDS describes is not $\mathcal{ALC}$ anymore. For example, if the labelling algebra asserts the transitivity of some roles, then the corresponding description language will be $\mathcal{ALC}$ with transitive roles, namely the language $\mathcal{S}$. However, we will still refer to the CLDS as an $\mathcal{ALC}$-CLDS as the description language contains exactly the constructors of $\mathcal{ALC}$.

With the labelling algebra restricted to universally quantified Horn clauses we can express description languages that are subsets of $\mathcal{SHI}$, i.e. $\mathcal{ALC}$ with transitive roles ($\mathcal{S}$), role hierarchies ($\mathcal{H}$), and inverse roles ($\mathcal{I}$). Such description languages have been studied and there exist tableau procedures that decide them [20].

TABLE 2. Axioms of the labelling algebra for the $\mathcal{ALC}$-CLDS

(a) Role axioms of the RBox expressed as axioms of a labelling algebra with universally quantified Horn clauses.

Property	RBox Axioms	Axiom
Transitivity	$Trans(r)$	$\forall x \forall y \forall z \left((R_r(x,y) \wedge R_r(y,z)) \rightarrow R_r(x,z) \right)$
Reflexivity		$\forall x R_r(x,x)$
Symmetry		$\forall x \forall y \left(R_r(x,y) \rightarrow R_r(y,x) \right)$
Role Hierarchy	$r_1 \dot{\sqsubseteq}_R r_2$	$\forall x \forall y \left(R_{r_1}(x,y) \rightarrow R_{r_2}(x,y) \right)$

(b) Role Constructors expressed in a labelling algebra with universally quantified Horn clauses.

Property	Constructor	Axiom
Inverse Role	r^-	$\forall x \forall y \left(R_r(x,y) \leftrightarrow R_{r^-}(y,x) \right)$
Role Intersection	$r_1 \cap r_2$	$\forall x \forall y \left((R_{r_1}(x,y) \wedge R_{r_2}(x,y)) \leftrightarrow R_{r_1 \cap r_2}(x,y) \right)$

(c) Role Constructors that can be expressed in a labelling algebra but not with universally quantified Horn clauses.

Property	Constructor	Axiom
Role Composition	$r_1 \circ r_2$	$\forall x \forall y \exists z \left(R_{r_1}(x,y) \wedge R_{r_2}(y,z) \leftrightarrow R_{r_1 \circ r_2}(x,z) \right)$
Role Complement	$\neg r$	$\forall x \forall y \left(R_r(x,y) \leftrightarrow \neg R_{\neg r}(x,y) \right)$

16. $\mathcal{ALC}$-CLDS Tableau for Reasoning in the Presence of a Non-Empty Labelling Algebra

In the case the labelling algebra is not empty it is necessary to extend the tableau calculus we have presented so far with rules that will handle the axioms of the labelling algebra. For every axiom of the labelling algebra, a new tableau rule has to be added. However, all such rules follow a general scheme: the preconditions of the rule are the formulas that appear on the left hand side of the implication, and the formula that will be added to the tableau is the formula at the right hand side of the implication.

In what follows we will consider a labelling algebra that contains a finite number of axioms of the general form:

$$\forall x_1, \ldots \forall x_n \left((R_{r_1}(x_1, x_2) \wedge \ldots \wedge R_{r_m}(x_{n-3}, x_{n-2})) \rightarrow R_{r_{m+1}}(x_{n-1}, x_n) \right)$$

In this case the tableau calculus will be extended with a finite number of rules (one for each axiom of the labelling algebra). Every such rule will have the general form described below.

- Formally, if $R_{r_1}(c_1, c_2) \in b$ and $\ldots$ and $R_{r_m}(c_{n-3}, c_{n-2}) \in B$ then add a node to the end of B: $l0$ labelled $R_{r_{m+1}}(c_{n-2}, c_n)$.
  This rule is represented as follows:

$$\frac{\langle n_1, R_{r_1}(c_1, c_2)\rangle, \ldots, \langle n_m, R_{r_m}(c_{n-3}, c_{n-2})\rangle \ , \ n_1, \ldots, n_m \in b_l}{\langle l0, R_{r_{m+1}}(c_{n-2}, c_n)\rangle}$$

In what follows we will refer to the above axiom as the *Role-axiom* and to the rule as the *Role-rule*.

It is worth mentioning that in the case the labelling algebra is not empty, the tableau may not terminate without blocking. Consider the following example.

Example 110. Let $\mathfrak{C}$ be a CLDS whose labelling algebra contains a transitivity axiom for the binary predicate R_r:

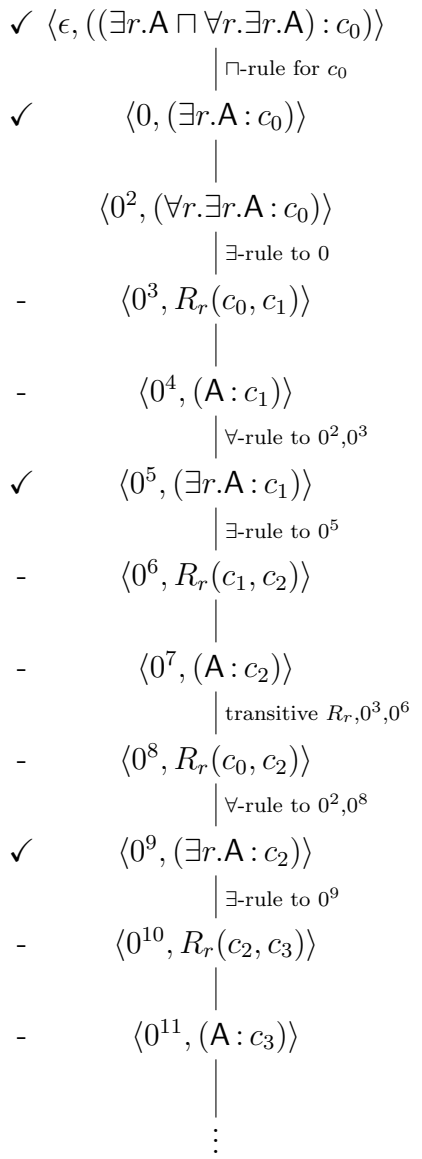
$$\forall x \forall y \forall z \left((R_r(x, y) \wedge R_r(y, z)) \rightarrow R_r(x, z) \right)$$

Assume that we are interested in knowing whether the concept $(\exists r.\mathsf{A} \sqcap \forall r.\exists r.\mathsf{A})$ is satisfiable. The corresponding tableau is presented in figure 9. The information that is associated with each label is:

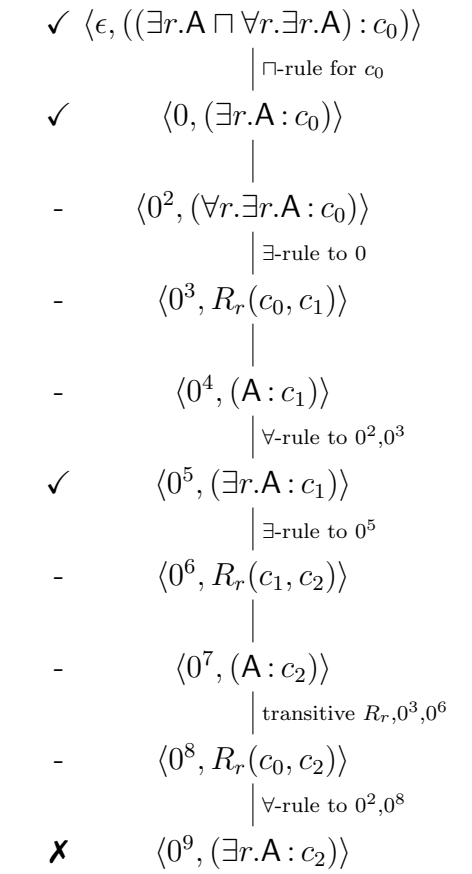
$$S(c_0) = \{(\exists r.\mathsf{A} \sqcap \forall r.\exists r.\mathsf{A}), \exists r.\mathsf{A}, \forall r.\exists r.\mathsf{A}\}$$
$$S(c_1) = \{\mathsf{A}, \exists r.\mathsf{A}\}$$
$$S(c_2) = \{\mathsf{A}, \exists r.\mathsf{A}\}$$

Therefore, label c_2 is blocked by c_1. It follows that node 0^9 is blocked and the $\exists$-rule cannot be applied. The tableau shown in figure 9(b) is a complete tableau and is also finite.

16.1. **Soundness.** To prove that the tableau calculus extended with the new rules is sound, it is sufficient to prove that every new rule preserves satisfiability. In other words, lemma 66 has to be extended with the case of the new rules. Since all the rules for the role axioms have are of the same form, it is enough to show that one rule (with the general form) preserves satisfiability.



(a) Tableau without blocking



(b) Tableau with blocking: c_2 is blocked by c_1.

FIGURE 9. Tableaux for example 110.

Lemma 111 (Extension of lemma 66: Tableau rules preserve satisfiability)**.** Let $\mathcal{C}$ be a satisfiable configuration and $\mathcal{T}_{AC} = B_{co}(\mathcal{C}) = B = \langle b, f, B_{lab}\rangle$ the initial tableau for $\mathcal{C}$. If a tableau rule can be applied to $\mathcal{T}_{AC}$, then the resulting tableau $\mathcal{T}'_{AC}$ will also be open.

Proof. Only the case of the Role-rule will be examined here.

- The applied rule is the Role-rule:
  That means that there exist m nodes in $B_{co}(\mathcal{C})$ labelled $R_{r_1}(c_1, c_2)$, $\ldots$, $R_{r_m}(c_{n-3}, c_{n-2})$. It follows that $R_{r_1}(c_1, c_2)$, $\ldots$, $R_{r_m}(c_{n-3}, c_{n-2})$ also belong in the configuration $\mathcal{C}$. Since $\mathcal{C}$ is satisfiable, there exists a semantic structure $\mathcal{M}$ that satisfies all the CLDS-formulas of $\mathcal{C}$. Therefore, $\mathcal{M} \models_{CLDS} R_{r_1}(c_1, c_2)$, $\ldots$, $\mathcal{M} \models_{CLDS} R_{r_m}(c_{n-3}, c_{n-2})$. Equivalently,

$\mathcal{M} \models_{FOL} R_{r_1}(c_1, c_2)$, ..., $\mathcal{M} \models_{FOL} R_{r_m}(c_{n-3}, c_{n-2})$. It follows that

$$\mathcal{M} \models_{FOL} R_{r_1}(c_1, c_2) \wedge \ldots \wedge R_{r_m}(c_{n-3}, c_{n-2}).$$

However, $\mathcal{M}$ satisfies the axioms of the extended labelling algebra, which include the axioms of the labelling algebra. That means that $\mathcal{M}$ satisfies the Role-axiom and therefore $\mathcal{M} \models_{FOL} R_{r_{m+1}}(c_{n-1}, c_n)$.

The resulting tableau $\mathcal{T}'_{AC}$ consists of a single branch B' which is the branch $B_{co}(\mathcal{C})$ extended with a node labelled $R_r(c_{n-1}, c_n)$. The corresponding configuration is $\mathcal{C}_{co}(B') = \mathcal{C} \cup \{R_r(c_{n-1}, c_n)\}$ and it can be easily seen that $\mathcal{M}$ satisfies this configuration as well. Therefore B' is also an open branch. □

16.2. **Completeness.** The proof of completeness for this tableau calculus follows the same line as the completeness proof of the tableau calculus in the case of the free TBox. It is enough to extend proposition 99 to include the Role-axiom.

Proof. (Extension of proof for proposition 99.) Recall that a semantic structure $\mathcal{M}$ has been defined with

$$[[\mathsf{C}]^*]^{\mathcal{H}} = \{d \in \mathcal{U} \mid [\mathsf{C}]^*(d) \in \mathrm{FOT}(\mathcal{C}_{co}(B))\}$$
$$R_r{}^{\mathcal{H}} = \{(d, d') \in \mathcal{U}^2 \mid R_r(d, d') \in \mathrm{FOT}(\mathcal{C}_{co}(B)), \text{ or}$$
$$d'' \text{ is the main blocking label of } d \text{ and } R_r(d'', d')\}$$

where $\mathcal{U}$ is the Herbrand universe of $\mathrm{FOT}(\mathcal{C})$. We need to show that $\mathcal{M}$ satisfies the Role-axiom:

$\mathcal{M} \models_{FOL} \forall x_1, \ldots \forall x_n \left((R_{r_1}(x_1, x_2) \wedge \ldots \wedge R_{r_m}(x_{n-3}, x_{n-2})) \rightarrow R_{r_{m+1}}(x_{n-1}, x_n)\right)$ iff
$\mathcal{M} \models_{FOL} (R_{r_1}(d_1, d_2) \wedge \ldots \wedge R_{r_m}(d_{n-3}, d_{n-2})) \rightarrow R_{r_{m+1}}(d_{n-1}, d_n)$

for every $d_1, \ldots, d_n \in \mathcal{U}$. Assume $\mathcal{M} \models_{FOL} R_{r_1}(d_1, d_2) \wedge \ldots \wedge R_{r_m}(d_{n-3}, d_{n-2})$ or equivalently $\mathcal{M} \models_{FOL} R_{r_1}(d_1, d_2)$ and ... and $\mathcal{M} \models_{FOL} R_{r_m}(d_{n-3}, d_{n-2})$. It follows that for each one of these binary predicates there exists a node of B labelled with the predicate. Since the tableau is complete and all the predicates appear in the branch, the Role-rule has been applied, therefore, $R_{r_{m+1}}(d_{n-1}, d_n)$ also appears in the branch. From the way the $\mathcal{H}$ has been defined, $(d_{n-1}, d_n) \in R_r{}^{\mathcal{H}}$ which is equivalent to $\mathcal{M} \models_{FOL} R_r(d_{n-1}, d_n)$.

The same can be done for every tableau rule that corresponds to an axiom of the labelling algebra. Therefore, a tableau calculus with a finite number of such rules is complete. □

Blocking does not play any role in the proof of completeness, and the above proof could have been done by extending the definition of Hintikka configurations (as was the completeness proof in the case of the free TBox without blocking). This is not unexpected, since blocking stops the application of generating rules. However, the Role-rule is not a generating one. Consider again example 110. The cycle that has been created due to the transitive role will be detected and stopped in node 0^9 which is labelled with an $\exists$-formula.

16.3. **Termination.** It remains to show termination. The proof is similar to that for the termination in the presence of a free TBox and of an ABox. Again, we have to prove that when a branch of an $\mathcal{ALC}$-CLDS tableau contains finitely many CLDS-labels, then the number of nodes labelled with Relation-predicates is finite. We will continue with the proof of this statement, which is obviously more general than proposition 79.

Proposition 112. Let $\mathfrak{C}$ be an $\mathcal{ALC}$-CLDS with a non-empty labelling algebra and $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ a knowledge base. Let $\mathcal{T}_{AC}$ be an $\mathfrak{C}$-tableau for an initial configuration $\mathcal{C}$. If a branch B of $\mathcal{T}_{AC}$ contains a finite number of CLDS-labels, then it also contains a finite number of nodes labelled with Relation-predicates.

Proof. First of all, the number of Relation-predicate symbols that appear in the branch is finite. That is because every Relation-predicate symbol that appears in the tableau appears in the ABox (i.e. the initial configuration), the TBox, or the labelling algebra. In other words, there is no way to create new Relation-predicate symbols other than those that appear in the aforementioned places.

Assume that there exist num_R different Relation-predicate symbols and let n denote the number of CLDS-labels that appear on the branch (n is by assumption finite). It is easy to see that with num_R Relation-predicate symbols and n CLDS-labels we can have $num_R \times n \times n$ different Relation-predicates.

Since the tableau rules do not allow the addition of a Relation-predicate in the tableau if it already appears in it, it follows that there can only be $n^2 \times num_R$ Relation-predicates in the branch. □

The proof of termination for theorem 108 needs to be modified in the case the labelling algebra is not empty. To be more specific, the direct R-successors of unblocked labels do not only originate from concepts of the form $\exists r.\mathsf{D}$ and the assertions of the ABox, but additionally from the Role-rules. However, we can still use the argument that there is a finite number of unblocked CLDS-labels, and each one of them will have a finite number of direct R-successors that result from the application of generating rules. Moreover, the number of direct R-successors due to role assertions in the ABox is also finite. We also have to consider the direct R-successors that will be added from the axioms of the labelling algebra. Since every axiom adds at most one R-successor to a label, and there are finitely may role axioms, it follows that the number of R-successors that can be added from the role axioms is finite. Therefore, there still exists a finite number of CLDS-labels.

17. CLDS for more expressive description languages

The CLDS that we have presented for $\mathcal{ALC}$ can be easily extended for description languages with more concept constructors. In this section we will briefly describe a system for a description language with number restrictions ($\mathcal{N}$).

17.1. **An $\mathcal{ALCN}$-CLDS.** $\mathcal{ALCN}$ is the language $\mathcal{ALC}$ extended with number restrictions. An $\mathcal{ALCN}$-CLDS is tuple $\langle\langle\mathcal{ALCN}, \mathcal{L}_L^{\mathcal{ALCN}}\rangle, \emptyset, \mathcal{R}\rangle$, where the labelling language $\mathcal{L}_L^{\mathcal{ALCN}}$ is the same as $\mathcal{L}_L^{\mathcal{ALC}}$ (the labelling language in the case of $\mathcal{ALC}$-CLDS) and in addition it includes equality. The reason for that will become apparent later. In the case of the empty knowledge base the set of inference rules is

$$\mathcal{R} = \{(\sqcap), (\sqcup), (\text{blocking-}\exists), (\forall), (\leq), (\geq)\}.$$

Notice that $\mathcal{R}$ contains two new tableau rules, $(\leq)$-rule and $(\geq)$-rule, that handle the constructors for the number restrictions, in addition to the tableau rules of the $\mathcal{ALC}$-CLDS when reasoning under the empty knowledge base.

In the presence of constructor for number restrictions it is necessary to be able to distinguish if two CLDS-labels (i.e. constants of the labelling language) are mapped to the same element of the domain. Remember that we have not adopted the Unique Name Assumption, therefore if the interpretation of two labels is different this has to be made explicit. To achieve that we allow *inequality assertions* between CLDS-labels: $\lambda_i \neq \lambda_j$ iff $\lambda_i^I \neq \lambda_j^I$ (this is the reason why the labelling language needs to contain equality). This is commonly used in tableaux for description languages that contain number restrictions (see for example [4]). Unique Name Assumption is usually not adopted, especially if number restrictions are used, because such an assumption results in a less flexible approach. After all, it can be imposed when necessary by using the inequality assertions. This will be made clear later with example 115.

Moreover, a concept that contains number restrictions can be written in Negation Normal Form. For example the concept $\neg \leq_n r$ is equivalent to $\geq_{n+1} r$ and similarly, $\neg \geq_n r$ is equivalent to $\leq_{n-1}$.

An $\mathcal{ALCN}$-CLDS tableau is defined in a similar way to the $\mathcal{ALC}$-CLDS tableau, but in addition, it may contain nodes labelled with inequality assertions. To be more specific, an $\mathcal{ALCN}$-CLDS tableau is a labelled tree whose nodes are labelled (i) with declarative units of the form $(\mathsf{C} : \lambda)$ where C is an $\mathcal{ALCN}$ concept, or (ii) with Relation-literals, or (iii) with inequality assertions.

We will now describe in more detail the tableau rules introduced for the number restrictions. Fix an $\mathcal{ALCN}$-CLDS tableau $\mathcal{T}_{\mathcal{ALCN}}^{CLDS}$ and a labelled branch B of $\mathcal{T}_{\mathcal{ALCN}}^{CLDS}$ whose leaf node is l.

The $(\leq)$-rule: Informally, if a node of B is labelled with a declarative unit of the form $(\leq_n r : \lambda)$, and λ has more than n R_r-direct successors that are not asserted to be different from each other (at least not all of them), then some of them will have to correspond to the same element of the domain. Therefore, for every possible combination of λ with one of its direct R_r-successors that we are not sure they are different form λ, we create a new tableau where the direct successor coincides with λ.

More formally, if there exists a node in B labelled $(\leq_n r : \lambda)$ and there exist $m > n$ nodes labelled with Relation-predicates $R_r(\lambda, \lambda'_1), \ldots, R_r(\lambda, \lambda'_m)$ ($\lambda'_i \prec \lambda'_j$ for all $1 \leq i < j \leq m$) but there exists λ'_i and λ'_j such that they

are not asserted to be different (namely there exists no node in B labelled $\lambda'_i \neq \lambda'_j$), then for every pair of nodes λ'_j, λ'_i create a new tableau identical to $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}$ and substitute every occurrence of λ'_j with λ'_i.

The application of this rule results to one or more new tableaux. In this case, the initial configuration is satisfiable if and only if one of the tableaux is open. If all tableaux close, then the initial configuration is not satisfiable.

The $(\geq)$-rule: Intuitively, if there exists a node labelled $(\geq_n r : \lambda)$ and λ has less than n direct R_r-successors, it is necessary to add enough new R_r-successors for λ in order to satisfy the constraint. The new successors that will be added must be distinct from each other and from λ. Notice that the addition of enough new successors can be achieved by adding only one successor every time. If there are still not enough successors, then the rule will be still applicable.

More formally, if there exists in B a node labelled $(\geq_n r : \lambda)$ and there exist no CLDS-labels $\lambda'_1, \ldots, \lambda'_n$ such that $R_r(\lambda, \lambda'_i) \in B$ $(1 \leq i \leq n)$ and $(\lambda'_i \neq \lambda'_j) \in B$ $(1 \leq i < j \leq n)$, then add to B two nodes $\langle l0, R_r(\lambda, \lambda')\rangle$ and $\langle l00, \lambda \neq \lambda'\rangle$, where λ' is a label that is new to the branch B.

The $(\geq)$-rule is a generating rule since it introduces a new label into the branch. On the other hand, the $(\leq)$-rule is a non-deterministic one. It also has to be made clear that these rules are a straightforward expression of the corresponding tableau rules for description logics. They have just been "translated" to fit into the CLDS framework.

It is also necessary to modify the definition of a closed branch. In addition to the conditions described in definition 52, a branch of an $\mathcal{ALCN}$-CLDS tableau is said to be closed if it contains a node labelled with a declarative unit $(\leq_n r : \lambda)$ and a node labelled $(\geq_m r : \lambda)$ with $n < m$.

We will now present some examples of $\mathcal{ALCN}$-CLDS tableaux. The first one demonstrates the use of at-most number restrictions, the second one focuses on the use of at-least restrictions, and the last one exemplifies how the non-adoption of the Unique Name Assumption is in a way more expressive in the a case of number restrictions.

Example 113. Assume that we want to check whether the $\mathcal{ALCN}$ concept

$$((\exists r.\mathsf{A}_1 \sqcap \exists r.\mathsf{A}_2) \sqcap (\exists r.\neg\mathsf{A}_1 \sqcap \leq_1 r))$$

is satisfiable w.r.t. the empty knowledge base. Intuitively, it is easy to see that this concept is not satisfiable because if there was an individual that belonged to the interpretation of the concept, then this individual would have at most one direct R_r-successor (due to the at-most number restriction) where A_1, A_2, and $\neg\mathsf{A}_1$ must hold. However, this would obviously lead to a contradiction.

We will now try to construct a complete $\mathcal{ALCN}$-CLDS tableau to check the satisfiability of this concept. The first steps of the construction of the corresponding tableau are presented in figure 10. In this tableau, there exist three direct R_r-successors of c_0: c_1, c_2, and c_3. However, none of them are asserted to be different. Therefore, according to the $\leq$-rule, we have to create a new tableau for each pair of direct R_r-successors of c_0. To be more specific:

- For the pair c_1, c_2 we create a new tableau where every occurrence of c_2 has been substituted by c_1. The new tableau is shown in figure 11(a). It has to be

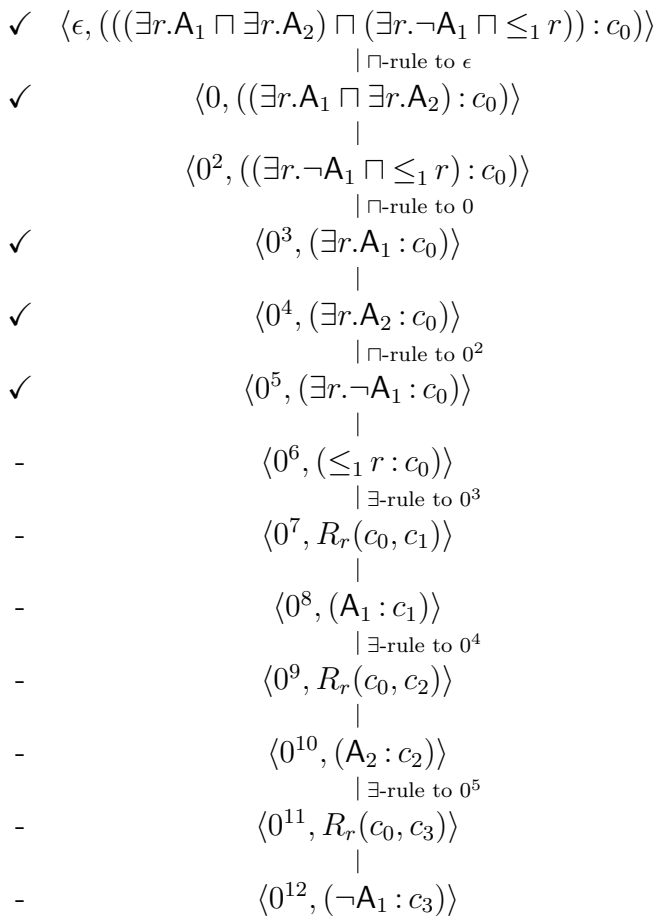


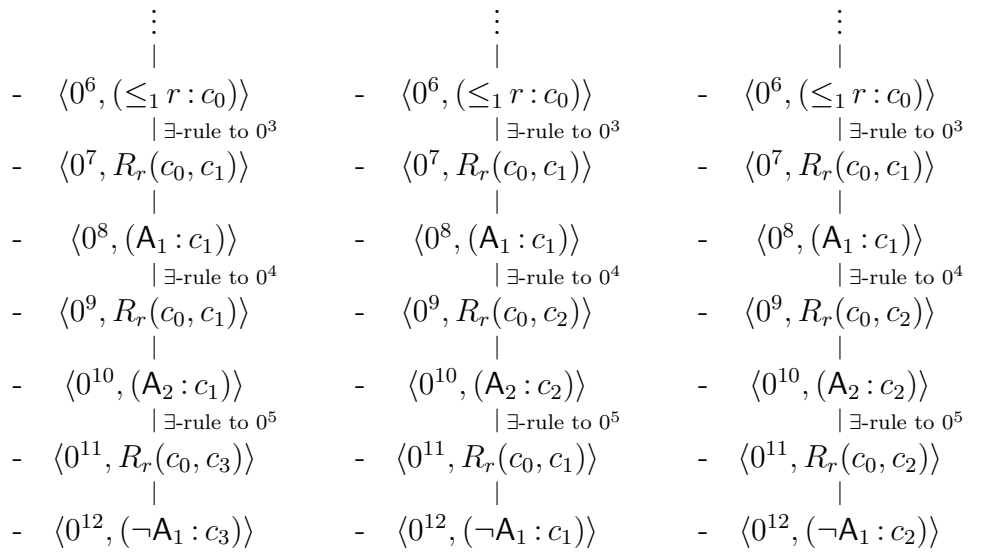
FIGURE 10. Tableau for example 113 before the application of the ≤-rule.

noted that in this figure we only show a part of the tableau that contains the aforementioned changes. The nodes that are not shown are identical to the corresponding nodes of the tableau of figure 10. We will use the notation $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_2/c_1]$ to refer to this tableau.

- For the pair c_1, c_3 a new tableau, $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_3/c_1]$, is constructed where every occurrence of c_3 has been substituted by c_1. The (partial) resulting tableau is presented in figure 11(b).
- Finally, the tableau of figure 11(c) is constructed for the pair of labels c_2, c_1. Every occurrence of c_2 has been substituted by c_1.

In order for the original concept to be satisfiable, al least one of the three aforementioned tableaux has to be open (once completed of course). However, $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_3/c_1]$ is closed. It remains to examine the tableaux for the other two pairs. $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_2/c_1]$ is not closed and the ≤-rule is again applicable since c_0 has two direct R_r-successors, c_1 and c_3. The resulting tableau, $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_2/c_1][c_3/c_1]$, is shown in figure 12(a) and is closed. Similarly, the application of the ≤-rule to tableau $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_3/c_2]$ results in the closed tableau $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_3/c_2][c_2/c_1]$ of figure 12(b).

Notice that the ≤-rule is non-deterministic since it explores several options in order to find a suitable model for the initial configuration. In the above example,



(a) Tableau $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_2/c_1]$: after the first application of the $\leq$-rule, c_2 has been replaced by c_1 in $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}$. This is an open tableau but not complete, since the $\leq$ rule is applicable.

(b) Tableau $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_2/c_1]$: after the first application of the $\leq$-rule, c_3 has been replaced by c_1. The tableau is closed because of nodes 0^8 and 0^{12}.

(c) Tableau $\mathcal{T}^{CLDS}_{\mathcal{ALCN}}[c_3/c_2]$: after the first application of the $\leq$-rule, c_3 has been replaced by c_2. The tableau is not closed yet and the $\leq$ rule is applicable.

FIGURE 11. Tableaux for example 113 after the first application of the $\leq$ rule.

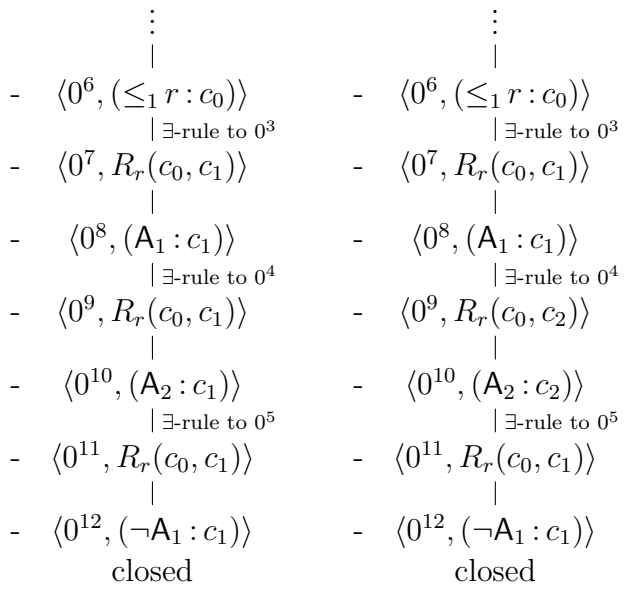
these alternatives are (i) c_1 and c_2 are mapped to the same element, and then c_3 is also mapped to the same element, (ii) c_1 and c_3 are mapped to the same element, and (iii) c_2 and c_3 are interpreted as the same element of the domain, and then c_2 is interpreted as this element as well. Compared to the other non-deterministic rule, the $\sqcup$-rule, it has to be said that $\leq$-rule changes a whole branch of the tableau, while $\sqcup$-rule only extends a branch is two different ways. This is why in the case of the $\sqcup$-rule we can create a tree, while in the case of the $\leq$-rule we have to create a set of tableaux, i.e. a forest.

Example 114. Assume that we want to check the satisfiability of the $\mathcal{ALCN}$ concept $\geq_2 r$. Obviously, this is a satisfiable concept. The corresponding tableau is shown in figure 13.

Example 115. We will now provide an example that makes clear why we chose not to adopt the Unique Name Assumption. Assume that we want to know whether the ABox $\mathcal{A} = \{r(a, b), r(a, c), \leq_1 r(a)\}$ is consistent. Under the UNA it obviously is inconsistent since a has two different successors b and c. On the other hand, if the UNA is not adopted, b and c can be interpreted as the same element of the domain and therefore $\mathcal{A}$ is not inconsistent. Therefore, if the UNA is not used, we are able to "hold more information". Moreover, one can argue that the UNA can be explicitly imposed by using inequality assertions.

Consider for example the $\mathcal{ALCN}$-CLDS tableau for the configuration

$$\mathcal{C} = \{R_r(c_a, c_b), R_r(c_a, c_c), (\leq_1 r : c_a)\}$$



(a) Tableau after the application of the $\leq$-rule to the tableau of figure 11(a): c_3 has been replaced by c_1. This is a closed tableau because of nodes 0^8 and 0^{12}.

(b) Tableau after the application of the $\leq$-rule to the tableau of figure 11(c): c_3 has been replaced by c_1. The tableau is closed because of nodes 0^8 and 0^{12}.

FIGURE 12. Tableaux for example 113 after the second application of the $\leq$ rule.

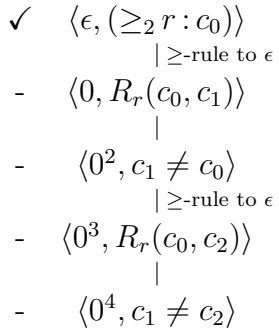
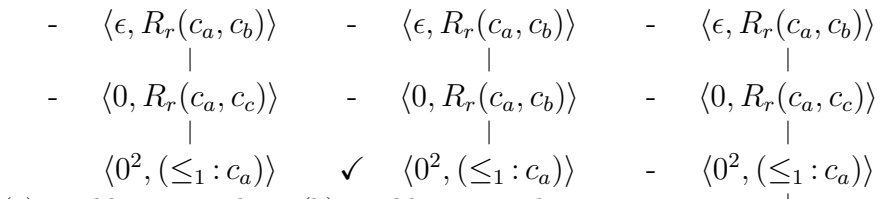


FIGURE 13. A complete tableau for example 114.

that corresponds to the aforementioned ABox $\mathcal{A}$. The corresponding tableau for $\mathcal{C}$ is presented in figure 14.

The soundness and completeness of the above $\mathcal{ALCN}$-CLDS tableau calculus can be easily proved with the same techniques that we have used so far. To be more specific, it is necessary to add in the extended labelling algebra axiom



(a) Tableau without UNA. The ≤-rule is applicable.

(b) Tableau without UNA. After the application of the ≤-rule in tableau 14(a).

(c) Tableau where the UNA is imposed by inequality assertions.

FIGURE 14. A complete $\mathcal{ALCN}$-CLDS tableau for the configuration of example 115.

schemata that will capture the semantics of the constructors for number restrictions:

$$\text{Ax-}(\leq)\quad \forall x \forall y_1 \ldots \forall y_{n+1} \Big([\leq_n r]^*(x) \rightarrow \big(\bigwedge_{1 \leq i \leq n+1} R_r(x, y_i) \rightarrow \bigvee_{1 \leq i < j \leq n+1} (y_i = y_j) \big) \Big)$$

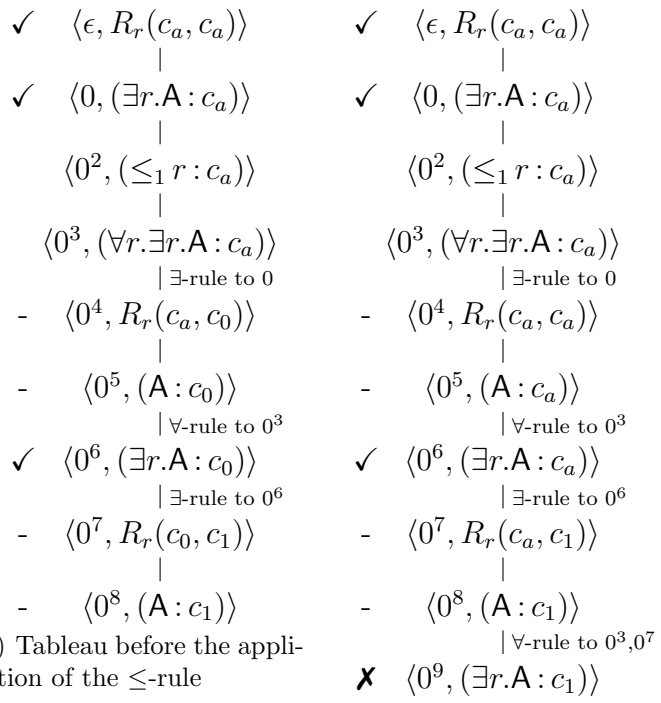
$$\text{Ax-}(\geq)\quad \forall x \exists y_1 \ldots \exists y_n \Big([\geq_n r]^*(x) \rightarrow \big(\bigwedge_{1 \leq i \leq n} R_r(x, y_i) \bigwedge_{1 \leq i < j \leq n} (y_i \neq y_j) \big) \Big)$$

The proof of termination is similar as well and it requires the use of blocking. In this case both the $(\exists)$ and the $(\geq)$ rule are generating rules, therefore, if a node is blocked none of those two rules can be applied to the node.

Example 116. [Example adapted from [4]] Assume that we want to check the satisfiability of the configuration

$$\mathcal{C} = \{r(a, a), (\exists r.\mathsf{A} : a), (\leq_1 r : a), (\forall r.\exists r.\mathsf{A} : a)\}.$$

The first steps of the construction of a tableau for this configuration are presented in figure 15(a). At this stage, c_a has two successors, c_a and c_0, and they have to be "merged" into one. Therefore, according to ≤-rule, c_0 has to be substituted by c_a. The resulting tableau is shown in figure 15(b). If we apply the ∀-rule then node 0^9 is added that is blocked because $S(c_1) = \{\mathsf{A}, \exists r.\mathsf{A}\} \subset S(c_)$. If we did not use blocking, then we could apply the ∃-rule to node 0^9 and obtain a successor of c_1. Then we could merge c_1 with c_a because c_1 is also a R_r-successor of c_a. This procedure would not terminate.



(a) Tableau before the application of the $\leq$-rule

(b) Tableau after the application of the $\leq$-rule: c_0 has been substituted by c_a. Now it is possible to apply the $\forall$-rule to nodes 0^3, 0^7.

FIGURE 15. Tableaux for example 116.

18. Extending the CLDS framework to include more dimensions

The basic idea behind Labelled Deductive Systems was to annotate every formula with a label that carries some "extra information". The next logical step was to associate a formula with more information, or in other words, with more than one labels.

Just to put this abstract description in a context, consider the following example (this is an informal example and the notions have not been properly formalized yet). Suppose *Happy* is a concept which describes that someone is happy. We could annotate this concept with the name of a person that is happy: $(Happy : MARY)$. However, Mary may not be always happy. In other words, we would like to be able to specify *when* Mary is happy. If we want to say that Mary is happy now, for example, we could write something like $(Happy : MARY, now)$.

In this section we will present an outline of how "multiple dimensions" can be represented in the CLDS framework. To be more specific, we will describe a CLDS for a description logic that has been extended with a temporal dimension.

18.1. Introduction in temporal Description Logics.

Generally, Description Logics are used to represent knowledge regarding a specific application domain. However, in many cases, this knowledge changes over time. This is the reason why several extensions of Description Logics with a temporal dimension have been proposed. For a survey of the most important temporal extensions of Description Logics reader is referred to [2].

In this section we will focus on a $\mathcal{ALCT}$, a description language that extends $\mathcal{ALC}$ with point-based temporal operators [21]. To be more specific, the available temporal operators are $\Diamond$ (sometime in the future), $\Box$ (always in the future), $\bigcirc$ (tomorrow, next time point), $\mathcal{U}$ (until), and U (reflexive until).

The $\mathcal{ALCT}$ concepts are the familiar $\mathcal{ALC}$ concepts plus the concepts that can be constructed using the temporal operators.

Definition 117 ($\mathcal{ALCT}$ Concept)**.** Every $\mathcal{ALC}$ concept is also an $\mathcal{ALCT}$ concept. Moreover, if C and D are $\mathcal{ALCT}$ concepts, then so are the following:

$$\Diamond\mathsf{C} \quad \Box\mathsf{C} \quad \bigcirc\mathsf{C} \quad \mathsf{C}\mathcal{U}\mathsf{D} \quad \mathsf{C}\mathsf{U}\mathsf{D}.$$

In general, temporal formulas are interpreted over flows of time. A flow of time is a pair $\mathcal{F} = \langle \mathcal{T}, < \rangle$, where $\mathcal{T}$ is a non-empty set of elements and $<$ is a strict total order on $\mathcal{T}$. This flow of time will be part of the $\mathcal{ALCT}$ interpretation. Informally, it will be used to interpret the temporal part of $\mathcal{ALCT}$ concepts.

Definition 118 ($\mathcal{ALCT}$ Interpretation)**.** An $\mathcal{ALCT}$ interpretation is a tuple $\mathcal{I} = \langle \mathcal{F}, \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\mathcal{F} = \langle \mathcal{T}, < \rangle$ is a flow of time, $\Delta^{\mathcal{I}}$ a non-empty set called the domain of the interpretation, and $\cdot^{\mathcal{I}}$ a function that maps atomic concepts to subsets of $\Delta^{\mathcal{I}} \times \mathcal{T}$ and atomic roles to subsets of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \times \mathcal{T}$.

As usual, the $\mathcal{ALCT}$ interpretation $\mathcal{I}$ is extended to handle the case of complex concepts. We will only present the interpretation of concepts that contain temporal operators, as the rest of them are interpreted exactly like in the case of

$\mathcal{ALCC}$ (see section 5.2).

$$
\begin{aligned}
(\Diamond \mathsf{C})^{\mathcal{I}} = \{(a,t) \in \Delta^{\mathcal{I}} \times \mathcal{T} \mid\ & \text{there exists a } t' \in \mathcal{T} \text{ such that} \\
& (t \leq t') \text{ and } (a,t') \in \mathsf{C}^{\mathcal{I}}\} \\
(\Box \mathsf{C})^{\mathcal{I}} = \{(a,t) \in \Delta^{\mathcal{I}} \times \mathcal{T} \mid\ & \text{for all } t' \in \mathcal{T} \text{ such that } (t \leq t') \\
& \text{it follows that } (a,t') \in \mathsf{C}^{\mathcal{I}}\} \\
(\bigcirc \mathsf{C})^{\mathcal{I}} = \{(a,t) \in \Delta^{\mathcal{I}} \times \mathcal{T} \mid\ & \text{there exists a } t' \in \mathcal{T} \text{ such that} \\
& (t < t') \text{ and } (a,t') \in \mathsf{C}^{\mathcal{I}} \\
& \text{and there exists no } t'' \in \mathcal{T} \text{ such that } t < t'' < t'\} \\
(\mathsf{C} \mathcal{U} \mathsf{D})^{\mathcal{I}} = \{(a,t) \in \Delta^{\mathcal{I}} \times \mathcal{T} \mid\ & \text{there exists a } t' \in \mathcal{T} \text{ such that} \\
& (t < t') \text{ and } (a,t') \in \mathsf{D}^{\mathcal{I}} \\
& \text{and for all } t'' \in \mathcal{T}, t < t'' < t' \text{ implies } (a,t'') \in \mathsf{C}^{\mathcal{I}}\} \\
(\mathsf{C} \mathsf{U} \mathsf{D})^{\mathcal{I}} = \{(a,t) \in \Delta^{\mathcal{I}} \times \mathcal{T} \mid\ & \text{there exists a } t' \in \mathcal{T} \text{ such that} \\
& (t \leq t') \text{ and } (a,t') \in \mathsf{D}^{\mathcal{I}} \\
& \text{and for all } t'' \in \mathcal{T}, t \leq t'' < t' \text{ implies } (a,t'') \in \mathsf{C}^{\mathcal{I}}\}
\end{aligned}
$$

18.2. **$\mathcal{ALCT}$-CLDS.** First of all, we will define the language of the $\mathcal{ALCT}$-CLDS. Remember that a CLDS-language is a pair $\langle \mathcal{L}_{\#}, \mathcal{L}_L \rangle$ (see definition 29). In the case of $\mathcal{ALCT}$-CLDS, $\mathcal{L}_{\#}$ is $\mathcal{ALCT}$. The interesting part, however, is the labelling language. Intuitively, we need two labelling languages, one that will be used for labels that contain information about individuals, and one that will be used for labels that contain time information. Therefore, we adjust the definition of the CLDS-language in order to take both labelling languages into consideration.

Definition 119 ($\mathcal{ALCT}$-CLDS language)**.** The language of an $\mathcal{ALCT}$-CLDS is a tuple $\langle \mathcal{ALCT}, \mathcal{L}_{ind}, \mathcal{L}_t \rangle$, where $\mathcal{L}_{ind}\langle \mathcal{P}, \emptyset, \mathcal{C} \rangle$ is a first-order language that coincides with $\mathcal{L}_L^{\mathcal{ALC}}$ (the labelling language in the case of $\mathcal{ALC}$-CLDS), and $\mathcal{L}_t\langle \{<\}, \emptyset, \mathcal{C}_t \rangle$ is a first-order language whose set of predicate names and constant names are disjoint from those of $\mathcal{L}_{ind}$.

In what follows will use the symbol c to refer to the elements of $\mathcal{C}$ (i.e. the constants of the labelling language $\mathcal{L}_{ind}$), and the symbol t to denote the constants of the labelling language $\mathcal{L}_t$.

As usually, labels are defined as the ground terms of the labelling languages. Since we have two labelling languages, there exist two types of labels. Notice that because the labelling languages contain no function symbols, the labels of $\mathcal{L}_{ind}$ are just the constants of $\mathcal{L}_{ind}$, and similarly, the labels of $\mathcal{L}_t$ are the constants of $\mathcal{L}_t$. Moreover, the set of predicates of $\mathcal{L}_{ind}$ does not contain binary predicates as it was the case for $\mathcal{ALC}$-CLDS, but instead it consists of ternary predicate symbols. However, each predicate symbol of $\mathcal{P}$ still corresponds to an $\mathcal{ALCT}$ role. Thus we will use the same notation for elements of $\mathcal{P}$ that we employed in the case of $\mathcal{ALC}$-CLDS: R_r is the ternary predicate that corresponds to the $\mathcal{ALCT}$ role r.

We can now adapt the definition of CLDS-formulas in the case of the $\mathcal{ALCT}$-CLDS.

TABLE 3. Axioms of $\text{Mon}(\mathcal{L}_{\#}, \mathcal{L}_L)$ for the $\mathcal{ALCT}$-CLDS. $t < t' < t''$ is used as an abbreviation for $(t < t') \wedge (t' < t'')$.

Name	Axiom
Ax-$\sqcap$	$\forall x \forall t \left([\mathsf{C}_1 \sqcap \mathsf{C}_2]^* (x,t) \rightarrow ([\mathsf{C}_1]^* (x,t) \wedge [\mathsf{C}_2]^* (x,t)) \right)$
Ax-$\sqcup$	$\forall x \forall t \left([\mathsf{C}_1 \sqcup \mathsf{C}_2]^* (x,t) \rightarrow ([\mathsf{C}_1]^* (x,t) \vee [\mathsf{C}_2]^* (x,t)) \right)$
Ax-$\neg$	$\forall x \forall t \left([\neg \mathsf{C}]^* (x,t) \leftrightarrow \neg [\mathsf{C}]^* (x,t) \right)$
Ax-$\forall$	$\forall x \forall t \left([\forall r.\mathsf{C}]^* (x,t) \rightarrow \forall y \left(R_r (x,y,t) \rightarrow [\mathsf{C}]^* (y,t) \right) \right)$
Ax-$\exists$	$\forall x \forall t \left([\exists r.\mathsf{C}]^* (x,t) \rightarrow \exists y \left(R_r (x,y,t) \wedge [\mathsf{C}]^* (y,t) \right) \right)$
Ax-$\Diamond$	$\forall x \forall t \left([\Diamond \mathsf{C}]^* (x,t) \rightarrow \exists t' \left((t \leq t') \wedge [\mathsf{C}]^* (x,t') \right) \right)$
Ax-$\Box$	$\forall x \forall t \left([\Diamond \mathsf{C}]^* (x,t) \rightarrow \forall t' \left((t \leq t') \rightarrow [\mathsf{C}]^* (x,t') \right) \right)$
Ax-$\bigcirc$	$\forall x \forall t \left([\bigcirc \mathsf{C}]^* (x,t) \rightarrow \exists t' \left((t < t') \wedge [\mathsf{C}]^* (x,t') \wedge \neg \exists t'' \left(t < t'' < t' \right) \right) \right)$
Ax-$\mathcal{U}$	$\forall x \forall t \left([\mathsf{C} \mathcal{U} \mathsf{D}]^* (x,t) \rightarrow \exists t' \left((t \leq t') \wedge [\mathsf{C}]^* (x,t') \wedge \forall t'' \left(t \leq t'' < t' \right) \rightarrow [\mathsf{C}]^* (x,t'') \right) \right)$
Ax-U	$\forall x \forall t \left([\mathsf{C} \mathsf{U} \mathsf{D}]^* (x,t) \rightarrow \exists t' \left((t < t') \wedge [\mathsf{C}]^* (x,t') \wedge \forall t'' \left(t < t'' < t' \right) \rightarrow [\mathsf{C}]^* (x,t'') \right) \right)$

Definition 120 ($\mathcal{ALCT}$-CLDS formulas)**.** An $\mathcal{ALCT}$-CLDS formula is either (i) a declarative unit of the form $(\mathsf{C} : c, t)$ where C is an $\mathcal{ALCT}$ concept, c is a label of $\mathcal{L}_{ind}$, and t is a label of $\mathcal{L}_t$, or (ii) a Relation-literal of the form $t < t'$, or (iii) a Relation-literal of the form $R_r(c_1, c_2, t)$.

A configuration for an $\mathcal{ALCT}$-CLDS is defined as a set of $\mathcal{ALCT}$-CLDS formulas. It can also be defined as a set of declarative units, and a set of Relation-predicates (known as the diagram).

The labelling algebra of an $\mathcal{ALCT}$-CLDS is a first-order theory written in $\mathcal{L}_{ind} \cup \mathcal{L}_{ind}$. It has to be noted that by the union of two languages L_1 and L_2 we mean the union of their well-formed formulas:

$$L_1 \cup L_2 = \{\phi \mid \phi \text{ is an } L_1 \text{ formula or } \phi \text{ is an } L_2 \text{ formula}\}.$$

In the general case, the labelling algebra would be a theory written in the union of the semi-extended labelling languages. However, as in the case of $\mathcal{ALC}$-CLDS, the semi-extended labelling language coincides with the labelling language itself.

Regarding the translation of a declarative unit into first-order logic, since every declarative unit uses two labels (and not one) it will be translated as a *binary* (instead of unary) predicate. For example, if $(\mathsf{C} : c, t)$ is an $\mathcal{ALCT}$-CLDS declarative unit, then its first-order translation will be $[\mathsf{C}]^*(c, t)$. It is now possible to define the extended labelling language of the CLDS as the semi-extended labelling language together with the set $\{[\mathsf{C}_0]^*, \ldots, [\mathsf{C}_n]^*, \ldots\}$ of binary predicate symbols (as usual, $\mathsf{C}_0, \ldots, \mathsf{C}_n, \ldots$ is considered to be an enumeration of $\mathcal{ALCT}$-concepts).

The extended labelling algebra of the $\mathcal{ALCT}$-CLDS is a theory written in the extended labelling language. The axioms of the extended labelling algebra for the $\mathcal{ALCT}$-CLDS are presented in table 3. Like in the case of the $\mathcal{ALC}$, these axioms "capture" the translation of $\mathcal{ALCT}$ concepts into first-order logic. The first five axioms of table 3 are similar to the axioms of the extended labelling algebra in the case of $\mathcal{ALC}$-CLDS. The only change is that now the predicate symbol $[\mathsf{C}]^*$ is a binary and not a unary predicate. The rest axioms cover the cases of the temporal operators. It can be easily verified that they express the semantics of the temporal operators as discussed in the previous section.

A semantic structure for an $\mathcal{ALCT}$-CLDS is a first-order structure for the corresponding extended labelling algebra.

19. $\mathcal{ALC}$-CLDS and Logic Programming

The combination of Description Logics with Logic Programming has been widely studied. Such a combination would be very useful, especially in the area of the Semantic Web, as it would allow reasoners (or agents) that use logic programming to reason about information represented in some description language [22]. In this section, we will focus on *Description Logic Programs* that were introduced in [18].

Description Logic Programs are defined as the intersection of Description Logics and Horn logic programs. A Horn logic program is a set of rules of the form

$$H \leftarrow B_1 \wedge \ldots \wedge B_n \quad (n \geq 0)$$

where H, B_1,...,B_n are atoms. H is referred to as the *head* of the rule, while the $B_1 \wedge \ldots \wedge B_n$ as the body of the rule. Notice that negation is not allowed in the head nor in the body of the rules.

The authors of [18] specify a fragment of description logics that can be expressed in terms of Horn logic programming rules. Towards that direction, they first specify the type of TBox and ABox axioms that can be expressed as rules of Horn logic programs, and then they introduce a recursive mapping T that maps axioms of the TBox and the ABox to the corresponding logic programming rules.

The TBox axioms that can be described as Horn logic programming rules are axioms of the form $\mathsf{C} \dot{\sqsubseteq} \mathsf{D}$ where

- C is an atomic concept, or a complex concept of the form $\mathsf{C}_1 \sqcap \mathsf{C}_2$, or $\exists r.\mathsf{C}_1$, and
- D is either an atomic concept, or a complex concept of the form $\mathsf{D}_1 \sqcap \mathsf{D}_2$, or $\mathsf{D}_1 \sqcup \mathsf{D}_2$, or $\forall r.\mathsf{D}_1$.

These conditions are summarized in table 4. Informally, every such axiom will be mapped to a rule of the form $\mathsf{D} \leftarrow \mathsf{C}$. It is not hard to see that if C and D fulfil the requirements of table 4, then the rule $\mathsf{D} \leftarrow \mathsf{C}$ can be written as one or more Horn logic program rules. In what follows, if a concept C is of one of the forms specified in the second column of table 4 we will say that is a *Body-appropriate* rule, while if it belongs to one of the forms specified in the third column the table, we will say it is a *Head-appropriate* rule.

If C, D are complex concepts, A an atomic concept, and r an atomic role, the mapping T is defined as follows:

Table 4. Allowed types for the concepts C and D of a TBox axiom $\mathsf{C} \dot{\sqsubseteq} \mathsf{D}$ in order for this axiom to be expressed as a Horn logic program rule.

Concept Type	C (Body-appropriate)	D (Head-appropriate)
Atomic	✓	✓
Conjunction	✓	✓
Disjunction	✓	✗
Universal	✓	✗
Existential	✗	✓
Negation	✗	✗

$$
\begin{aligned}
T(\mathsf{C} \dot{\sqsubseteq} \mathsf{D}) &= T_h(\mathsf{D}, x) \leftarrow T_b(\mathsf{C}, x) \\
T_h(\mathsf{A}, x) = T_b(\mathsf{A}, x) &= \mathsf{A}(x) \\
T_h(\mathsf{C} \sqcap \mathsf{D}, x) &= T_h(\mathsf{C}, x) \wedge T_h(\mathsf{D}, x) \\
T_h(\forall r.\mathsf{C}, x) &= T_h(\mathsf{C}, x) \leftarrow r(x, y) \\
T_b(\mathsf{C} \sqcap \mathsf{D}, x) &= T_b(\mathsf{C}, x) \wedge T_b(\mathsf{D}, x) \\
T_b(\mathsf{C} \sqcup \mathsf{D}, x) &= T_b(\mathsf{C}, x) \vee T_b(\mathsf{D}, x) \\
T_b(\forall r.\mathsf{C}, x) &= T_b(\mathsf{C}, y) \leftarrow r(x, y)
\end{aligned}
$$

At this point, an example is necessary to better understand how this mapping works.

Example 121. Assume that a TBox $\mathcal{T}$ contains the following axioms:

$$\mathcal{T} = \{\mathsf{A}_1 \sqcap \mathsf{A}_2 \dot{\sqsubseteq} \mathsf{A}_3, \mathsf{A}_1 \dot{\sqsubseteq} \forall r.\mathsf{A}_2, \exists r.\mathsf{A}_2 \dot{\sqsubseteq} \mathsf{A}_3\}.$$

First of all, every axiom is of the form specified in table 4. Therefore, it can be expressed as a set of Horn logic program rules. The application of T to the TBox axioms, results in the following logic programming rules:

$$
\begin{aligned}
\mathsf{A}_3(x) &\leftarrow \mathsf{A}_1(x) \vee \mathsf{A}_2(x) \\
(\mathsf{A}_2(y) \leftarrow r(x, y)) &\leftarrow \mathsf{A}_1(x) \\
\mathsf{A}_3(x) &\leftarrow \mathsf{A}_2(y) \wedge r(x, y)
\end{aligned}
$$

This can be written equivalently as the following set of Horn logic programming rules:

$$
\begin{aligned}
\mathsf{A}_3(x) &\leftarrow \mathsf{A}_1(x) \\
\mathsf{A}_3(x) &\leftarrow \mathsf{A}_2(x) \\
\mathsf{A}_2(y) &\leftarrow r(x, y) \wedge \mathsf{A}_1(x) \\
\mathsf{A}_3(x) &\leftarrow \mathsf{A}_2(y) \wedge r(x, y)
\end{aligned}
$$

What the above method has accomplished is to express a TBox (not any given one, but one of the specific form described) to a set of rules for logic programming.

The authors of [18] consider also ABoxes that contain role assertions and concept assertions of the form $\mathsf{C}(a)$, where C a head-appropriate concept. The mapping T is extended in order to map ABox assertions to facts (i.e. logic programming rules with the empty body). This can be done since C is assumed to be a head-appropriate concept, and therefore it can appear to the head of a rule.

$$
\begin{aligned}
T(\mathsf{C}(a)) &= T_h(\mathsf{C}, a) \\
T(r(a, b)) &= r(a, b)
\end{aligned}
$$

This approach has many limitations. First of all the TBox must contain axioms of the specific form described above, and moreover, it must be acyclic.

Example 122. Consider for example the ABox

$$\mathcal{A} = \{(\mathsf{A}_1 \sqcap \mathsf{A}_2)(a), r(a, b), r(b, c)\}.$$

Concept $\mathsf{A}_1 \sqcap \mathsf{A}_2$ is a head-appropriate concept. The corresponding Horn logic program rule is:

$$(\mathsf{A}_1 \sqcap \mathsf{A}_2)(a) \leftarrow$$

which can be equivalently written as:

$$\mathsf{A}_1(a) \leftarrow$$
$$\mathsf{A}_2(a) \leftarrow$$

The rules for the role assertions are

$$r(a, b) \leftarrow \qquad \text{and} \qquad r(b, c) \leftarrow .$$

Since we have the ABox and the TBox as in the form of Horn logic programming rules, it is now possible to run an LP reasoning engine to answer questions w.r.t. this knowledge base. The main kind of supported queries are atom queries, i.e. if a belongs to the interpretation of a concept. It is also possible to ask for all the elements that belong to the interpretation of a concept.

Example 123. Given the TBox and the ABox defined above, and the corresponding logic programming rules, it is easy to write a Prolog program for this knowledge base.

```
a_3(X):-a_1(X).
a_3(X):-a_2(X).
a_2(Y):-a_1(X),r(X,Y).
a_3(X):-a_2(Y),r(X,Y).
a_1(a).
a_2(a).
r(a,b).
r(b,c).
```

Now we can run several queries. For example if we want to know whether b is an instance of C_3, we run the query, `a_3(b).` and the answer is `Yes`. However the answer is `No` to the query `a_3(c).`

Moreover, since we have already expressed in a previous section some axioms of the RBox and some role constructors as Horn clauses, we can use them as logic programming rules.

Example 124. Consider again the code mentioned in the previous example, extended with one more rule which asserts that r is transitive:

```
r(X,Z):-r(X,Y),r(Y,Z).
```

This time as expected the answer to the query `a_3(c).` is `Yes`.

20. Conclusions and Future Work

To conclude, we have presented a tableau CLDS for the description language $\mathcal{ALC}$ when reasoning w.r.t. a non-empty knowledge base and when the labelling algebra contains universally quantified Horn clauses. The formalization of tableau CLDS and the machinery for the proofs of soundness, completeness, and termination of the tableau algorithm can be generalized and used for tableau CLDS for other logics.

In the case of CLDS for modal logics (MLDS), this formalization was in a sense "more expressive" than standard modal logics. To be more specific, it is a well-known fact there is no modal formula that can characterize irreflexive frames. However, as it is shown in [23], a MLDS is applicable to any modal frame logic, including irreflexive frames. In the case of Description Logics, on the other hand, one has the liberty to add any role and concept constructor he/she wishes, and any role axiom in the RBox. Therefore, it is possible to explicitly declare that a role is irreflexive, by inserting an appropriate axiom in the RBox.

Regarding future work, it would be interesting to formally present a tableau CLDS for another logic, for example for hybrid logics, using the formalization introduced here. Moreover, regarding the tableau CLDS for description logics, the complexity of the tableau algorithms presented here has to be investigated. It has to be noted that our objective was not to present the most efficient tableau algorithms, thus many improvements can be done in this field. Another direction for future research is to consider more complex labelling algebras and more complex labelling languages. Finally, it would be challenging to allow CLDS-labels to be terms of the labelling language and not only ground terms. In this case, we would have to make use of free-variable tableaux.

References

[1] A unified compilation style natural deduction system for modal, substructural and fuzzy logics. In *Discovering World with Fuzzy Logic: Perspectives and Approaches to Formalization of Human-consistent Logical Systems*. Springer-Verlag.

[2] Alessandro Artale and Enrico Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):171–210, 2000.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2003.

[4] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69, 2000.

[5] Patrick Blackburn, Johan F. A. K. van Benthem, and Frank Wolter. *Handbook of Modal Logic, Volume 3 (Studies in Logic and Practical Reasoning)*. Elsevier Science Inc., New York, NY, USA, 2006.

[6] Thomas Bolander and Patrick Blackburn. Termination for hybrid tableaus. *J. Log. Comput.*, 17(3):517–554, 2007.

[7] Thomas Bolander and Torben Braüner. Tableau-based decision procedures for hybrid logic. *J. Log. Comput.*, 16(6):737–763, 2006.

[8] Krysia Broda. A decidable CLDS for some propositional resource logics. In *Computational logic: logic programming and beyond: essays in honour of Robert A.Kowalski, Part II*, volume 2408, pages 135–159. Springer, 2002.

[9] Krysia Broda, Marcelo Finger, and Alessandra Russo. Labelled natural deduction for substructural logics. *Logic Journal of the IGPL*, 7(3):283–318, 1999.

[10] Krysia Broda, Dov M. Gabbay, Luis C. Lamb, and Alessandra Russo. *Compiled Labelled Deductive Systems: A Uniform Representation of Non-Classical Logics*. Research Studies Press, 2004.

[11] Krysia Broda, Dov M. Gabbay, Luis C. Lamb, and Alessandra M. Russo. Labelled natural deduction for conditional logics of normality. *Logic Journal of the IGPL*, 10(2):123–163, 2002.

[12] Krysia Broda and Alessandra Russo. Compiled Labelled Deductive Systems for Access Control (2005). In *We Will Show Them: Essays in Honour of Dov Gabbay*, volume 1, pages 309–338. College Publications, 2005.

[13] Krysia Broda and Alessandra Russo. A tableau compiled labelled deductive system for hybrid logic. Automated Reasoning Workshop ARW07, 2007. Extended Abstract.

[14] Marcello D'Agostino, Dov M. Gabbay, Reiner Hähnle, and Joachim Posegga, editors. *Handbook of Tableau Methods*. Springer, 1999.

[15] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. *Inf. Comput.*, 134(1):1–58, 1997.

[16] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. pages 191–236, 1997.

[17] Dov M. Gabbay. *Labelled Deductive Systems*, volume 1. Oxford Clarendon Press, 1996.

[18] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 48–57, New York, NY, USA, 2003. ACM.

[19] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, 2006.

[20] Ian Horrocks and Ulrike Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

[21] Schild Klaus. Combining terminological logics with tense logic. In *EPIA '93: Proceedings of the 6th Portuguese Conference on Artificial Intelligence*, pages 105–120, London, UK, 1993. Springer-Verlag.

[22] Boris Motik and Riccardo Rosati. A faithful integration of description logics with logic programming. In *IJCAI*, pages 477–482, 2007.

[23] Alessandra Russo. *Modal logics as labelled deductive systems*. PhD thesis, Department of Computing, Imperial College London, 1996.
[24] Manfred Schmidt-Schaubß and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.