南 开 大 学

本科生学年论文（设计）

中文题目：描述逻辑 ALC 及其 Tableau 算法

外文题目：Description Logic ALC and its Tableau algorithm

学　　号：

姓　　名：

年　　级：

专　　业：

系　　别：

学　　院：哲学院

指导教师：

完成日期：2020.4.30

摘　要

描述逻辑作为基于逻辑的知识表示语言和推理机制，致力于以形式化、结构化且易于理解的方式对应用域的知识进行呈现。形式上，描述逻辑利用概念描述对知识进行表示；语义上，描述逻辑与经典一阶逻辑大体一致。

本文所介绍的 $\mathcal{ALC}$ 是最基本的描述逻辑语言，它包含了基本的布尔算子：$\sqcap$，$\sqcup$，$\neg$，以及量词：$\exists$，$\forall$。$\mathcal{ALC}$ 具有一定的表达能力和推理能力，在此基础之上，本文详细介绍了 Tableau 推理方法，尤其是介绍了 $\mathcal{ALC}$ 的 Tableau 算法及其正确性证明。

$\mathcal{ALC}$ 作为基础语言可以通过增加概念构造子的方式提升语言表达力，同步地，Tableau 算法也需要进行适当的优化以保证其正确性以及运行效率。

关键词：描述逻辑；知识表示；描述语言 $\mathcal{ALC}$；概念描述；Tableau 方法；算法正确性；表达力

Abstract

As a logic-based knowledge representation language and reasoning mechanisim, description logics(DLs) are dedicated to represent the knowledge of an application domain in a formalized, structured and well-understood way. Formally, DLs use concept descriptions to represent knowledge; Semantically, DLs are equipped with the same semantics as that of classical first-order-logic.

This article introduces the most basic description logic language $\mathcal{ALC}$, which contains the basic Boolean operators: $\sqcap$, $\sqcup$, $\neg$, and quantifiers: $\exists$, $\forall$. $\mathcal{ALC}$ possesses certain expressive power and reasoning ability. On this basis, this article introduces Tableau reasoning technique in detail, particularly the Tableau algorithm for $\mathcal{ALC}$ and its correctness proof.

$\mathcal{ALC}$ as the basic DL language is able to raise its expressivity by adding concept constructors. Synchronously, tableau algorithm also needs to be optimized appropriately for the sake of its correctness and operating efficiency.

Keywords: Description logic; Knowledge representation; Description language $\mathcal{ALC}$; Concept description; Tableau technique; Algorithm correctness; Expressive power

目录

描述逻辑 $\mathcal{ALC}$ 及其 Tableau 算法

1 描述逻辑概述

描述逻辑[①](Description Logic) 是以结构化、易于理解的方式表示应用领域知识的一个知识表示语言和推理机制。其发展可以追溯到 20 世纪 80 年代末，描述逻辑是基于逻辑的知识表示与推理的活跃研究领域，在知识密集的信息系统中有着广泛的应用。

因此，要理解描述逻辑是什么、能够做什么，首先，我们就应该对知识表示有所了解。在下文中我们将对一些概念进行定义，例如："人工智能"，"知识"，"表示" 等。有些概念或许在哲学上有着历史久远的分歧，但在本文中将回避争议，主要服务于对知识表示以及描述逻辑的核心思想的理解。

1.1 描述逻辑与知识表示

人工智能[②](Artificial Intelligence)，是指通过计算的方式实现对智能行为的研究。知识表示和推理则是作为人工智能的一部分，将思维作为一个计算过程，对代理 (agent) 如何使用其所知信息来决定其行动进行研究。

知识[③](Knowledge)，是指知识主体 (knower) 与命题 (proposition) 之间的关系。知识主体可以是人，也可以是代理；命题是用简单陈述句表达的想法。如："约翰知道玛丽会来参加派对。" 其中约翰是知识主体，"玛丽会来参加派对" 则是命题。当我们说约翰知道某事时，实质上是说约翰拥有或形成了某种判断。当然，根据命题的性质，这些判断可能为真，也可能为假。但对于主体来说，这是指约翰相信 (believe) 世界是以一种形式存在，而不是以另一种形式存在。

表示[④](Representation)，是指表示者 (representor) 对被表示者的表征。表示者通常更具体、更直接、更易获得。比如，在加号上画一个圆圈可以代表女性的抽象概念，人民代表可以代表一定选区的人民。本文主要讨论的表示者是形式符号，也就是从某种预先给定的字母表中形成的字符或字符串。通常情况下，符号比被其表征的更容易识别、区分、展示以及处理。

知识表示[⑤](Knowledge representation)，则是关于使用形式符号来表示被某些假定的代理所相信的命题集合的研究。一个基于知识的系统（或代理）它的行动取决于其对世界的看

[①] "Description logics (DLs) are a family of knowledge representation languages that can be used to represent knowledge of an application domain in a structured and well-understood way."Knowledge Representation And Reasoning, P1

[②] "One definition of Artificial Intelligence(AI) is that it is the study of intelligent behavior achieved through computational means." Knowledge Representation And Reasoning, P1

[③] "…knowledge is a relation between a knower, like John, and a proposition, that is, the idea expressed by a simple declarative sentence, like' Mary will come to the party.'" Knowledge Representation And Reasoning, P2

[④] "…representation is a relationship between two domains, where the first is meant to 'stand for' or take the place of the second." Knowledge Representation And Reasoning, P3

[⑤] "Knowledge representation, then, is the field of study concerned with using formal symbols to represent a collection of propositions believed by some putative agent." Knowledge Representation And Reasoning, P4

法，但是，通常我们不会将一个代理所相信的全部命题都表示出来，因为这很可能是无穷的。与此同时，单纯的事实的堆积也并不会帮助我们获得“显而易见”的事实，比如：给出“所有人都会死”以及“苏格拉底是人”这两个命题，我们自然期望得到“苏格拉底会死”这一命题。这通过利用推理的逻辑后承概念是很容易实现的，但在一个没有推理工具的数据库系统却难以实现。因此，出于对知识库规模的控制以及对更高效率的追求，推理负责在系统中计算知识库的逻辑后承。

上文说到：知识表示与推理是关于代理如何使用它所知道的来决定应当做什么。一个知识表示系统的构造是为了实现一定的“目标”的，比如人工智能 AlphaGo，它的目标是实现围棋制胜，它就会基于对棋局状况分析的知识来决定其策略与行动。如果我们基于符号化的表示来构建系统以实现一定目标，我们就称这样的系统为基于知识的系统 (knowledge-based systems)，包含这些符号化表示的则称为知识库 (knowledge bases)。根据基于知识的系统的设计，它能够被“告知”关于世界的事实，并对其行动做出相对应的调整。即，用户可以通过对系统增添新的知识或信念以扩张或变更系统的行为。并且，知识库中声明性的事实可以满足广泛的潜在用途。例如：一个地区的交通路线图的知识库系统可以用于对不同时间段车流量的计算，也可以为城市规划建设提供相关信息。另外，知识库的符号化结构表达使扩张、纠错以及对系统的解释更容易完成。

逻辑[①]是对后承关系的研究——语言、真值条件以及推理规则。因此，据其所涉的重要概念，它对知识表示系统的重要性不言而喻。其中，形式符号逻辑作为基于知识的系统的工具尤其适合。在实践中，许多知识表示语言都是以经典的一阶逻辑 (First Order Logic，以下简称 FOL) 作为基点，讨论其子集或超集。后文中将讨论的描述逻辑 $\mathcal{ALC}$ 则正是一阶逻辑的一个可判定的片段。

1.2 描述逻辑的理论特征

描述逻辑（以下简称 DL）作为基于逻辑的知识表示语言，一方面它不同于早期的非基于逻辑的知识表示形式，例如语义网以及框架系统：它们以特设 (ad hoc) 数据结构的形式表示知识，而且即使借用了 FOL 的语义，其推理仍然是通过特设程序对数据结构进行操作来实现。即，对同一应用域的知识可以有多种的表示方法，知识的表示方法又直接影响相应的推理方式与效率。并且非基于逻辑的知识表示系统通常是面对某一特定的、具体的研究领域进行开发，具有一定的针对性及局限性，因此非基于逻辑的知识表示系统之间的表示方式与推理方法不具有互通性。而 DL 所具备的形式化、基于逻辑的语义（塔斯基式语义），使得其推理能够通过证明逻辑后承这一普适方式得到实现，正是这一点使其具有“基于逻辑”的特征。另一方面，描述逻辑之所以被称其为描述逻辑，是由于其论域 (domain) 中的元素是以概念描述 (concept description) 的形式被表达，即，利用特定 DL 语言中的概念和关系构造算子，由原子概念 (一元谓词) 和原子关系 (二元谓词) 来构建表达式，这将在第二部分中具体说明。

DL 要在基于知识的应用程式设计中发挥作用，也就是 DL 语言要在 KR 系统中提供一种定义知识库的语言以及对其进行推理的工具。其实现基于两个方面的要求：一是对知识库的精确刻画，包括对此系统中指定知识类型的刻画，以及对此系统所提供的推理服务的明确定义。二是为用户提供丰富的开发环境，使用户与系统的交互更有效率。

对第一项要求的实现，受益于语义网和框架的历史努力——“知识表示的功能方法”，

[①] “···logic is the study of entailment relations—languages, truth conditions, and relied of inference.” Knowledge Representation And Reasoning, P11

其主要思想是对知识库提供的功能给出精确规范，特别是由知识库进行的推理——独立于任何具体系统的实现进行研究。在实践中，推理系统的功能描述是通过所谓的 “Tell & Ask” 接口具体表现的。这样的接口指定了支持知识库构造的操作 (Tell 操作) 和允许从知识库获取信息的操作 (Ask 操作)。一个典型的 DL 知识库通常由两部分组成：一个 TBox 与一个 ABox。TBox 包含以分类学 (taxonomy) 的形式构成的内涵性知识，由描述概念普遍性质的声明组成。内涵性知识通常被认为是不会改变的，即不受时间影响的。且根据组成分类学概念的包含关系的自有性质，TBox 通常具有格状结构，从而为在知识库中的推理提供方便。ABox 则包含外延知识（也称断言知识），即包含特定论域中关于个体的知识。外延性知识通常是偶然的，或依赖于具体情形的。更为详细的介绍同样将在第二部分中呈现。

对于第二项要求，基于 DL 应用开发的一个重要元素是 KR 系统的易用性：既要使系统表达力与可用性足够满足用户的使用需要，同时又要降低系统的使用门槛。这也是当代 DL-KRS 研究的焦点，即在保证提供完全性算法的前提下，最大限幅提高 DL 语言的表达力。根据这一理论基础，DL 可以作为普遍工具在众多领域实现应用开发，如：软件工程、配置、医学信息学、自然语言处理等。

以上，我们对描述逻辑及知识表示的具体内涵与理论特征进行了概述。下面，在文章的第二部分，我们将选取一个特定的描述语言 $\mathcal{ALC}$ 进行介绍，以此为根基对描述逻辑的核心元素及构成方式进行展示。在第三部分，我们将引入 Tableau 推理方法，说明此方法的基本原理并基于此方法为 $\mathcal{ALC}$ 提供一种判定算法，并对其正确性进行详细证明。最后，在文章的第四部分，我们将对 $\mathcal{ALC}$ 语言以及 Tableau 方法的特征进行总结概括。

2 描述逻辑 $\mathcal{ALC}$ 的语言和模型

本文选取介绍的语言 $\mathcal{ALC}$(Attributive concept Language with Complements) 是 DL 最基本的描述语言，其他描述语言都是基于 $\mathcal{ALC}$ 的扩展。

从第一部分，我们已经知道：描述语言（description language）是一个描述逻辑的核心。用户利用描述语言构造概念（关系）描述，从原子概念（关系）构造复合概念（关系），这些描述构成了 DL 的知识库。知识库（以下简称 KB）一分为二：一部分是术语的 (terminological)，记为 TBox，包含对概念的定义以及约束。另一部分是断言的 (assertional)，记为 ABox，包含对个体与概念之间关系的断言以及个体与个体之间关系的断言。

下面我们将基于 $\mathcal{ALC}$ 对概念描述以及 KB 给出形式化定义：

2.1 $\mathcal{ALC}$ 概念描述及其解释

首先，我们需要一个概念名（原子概念）的集合与一个关系名（原子关系）的集合，在此基础之上利用概念构造算子构造概念描述:

定义 2.1. （*$\mathcal{ALC}$ concept description*）设 $\mathbf{C}$ 是概念名的一个集合，$\mathbf{R}$ 是与 $\mathbf{C}$ 不相交的一个关系名的集合。那么，基于 $\mathbf{C}$ 与 $\mathbf{R}$ 的 $\mathcal{ALC}$ 概念描述的集合可由以下归纳定义而得:

1) 若 A 是一个概念名（$A \in \mathbf{C}$），则 A 是一个 $\mathcal{ALC}$ 概念描述；

2) $\top$ 与 $\bot$ 是 $\mathcal{ALC}$ 概念描述；

3) 若 C 和 D 是 $\mathcal{ALC}$ 概念描述，r 是一个关系名，则下列字符串是 $\mathcal{ALC}$ 概念描述：

$C \sqcap D$; $C \sqcup D$; $\neg C$; $\exists r.C$; $\forall r.C$.

定义 2.1 给出了 $\mathcal{ALC}$ 概念描述的语法，表明了符合语法规则的表达式（字符串）。如：$\exists r.(\forall s.C \sqcup D)$ 就是一个 $\mathcal{ALC}$ 概念描述，而，$r \sqcup E\forall$ 则不是，因为它并不符合概念描述的构成规则。

定义中所使用的布尔算子 $\sqcap$，$\sqcup$，$\neg$ 分别代表合取，析取，否定。$\exists$ 与 $\forall$ 分别是存在量词与全称量词。$\top$ 是 $A \sqcup \neg A$ 的缩写，其中 A 是任一概念名，因此 $\top$ 代表整个论域。$\bot$ 是 $\neg\top$ 的缩写，代表空集。

另外，概念名与关系名集合的元素构成是与其应用域及用户指定相关的。例如：若以学校为讨论对象，概念名的集合可以是：$\mathbf{C} = \{$老师, 学生, 学科, 课程名…$\}$，关系名的集合则可以是：$\mathbf{R} = \{$讲授, 学习…$\}$。若以公司为讨论对象，概念名的集合可以是：$\mathbf{C} = \{$员工, 老板, 项目…$\}$，关系名的集合则可以是：$\mathbf{R} = \{$雇佣, 负责…$\}$。

定义 2.2. （$\mathcal{ALC}$ *Interpretation*）一个解释 $I = \left(\Delta^I, \cdot^I\right)$ 由一个非空集 Δ^I 与一个映射 $\cdot^I$ 组成。其中 Δ^I 为解释域，映射 $\cdot^I$ 将每个概念名 $A \in \mathbf{C}$ 映射到一个集合 $A^I \subseteq \Delta^I$，将每个关系名 $r \in \mathbf{R}$ 映射到一个二元关系 $r^I \subseteq \Delta^I \times \Delta^I$。并且满足以下语义：

$$
\begin{aligned}
\top^I &= \Delta^I \\
\bot^I &= \emptyset \\
(C \sqcap D)^I &= C^I \cap D^I \\
(C \sqcup D)^I &= C^I \cup D^I \\
(\neg C)^I &= \Delta^I \setminus C^I \\
(\exists r.C)^I &= \left\{d \in \Delta^I \mid \text{存在} e \in \Delta^I, \text{使} (d,e) \in r^I \text{且} e \in C^I\right\} \\
(\forall r.C)^I &= \left\{d \in \Delta^I \mid \text{对所有} e \in \Delta^I, \text{若} (d,e) \in r^I \text{则} e \in C^I\right\}
\end{aligned}
$$

其中，C^I 被称为 C 在 I 中的外延；若 $(a,b) \in r^I$，则 $b \in \Delta^I$ 被称为 a 的 r-后继。注意，对解释域的限制仅仅是其必然非空，其基数可以是无穷的。

为了帮助对以上两个定义的理解，我们以下面给出的解释 I_1 为例进行说明：

$$
\begin{aligned}
\Delta^{I_1} &= \{m, n, w, X, Y\} \\
\text{人类}^{I_1} &= \{m, n, w\} \\
\text{男性}^{I_1} &= \{m, n\} \\
\text{女性}^{I_1} &= \{w\} \\
\text{老板}^{I_1} &= \{m\} \\
\text{员工}^{I_1} &= \{m, w\} \\
\text{雇佣}^{I_1} &= \{(m,n), (m,w)\} \\
\text{项目}^{I_1} &= \{X, Y\} \\
\text{负责}^{I_1} &= \{(m,X), (w,X), (w,Y)\}
\end{aligned}
$$

注意，关于语义，我们给出的是一种集合论的解释：概念被解释为个体的集合，关系则被解释为个体有序对的集合。因此，在其中我们很容易能够分辨出“雇佣”以及“负责”是关系名，其余则是概念名（当然，在具体实践中是由用户进行定义的）。且，形如“女性 $\sqsubseteq \exists$负责.项目”的概念描述则是由“女性”，“项目”两个概念名（原子概念）以及“负责”关系名（原子关系）所构成的复合概念，其所描述的概念是“是女性且存在负责的项目”，通过映射 $\cdot^{I_1}$，我

们可以在定义域 Δ^{I_1} 上寻找满足此概念描述的一个子集，并由此得到 $\{w\}$。

2.2 $\mathcal{ALC}$ 的知识库及其模型

基于 2.1 对概念描述的定义，以及前文对于知识库两个构成部分特征的介绍。下面我们给出 $\mathcal{ALC}$ 知识库的形式化定义：

定义 2.3. （*$\mathcal{ALC}$ TBox*）设 C 与 D 是 $\mathcal{ALC}$（复合）概念，则形如 $C \sqsubseteq D$ 的表达式称为一个 $\mathcal{ALC}$ 一般概念包含式（*general concept inclusion*），简称 *GCI*。我们用 $C \equiv D$ 作为 $C \sqsubseteq D$，$D \sqsubseteq C$ 两个表达式的缩写。并称形如 $A \equiv C$ 的表达式为 A 的概念定义，其中 A 为概念名，C 为（复合）概念。一个 *GCI* 的有穷集合称为一个 $\mathcal{ALC}$ *TBox*。

若解释 I 中有 $C^I \subseteq D^I$，则称 I 满足 GCI $C \sqsubseteq D$。若一个解释满足一个 TBox T 中的每个 GCI，那么此解释是 T 的一个模型（model）。

可以看出，TBox 中所包含的是对世界更为一般且抽象的知识。具体例如：“人类 $\sqsubseteq$ 哺乳动物” 就是一个 GCI，“母亲 $\equiv$ 女性 $\sqcap \exists$生育.人类” 则是一个概念定义，即 “母亲 $\sqsubseteq$ 女性$\sqcap\exists$生育.人类” 以及 “母亲 $\sqsupseteq$ 女性$\sqcap\exists$生育.人类” 两个表达式的缩写。假设有一个 TBox T_1=$\{$员工 $\sqsubseteq \exists$负责.项目$\}$，根据 2.1 中给出的解释 I_1，对其中唯一（全部）的 GCI 公式有 员工$^{I_1} \subseteq$ ($\exists$负责.项目)^{I_1}。因此，解释 I_1 满足 GCI 员工 $\sqsubseteq \exists$负责.项目，I_1 是 T_1 的一个模型。

定义 2.4. （*$\mathcal{ALC}$ ABox*）设 $\boldsymbol{I}$ 是与 $\boldsymbol{R}$ 和 $\boldsymbol{C}$ 不相交的一个个体名的集合，个体名 $a, b \in \boldsymbol{I}$，（复合）概念 C 和关系名 $r \in \boldsymbol{R}$，则：

$a{:}C$ 称为一个 $\mathcal{ALC}$ 概念断言，

$(a, b) : r$ 称为一个 $\mathcal{ALC}$ 关系断言。

一个 $\mathcal{ALC}$ 概念和关系断言的有穷集合称为一个 $\mathcal{ALC}$ *ABox*。

若解释 I 中有 $a^I \in C^I$，则称 I 满足概念断言 $a : C$。若有 $(a^I, b^I) \in r^I$，则称 I 满足关系断言 $(a, b) : r$。若一个解释满足一个 ABox A 中的每个概念断言和关系断言，那么此解释是 A 的一个模型。

可以看出，ABox 中所包含的是对具体元素的性质以及关系的知识。具体例如：“m : 男性 $\sqcap$ 老板” 就是一个概念断言，“(m, n) : 雇佣” 则是一个关系断言，显然，在 2.1 中的解释 I_1 下，有 $m^{I_1} \in$ (男性 $\sqcap$ 老板)^{I_1} 以及 $(m^{I_1}, n^{I_1}) \in$ 雇佣^{I_1}。假设有一个 ABox $A_1 = \{m$: 男性 $\sqcap$ 老板, (m, n) : 雇佣$\}$，那么解释 I_1 满足其中所有的概念断言以及关系断言，因此解释 I_1 是 A_1 的一个模型。

定义 2.5. （*$\mathcal{ALC}$ Knowledge Base*）一个 $\mathcal{ALC}$ 知识库 $K = (T, A)$ 由一个 $\mathcal{ALC}$ *TBox* T 和一个 $\mathcal{ALC}$ *ABox* A 组成。

若一个解释既是 T 的模型，又是 A 的模型，则称其为 K 的模型。

例：假设由上文中的 TBox T_1 与 ABox A_1 构成 KB K_1，由于 I_1 既是 T_1 的模型，又是 A_1 的模型，因此，I_1 是 K_1 的模型。

2.3 相关术语

2.3.1 概念的否定范式（Negation Normal Form）

为了避免冗余、简化表达，我们可以假定 TBox 与 ABox 中所有概念都以否定范式的形式出现，即否定符号（$\neg$）只出现在概念名之前。这项工作可以利用德摩根律实现，使任

一 $\mathcal{ALC}$ 概念被等价地转换为其否定范式的形式：

$$\neg\exists r.C = \forall r.\neg C \qquad \neg(C \sqcap D) = \neg C \sqcup \neg D \qquad \neg\top = \bot \qquad \neg\neg C = C$$
$$\neg\forall r.C = \exists r.\neg C \qquad \neg(C \sqcup D) = \neg C \sqcap \neg D \qquad \neg\bot = \top$$

根据定义 2.2, 我们可以得到：在任一解释 I 中, 对任意 $\mathcal{ALC}$ 概念 C 有 $C^I = nnf(C)^I$, 即 C 与其否定范式等价。

另外，我们称一个 ABox A 是范式化的，若：A 是非空的，且每个在 A 中出现的个体名都至少在 A 中以 $a : C$ 的形式出现一次。注意，这一要求并不对 ABox 造成实质性的限制，因为我们可以对任意的个体名 a，在 A 中增加断言 $a : \top$，并且此方法在引入新个体名时同样适用。

2.3.2 概念的规模（size）与子概念（subconcept）

对给定的 $\mathcal{ALC}$ 概念 C，通过对 C 的结构施归纳。我们对其规模定义 $\mathrm{size}(C)$，对其子概念集合定义 $\mathrm{sub}(C)$:

1) 当 $C = A \in \mathbf{C} \cup \{\top, \bot\}$，那么 $\mathrm{size}(C) = 1$，$\mathrm{sub}(C) = \{A\}$;

2) 当 $C = C_1 \sqcap C_2$ 或 $C = C_1 \sqcup C_2$，那么 $\mathrm{size}(C) = 1 + \mathrm{size}(C_1) + \mathrm{size}(C_2)$，$\mathrm{sub}(C) = \{C\} \cup \mathrm{sub}(C_1) \cup \mathrm{sub}(C_2)$;

3) 当 $C = \neg D$ 或 $C = \exists r.D$ 或 $C = \forall r.D$，那么 $\mathrm{size}(C) = 1 + \mathrm{size}(D)$，$\mathrm{sub}(C) = \{C\} \cup \mathrm{sub}(D)$.

size() 仅仅是将概念名（包括 $\top$ 和 $\bot$），关系名以及布尔运算符的出现次数进行计数。例如：

$$\mathrm{size}(A \sqcap \exists r.(A \sqcup B)) = 1 + 1 + (1 + (1 + 1 + 1)) = 6$$

对于同一概念，其子概念集则是：

$$\mathrm{sub}(A \sqcap \exists r.(A \sqcup B)) = \{A \sqcap \exists r.(A \sqcup B), A, \exists r.(A \sqcup B), A \sqcup B, B\}$$

我们可以将这两个概念扩张到 TBox 上：

$$\mathrm{size}(T) = \sum_{C \sqsubseteq D \in T} \mathrm{size}(C) + \mathrm{size}(D)$$

$$\mathrm{sub}(T) = \bigcup_{C \sqsubseteq D \in T} \mathrm{sub}(C) + \mathrm{sub}(D)$$

由此易知，一个概念或一个 TBox 的子概念的数量是被此概念或此 TBox 的规模所限制的。据此易得引理 2.6:

引理 2.6. *若 C 是一个 $\mathcal{ALC}$ 概念，T 是一个 $\mathcal{ALC}$ TBox，那么 $|\mathrm{sub}(C)| \leq \mathrm{size}(C)$ 且 $|\mathrm{sub}(T)| \leq \mathrm{size}(T)$.*

另外，在 ABox 与 KB 上子概念的定义亦十分显然：

$$\mathrm{sub}(A) = \bigcup_{a:C \in A} \mathrm{sub}(C)$$

$$\mathrm{sub}(K) = \mathrm{sub}(T) \cup \mathrm{sub}(A)$$

根据以上定义以及引理 2.6，我们可以直接推出引理 2.7：

引理 2.7. *对所有 $\mathcal{ALC}$ ABox A，都有 $|\mathrm{sub}(A)| \leq \sum_{a:C \in A} \mathrm{size}(C)$.*

以上定义以及引理将应用于第三部分的算法介绍及证明。

3 $\mathcal{ALC}$ 的 Tableau 算法

在第二部分我们引入了 $\mathcal{ALC}$ 语言，并基于 $\mathcal{ALC}$ 语言对知识的形式化、符号化表示以及对知识库的刻画方法的进行了介绍。接下来，我们将为基于 $\mathcal{ALC}$ 的系统提供一种推理工具。

在这一部分，我们将介绍一个算法，它以 $\mathcal{ALC}$ 知识库 $K = (T, A)$ 作为输入，运行输出结果“一致的”或“不一致的”。然后，我们将证明此算法是一个 $\mathcal{ALC}$ 知识库一致性的判定算法。即，此算法是必然终止的，并且它返回“一致的”当且仅当作为输入的知识库 K 是一致的。

下面我们将介绍关于此算法实现的基本原理和方法：

首先，虽然在表面上此算法是仅关于知识库一致性问题的算法。但实际上，是在所讨论的推理问题复杂度不会在系统中产生指数爆炸的前提下，对其他重要推理问题 (如：概念的可满足性、包含关系、语义后承等) 的共同讨论。这一点主要得益于其他主要的推理问题都能够被规约为对知识库一致性问题的探讨。

其次，一个算法是判定算法，指其能够终止，并可靠且完全。算法的终止性确保我们能够通过算法在有限步内得到一个输出结果，可靠性与完全性则保证我们通过算法得出的结果是正确的。

接下来，在具体的实现方法上，对知识库一致性的讨论主要是基于语义定义，即：若一个知识库 K 是一致的，则必然存在解释是 K 的模型。假设解释 I_1 是 K_1 的模型，即 I_1 的存在证明了 K_1 是一致的，那么我们就称 I_1 是 K_1 一致性的见证（witness）。

基于 Tableau 的推理方法就是通过论证一知识库存在恰当的见证，以证明此知识库的一致性。在此方法中，对见证存在的证明是构造性的：这项工作将以知识库 $K = (T, A)$ 的 ABox A 为出发点，利用扩张规则（expansion rule）对 A 与 T 中的概念及等式约束进行语义解释及扩张，其扩张结果要么是成功构造了 K 的一个见证，要么是在构造过程中发现了矛盾，即此 K 不存在见证，也就是 K 是不一致的。

下面，我们将分两步对 KB 一致性算法进行介绍：首先我们将考虑 ABox 一致性的判定算法，即 KB 的 TBox 为空的情形。然后再将算法推广到一般的 KB 的情形。

3.1 $\mathcal{ALC}$ ABox 一致性的判定算法

考虑 $K = (\emptyset, A)$ 的情形，算法将尝试构造一个森林状的 ABox，通过使用扩张规则将 A 扩张至完全，然后通过检测其中是否存在显然的矛盾来判断其是否一致。由于 TBox 为空，扩张规则只需对在 A 的概念断言中出现的概念进行语义解释。这些规则实质上是在句法上分解概念，一旦所有概念都被完全分解了，算法自然也就终止了。

3.1.1 算法的基本内容

在正式介绍算法之前，我们将引入一些将会在证明中利用到的辅助定义：

首先，我们将用 $\mathrm{con}_A(a)$ 来表达在 A 中形式如 $a : C$ 的概念断言中概念 C 的集合，即：$\mathrm{con}_A(a) = \{C \mid a : C \in A\}$。

其次，我们需要对算法中出现的“矛盾”，以及其所使用的扩张规则进行形式化的定义：

定义 3.1. *我们称一个 ABox A 包含冲突（contains a clash），如果存在个体名 a 以及概念 C 使得 $a : C, a : \neg C \subseteq A$，或者 $a : \bot \subseteq A$。如果 A 不包含冲突，我们就称其为无冲突的*

(*clash-free)*。我们称 A 是完全的（*complete*），如果它包含冲突，或者没有一个扩张规则是可使用的。

扩张规则：

$\sqcap$-规则：若 $a : C \sqcap D \in A$，且 $\{a : C, a : D\} \nsubseteq A$，则 $A \to A \cup \{a : C, a : D\}$

$\sqcup$-规则：若 $a : C \sqcup D \in A$，且 $\{a : C, a : D\} \cap A = \emptyset$，则 $A \to A \cup \{a : X\}$ 其中 $X \in \{C, D\}$

$\exists$-规则：若 $a : \exists r.C \in A$，且不存在 b 使得 $\{(a, b) : r, b : C\} \subseteq A$，则 $A \to A \cup \{(a, d) : r, d : C\}$ 其中个体 d 是新引入 A 的

$\forall$-规则：若 $\{a : \forall r.C, (a, b) : r\} \subseteq A$，且 $b : C \notin A$，则 $A \to A \cup \{b : C\}$

图 1: $\mathcal{ALC}$ ABox 一致性的语法扩张规则

需要注意的是，除 $\sqcup$-规则之外的所有扩张规则的应用都能够对 ABox 进行唯一、确定地扩张。而 $\sqcup$-规则将导致不同的可能性扩张。如：$\{a : \neg D, a : C \sqcup D\}$ 的可能扩张有两种：$\{a : \neg D, a : C \sqcup D, a : C\}$ 与 $\{a : \neg D, a : C \sqcup D, a : D\}$，即使显然后者是包含冲突的，但在算法的运行过程中这两个扩张路径都是均等可能的。另外，在我们的确定性算法中，我们将递归地检测从 $\sqcup$-规则的应用中得出的每个扩张的一致性，若其中任意一个扩张是一致的，我们就称此 ABox 是一致的。因此，由于上例中的 $\{a : \neg D, a : C \sqcup D\}$ 的第一种扩张是一致的，所以它自身是一致的。

最后，定义一个将在算法中使用的函数 exp()，其输入是由一个无冲突的、范式化的 $\mathcal{ALC}$ ABox A，一个规则 R 以及 A 中的一个断言 α 构成的三元组，其中规则 R 是能够应用于 α 的。函数输出的是在 A 中对 α 应用了 R 后得到的所有 ABox 的集合 $\exp(A, R, \alpha)$。例如：

- $\exp(\{a : \forall r.C, b : D, (a, b) : r\}, \forall - \text{规则}, (a : \forall r.C, (a, b) : r))$ 将输出单例集 $\{a : \forall r.C, b : D, (a, b) : r, b : C\}$.
- $\exp(\{a : \neg C, a : C \sqcup D\}, \sqcup - \text{规则}, a : C \sqcup D)$ 将输出包含两个 ABox 的集合：$\{a : \neg C, a : C \sqcup D, a : C\}$ 和 $\{a : \neg C, a : C \sqcup D, a : D\}$.

准备工作已经完毕，下面将正式给出算法：

定义 3.2. *$\mathcal{ALC}$ ABox 一致性的算法* consistent，*输入范式化的 $\mathcal{ALC}$ ABox A 并利用算法* expand *将图 1 中的扩张规则应用于 A，两个算法如下：*

Algorithm 1 consistent()

Input: 一个范式化的 $\mathcal{ALC}$ ABox A
  if expand(A) $\neq \emptyset$ **then**
    return “一致的”
  else
    return “不一致的”
  end if

Algorithm 2 expand()

Input: 一个范式化的 $\mathcal{ALC}$ ABox A

if A 不是完全的 **then**
  选择一个能够对 A 应用的规则 R 以及 A 中能够被 R 应用于的一个或一对断言 α
  if 存在 $A' \in \exp(A, R, \alpha)$使$\text{expand}(A') \neq \emptyset$ **then**
    return $\text{expand}(A')$
  else
    return $\emptyset$
  end if
else
  if A 包含冲突 **then**
    return $\emptyset$
  else
    return A
  end if
end if

可以看到，对 consistent 算法输入一个范式化的 $\mathcal{ALC}$ ABox A，consistent 算法将对 expand 算法进行调用:

在 expand 算法中，首先将对 A 的完全性进行检测：

若 A 是完全的（定义 3.1），那么就将对 A 中是否存在冲突进行检测：若存在冲突，例如在 A 中有 $\{a : C, a : \neg C\}$，那么显然不存在满足此 ABox A 的解释，因此 A 必然是不一致的，根据 expand 算法输出结果 $\emptyset$，并返回到 consistent 算法最终输出 “不一致的”；若不存在冲突，又因为此 ABox A 是完全的（没有再次扩张而引入冲突的可能），因此对这一 ABox A 必然是存在模型的，那么 expand 算法将输出 A，并返回 consistent 算法输出结果 “一致的”。

若 A 不是完全的，那么将选取一个能够对 A 中某个断言 α 应用的规则 R。并将这三个元素组成的三元组 (A, R, α) 输入到函数 exp() 中，得出其结果，即扩张后的 ABox 集合 $\exp(A, R, \alpha)$。expand 算法将递归的调用自身，直到出现以下两种情况：或者 A 被扩张成一个完全的 ABox A'，从而回到上文中对 ABox 是完全的情况的讨论；或者 A 在利用 exp() 函数进行扩张时产生了冲突（因为根据 exp() 函数的定义，其输入仅限于无冲突的 ABox），导致 A 不能继续利用 exp 函数应用扩张规则进行扩张，expand 算法输出 $\emptyset$，并返回到 consistent 算法最终输出结果 “不一致的”。

3.1.2 算法的正确性证明

在前文，我们已经说明了一个判定算法应当具有的三个性质，即：终止性、可靠性与完全性。终止性确保了算法对任一合法输入，在有穷步骤之内能够输出一项结果，而不是无穷尽地运行下去。如：对定义 3.2 中算法输入一个范式化的 $\mathcal{ALC}$ ABox A，在有穷步内必然输出一个结果 “一致的” 或 “不一致的”。可靠性与完全性则确保了算法所输出的结果是正确的：（根据可靠性）既不会出现一个输入的 ABox 是不一致的，但是算法输出为 “一致的”；（根据完全性）也不会出现一个一个输入的 ABox 是一致的，但是算法输出为 “不一致的”。

下面我们将给出对定义 3.2 中算法正确性的严格证明：

引理 3.3. （终止性）对任一 $\mathcal{ALC}$ *ABox* A，consistent(A) 能够终止。

证明. 设 $m = |\text{sub}(A)|$。终止性是根据扩张规则下列的性质所得出的推论：

(1) 扩张规则不会从 A 中移除断言，并且每次应用规则增加的是形如 $a : C$ 的断言，其中 a 为个体名且概念 $C \in \text{sub}(A)$。从引理 2.6 与 2.7 可知，sub(A) 的规模被 A 的规模所限制。且 $|\text{con}_A(a)| \leq m$，因此对任一个体名 a，至多能够使用 m 次规则以增加形如 $a : C$ 的概念断言。

(2) 每次使用 ∃-规则时会在 A 中增加一个个体名。∃-规则对形如 $a : \exists r.D$ 的断言使用，对任一个体名每个这样的断言至多能增加一个新的个体名。（与（1）同理）由于在 A 中至多有不超过 m 个不同的存在限制，那么每个给定的个体名至多能够增加 m 个新个体名。因此，森林状 ABox 的每棵树的出度上限为 m。

(3) ∃-以及 ∀-规则分别应用于形如 $a : \exists r.C$ 以及 $a : \forall r.C$ 的断言，并且它们增加的仅可能为形如 $b : C$ 的断言，其中 b 是 a 的后继。在两种情况下，C 都是概念 $\exists r.C$ 和 $\forall r.C$ 的严格子概念。那么，若任意规则仍然能够应用于 $b : C$，在 A 中增加概念 $b : D$，那么 D 是 C 的子描述。因此 $\text{sub}(\text{con}_A(b)) \subseteq \text{sub}(\text{con}_A(a))$ 并且 $\text{con}_A(b)$ 中规模最大的概念严格小于 $\text{con}_A(a)$ 中规模最大的概念。因此，森林状 ABox 的每棵树的深度上限为 m。

这些性质保证了对 expand 的递归使用是有次数上限的，以及通过扩张规则构造的 ABox 规模是有界限的。因此，consistent 算法是必然能够终止的。 □

引理 3.4. （可靠性）若 consistent(A) 输出“一致的”，那么 A 是一致的。

证明. 设 A' 是从 expand(A) 输出的集合。因为算法输出了“一致的”，那么 A' 是完全且不包含冲突的 ABox。

证明的核心思想是为 A' 构建一个模型 I（即，满足 A' 中的所有断言）。因为扩张规则从不会移除断言，有 $A \subseteq A'$，那么 I 也是 A 的模型，也就是 A 一致性的见证。因此，利用构造典范模型的方法，我们对 A' 构建一个合适的解释 I：

$$\Delta^I = \{a \mid a : C \in A'\}$$
对任一在 A' 中出现的个体名 a，$a^I = a$
对任一在 sub(A') 中的概念名 A，$A^I = \{a \mid A \in \text{con}_{A'}(a)\}$
对任一在 A' 中出现的关系 r，$r^I = \{(a, b) \mid (a, b) : r \in A'\}$

注意，在 A' 中出现的每个个体名 a 都至少在一个概念断言中出现：对于根个体（即已经出现在输入 ABox 中的个体），这一结论可以从 A 的结构推出。对于叶个体（即通过 ∃-规则后来引入的个体），这一结论可以从 ∃-规则的定义推出，因此 A' 是范式化的 ABox。另外，我们容易证明 I 作为解释是良定义的：根据假设，A 包含至少一个形式为 $a : C$ 的断言，因此 Δ^I 非空。且通过构造 $\cdot^I$ 将 A' 中的每个个体名映射到 Δ^I 中的一个元素，将每个概念名 $A \in \text{sub}(A')$ 映射到 Δ^I 的一个子集上，将每个 A' 中的每个关系 r 映射到 $\Delta^I \times \Delta^I$ 的一个子集上。

根据定义 2.4，I 要作为 A' 的模型，则 I 必须满足 A' 中的所有概念和关系断言。一方面，根据 I 的构造，易知 A' 中的所有关系断言都被满足。另一方面，要证明 I 满足 A' 中的所有概念断言，我们将通过对概念的结构施归纳，证明以下性质：

$$\text{若 } a : C \in A', \text{ 则 } a^I \in C^I$$

奠基步：若 C 是概念名：$a:C\in A'$，则根据 I 的构造可得 $a^I\in C^I$。
归纳步:

(1) 若 $C=\neg D$：因为 A' 是无冲突的，$a:\neg D\in A'$ 意味着 $a:D\notin A'$。且 A 中的所有概念都是否定范式的形式，D 是一个概念名。根据 I 的定义，有 $a^I\notin D^I$，即 $a^I\in\Delta^I\backslash D^I=C^I$。

(2) 若 $C=D\sqcup E$：则 $a:D\sqcup E\in A'$，那么 A' 的完全性表明 $\{a:D,a:E\}\cap A'\neq\emptyset$（否则 $\sqcup$-规则是能够应用的）。根据 $\sqcup$ 的语义定义，有 $a^I\in D^I\cup E^I=(D\sqcup E)^I$。

(3) 若 $C=D\sqcap E$：则 $a:D\sqcap E\in A'$，那么 A' 的完全性表明 $\{a:D,a:E\}\subseteq A$（否则 $\sqcap$-规则是能够应用的）。根据 $\sqcap$ 的语义定义，有 $a^I\in D^I\cap E^I=(D\sqcap E)^I$。

(4) 若 $C=\forall r.D$：则 $a:\forall r.D\in A'$，因为 A' 是完全的，那么必然存在 b 使 $(a^I,b^I)\in r^I$。因此对 $a^I\in(\forall r.D)^I$，需证 $b^I\in D^I$。根据 I 的定义，$(a,b):r\in A'$。因为 A' 是完全的，且 $a:\forall r.D\in A'$，则 $b:D\in A'$（否则 $\forall$-规则是能够应用的）。根据归纳假设，对所有满足 $(a^I,b^I)\in r^I$都有$b^I\in D^I$。因此根据 $\forall$ 的语义定义，有 $a^I\in(\forall r.D)^I$。

(5) 若 $C=\exists r.D$：则 $a:\exists r.D\in A'$，因为 A' 是完全的，那么必然存在 b 使 $\{(a,b):r,b:C\}\subseteq A'$（否则 $\exists$-规则是能够应用的）。那么对 $a^I\in(\exists r.D)^I$，需证 $b^I\in D^I$。根据 I 的定义，有 $(a^I,b^I)\in r^I$。根据归纳假设，可得 $b^I\in D^I$。因此根据 $\exists$ 的语义定义，有 $a^I\in(\exists r.D)^I$。

综上，I 满足 A'（以及 A）中的所有概念断言。并且根据定义，I 也满足 A'（以及 A）中所有关系断言。因此 A 存在模型，即 A 是一致的。 □

引理 3.5. （完全性）若 A 是一致的，那么 consistent(A) 输出“一致的”。

证明. 设 A 是一致的，A 的一个模型为 $I=(\Delta^I,\cdot^I)$。因为 A 是一致的，那么 A 不包含冲突。

若 A 是完全的，又因为它不包含冲突，那么 expand 会直接输出 A 且 consistent 会输出“一致的”。

若 A 并非完全的，那么 expand 将递归地调用自身直到 A 扩张到完全，每一次调用都会选取一个规则应用于 A。因此，通过证明所有的扩张规则都是保持一致性的，可使引理得证。下面是对规则的分别讨论：

(1) $\sqcup$-规则：若 $a:C\sqcup D\in A$，那么 $a^I\in(C\sqcup D)^I$。且定义 2.2 表明或 $a^I\in C^I$，或 $a^I\in D^I$。所以，至少存在一个 ABox $A'\in\exp(A,\sqcup-\text{规则},a:C\sqcup D)$ 是一致的。因此，至少有一个 expand 的调用是应用于一个一致的 ABox。

(2) $\sqcap$-规则：若 $a:C\sqcap D\in A$，那么 $a^I\in(C\sqcap D)^I$。且定义 2.2 表明 $a^I\in C^I$且$a^I\in D^I$。所以 I 仍是 $A\cup\{a:C,a:D\}$ 的模型。因此在此规则的使用后 A 仍是一致的。

(3) $\exists$-规则：若 $a:\exists r.C\in A$，那么 $a^I\in(\exists r.C)^I$。且定义 2.2 表明存在 $x\in\Delta^I$ 使 $(a^I,x)\in r^I$且$x\in C^I$。因此存在 A 的模型 I' 中有新的个体名 d 使 $d^{I'}=x$，其余的与 I 一致。模型 I' 仍是 $A\cup\{(a,d):r,d:C\}$ 的模型。因此在此规则的使用后 A 仍是一致的。

(4) $\forall$-规则：若 $\{a:\forall r.C,(a,b):r\}\subseteq A$，那么 $a^I\in(\forall r.C)^I$，$(a^I,b^I)\in r^I$。且定义 2.2 表明 $b^I\in C^I$。所以 I 仍是 $A\cup\{b:C\}$ 的模型。因此在此规则的使用后 A 仍是一致的。

□

定理 3.6. *定义 3.2 中的 tableau 算法是对 $\mathcal{ALC}$ ABox 一致性的判定算法。*

证明. 任意一个 $\mathcal{ALC}$ ABox 都能被等价地转换为范式化的 ABox。定义 3.2 中的算法是范式化 $\mathcal{ALC}$ ABox 一致性的判定算法可由引理 3.3，3.4 以及 3.5 直接推出。 □

3.2 $\mathcal{ALC}$ KB 的一致性的判定算法

接下来我们将把 tableau 算法扩张到完整的 $\mathcal{ALC}$ Knowledge Base 上。为了处理新增加的 TBox，首先我们将“范式化”这一概念进行扩张：

根据语义定义，我们可以得到下面两个等式：

- I 满足 $C \sqsubseteq D$ 当且仅当 I 满足 $\top \sqsubseteq D \sqcup \neg C$
- I 满足 $C \equiv D$ 当且仅当 I 满足 $\top \sqsubseteq (D \sqcup \neg C) \sqcap (C \sqcup \neg D)$

利用上面两个等式，我们就可以在不丧失讨论情况普遍性的前提下将所有的 TBox 中的表达式转化为 $\top \sqsubseteq E$。据此，将范式化的概念扩张到 TBox 以及 KB 上，即：如果一个 TBox 中的所有表达式都是 $\top \sqsubseteq E$ 形式的（其中 E 是否定范式），那么我们就称此 TBox 是范式化的。如果一个知识库 $K = (T, A)$ 中，T 与 A 都是范式化的，那么我们就称 K 为范式化的。

3.2.1 算法的基本内容

在介绍算法之前，我们将对图 1 中的扩张规则进行两项扩充：一是增加 $\sqsubseteq$-规则，以处理 TBox 中的 GCIs。其二是为 $\exists$-规则增加一项说明，以保证算法的终止性。

注意，第二项扩充所针对的终止性问题产生于 $\exists$-规则与 $\sqsubseteq$-规则的简单合并。例如：若 KB 为 $(\{A \sqsubseteq \exists r.A\}, \{a : A\})$，此情况将导致引入 a 的一个后继 b，但 $\mathrm{sub}(\mathrm{con}_A(b))$ 不再是 $\mathrm{sub}(\mathrm{con}_A(a))$ 的严格子集。对比引理 3.3 中对 ABox 算法终止性的证明，此时的森林状 ABox 中树的深度不再有限制。

扩充后的算法如图 2 所示：

扩张规则：

$\sqcap$-规则：若 $a : C \sqcap D \in A$，且 $\{a : C, a : D\} \nsubseteq A$，则 $A \to A \cup \{a : C, a : D\}$

$\sqcup$-规则：若 $a : C \sqcup D \in A$，且 $\{a : C, a : D\} \cap A = \emptyset$，则 $A \to A \cup \{a : X\}$ 其中 $X \in \{C, D\}$

$\exists$-规则：若 $a : \exists r.C \in A$，且不存在 b 使得 $\{(a, b) : r, b : C\} \subseteq A$，并且 a 没有被封锁，则 $A \to A \cup \{(a, d) : r, d : C\}$ 其中个体 d 是新引入 A 的

$\forall$-规则：若 $\{a : \forall r.C, (a, b) : r\} \subseteq A$，且 $b : C \notin A$，则 $A \to A \cup \{b : C\}$

$\sqsubseteq$-规则：若 $a : C \in A, \top \sqsubseteq D \in T$，且 $a : D \notin A$，则 $A \to A \cup \{a : D\}$

图 2: $\mathcal{ALC}$ KB 一致性的语法扩张规则

具体地说，对 $K = (T, A)$ 以及 ABox 中的任一个体名 a，$\mathrm{con}_A(a) \subseteq \mathrm{sub}(K)$。因此在 tableau 算法的运行中，一棵树的每一分支在其出现两个不同的个体名 a 与 b 使得 $\mathrm{con}_A(a) = \mathrm{con}_A(b)$ 的情况之前，至多能够拥有 $2^{|\mathrm{sub}(K)|}$ 个实质不同的个体名。并且在上述情况出现之后，扩张规则对 b 的应用最终亦只能是引入一个个体名 b' 使得 $\mathrm{con}_A(b) = \mathrm{con}_A(b')$，并且这种构造将无限制、无穷地进行下去。为了解决这个问题，即，当出现在一个分支的构

造过程中出现了两个个体名互为对方的“克隆”的情况时，我们将立即停止此分支的构造，避免显式地引入更多的“克隆”。这种技术被称为封锁 (blocking)，根据这一理念，我们可以保证算法的终止性。下面是对封锁的定义及介绍：

定义 3.7. *($\mathcal{ALC}$ blocking) 在 $\mathcal{ALC}$ ABox A 中，若个体名 a 是个体名 b 的祖先，且 $\mathrm{con}_A(a) \supseteq \mathrm{con}_A(b)$。则称 b 被 a 封锁。*

若存在个体名 a 封锁 b，或者存在 b 的祖先在 A 中被封锁，我们就称个体名 b 在 A 中被封锁。

注意，“祖先”是指“先驱”的传递闭包概念，即若 a 是 b 的先驱，则 a 是 b 的祖先。若 a 是 b 的某一先驱 b' 的先驱，则 a 是 b 的祖先。同理，“后代”是“后继”的传递闭包概念。根据以上定义我们可以得到以下推论：当一个个体名被封锁了，那么它的所有后代都是被封锁的；由于根个体没有祖先，所以它不可能被封锁。

下面我们将给出 $\mathcal{ALC}$ KB 一致性的判定算法，实质上此算法就是在定义 3.2 中给出的算法基础之上，改变了扩张规则 (从图 1 到图 2)，增添了处理范式化 $\mathcal{ALC}$ TBox T 的工作。

定义 3.8. *$\mathcal{ALC}$ KB 一致性的算法 consistent，输入范式化的 $\mathcal{ALC}$ KB $K = (T, A)$ 并利用算法 expand 在考虑到 TBox 中表达式的前提下，将图 2 的扩张规则应用于 A，两个算法如下：*

Algorithm 3 consistent()

Input: 一个范式化的 $\mathcal{ALC}$ KB (T, A)
  if $\mathrm{expand}(T, A) \neq \emptyset$ **then**
    return “一致的”
  else
    return “不一致的”
  end if

Algorithm 4 expand()

Input: 一个范式化的 $\mathcal{ALC}$ KB (T, A)
  if A 不是完全的 **then**
    选择一个能够对 A 应用的规则 R 以及 A 中能够被 R 应用于的一个或一对断言 α
    if 存在 $A' \in \exp(A, R, \alpha)$ 使 $\mathrm{expand}(T, A') \neq \emptyset$ **then**
      return $\mathrm{expand}(T, A')$
    else
      return $\emptyset$
    end if
  else
    if A 包含冲突 **then**
      return $\emptyset$
    else
      return A
    end if
  end if

此算法与定义 3.2 中算法的构造与核心思想基本一致。由于 TBox 中的 GCI 是对概念的约束限制，通过利用 $\sqsubseteq$-规则使对概念的限制直接作用于个体名上。例如：若存在范式化

GCI $\top \sqsubseteq \neg A \sqcup D$，那么对任一在 ABox A 中出现的个体名 a，都有 $a : \neg A \sqcup D$。因此，在 expand 算法中我们只需要考虑 ABox 是否完全以及是否包含冲突。

3.2.2 算法的正确性证明

对定义 3.8 中算法正确性的证明，与 3.1.2 的基本思想一致。在具体证明上增加了需要纳入考虑的部分，如前文提及的终止性问题，以及 $\sqsubseteq$-规则的引入，下面是具体的证明过程：

引理 3.9. （终止性）对任一 $\mathcal{ALC}$ *knowledge base* K，consistent(K) 能够终止。

证明. 该证明与引理 3.3 中的证明理念一致，具体证明过程的唯一区别在第三个性质，即森林状 ABox 每棵树的深度上限一处。

设 $m = |\mathrm{sub}(K)|$。根据图 2 中扩张规则的定义，可得 $\mathrm{con}_A(a) \subseteq \mathrm{sub}(K)$，且至多存在 2^m 个这样不同的集合。

(1) 对任一个体名 a，至多能够使用 m 次规则（见引理 3.3）。

(2) 森林状 ABox 的每棵树的出度上限为 m（见引理 3.3）。

(3) 因为 $\mathrm{con}_A(a) \subseteq \mathrm{sub}(K)$ 且 $|\mathrm{sub}(K)| = m$，那么在 ABox 中树个体的任一支出现包含两个不同的个体名 a 与 b，使 b 是 a 的后代且 $\mathrm{con}_A(a) \supseteq \mathrm{con}_A(b)$ 的情况之前，此分支至多包含 2^m 个个体名。在此情形下，对 b 应用 $\exists$-规则将使其所有后代被封锁。因此，森林状 ABox 的每棵树的深度上限为 2^m。

其中，第二与第三个性质确保了生成的新个体数量有限，即 ABox 的规模有限；第一个性质则确保了算法的终止（至多使用 m 次规则后得到输出）。 □

引理 3.10. （可靠性）若 consistent(K) 输出“一致的”，那么 K 是一致的。

证明. 与在引理 3.4 中的证明理念相似，我们将利用由 expand(K) 输出的完全且无冲突的 ABox A' 为 K 构造一个合适的模型 $I = (\Delta^I, \cdot^I)$，具体证明过程增加的部分即对那些被封锁的个体名的处理，这项工作可以通过两个步骤实现：第一步是构造一个包含 A' 中除被封锁的个体名之外所有断言的 ABox A''，并对在 A' 中出现的每个关系断言 $(a, b) : r$ 增加一个新的“环回（loop-back）”关系断言 $(a, b') : r$，其中 a 不被封锁，b 被 b' 封锁。第二步即利用 A'' 为 K 构造模型。

首先，我们对 A'' 进行构造：

$$\begin{aligned} A'' = &\{a : C \mid a : C \in A' \text{且} a \text{不被封锁}\} \cup \\ &\{(a, b) : r \mid (a, b) : r \in A' \text{且} b \text{不被封锁}\} \cup \\ &\{(a, b') : r \mid (a, b) : r \in A' \text{且} a \text{不被封锁且} b \text{被} b' \text{封锁}\} \end{aligned}$$

易知，因为 $A \subseteq A'$，所以有 $A \subseteq A''$。且 A 中的所有断言 $a : C$ 与 $(a, b) : r$ 中的个体 a 与 b 都是根个体，因此 A 中的所有个体名都不会被封锁。与此同时，A'' 中出现的所有个体名也不会被封锁，这可以根据对上面对 A'' 的构造分情形讨论证明：对于概念断言，这可以直接根据定义推出。对于关系断言，区分两种情形：一是 $(a, b) : r \in A'$ 且 b 不被封锁，那么显然 a 也不可能被封锁，因为一个被封锁的个体名的所有后代都是被封锁的。第二种情况，即 b 被封锁，而其先驱 a 不被封锁。那么只可能是 b 被 A’ 中存在的 b 的其他

先驱 b' 封锁了。那么，或者 $b' = a$ 或者 b' 是 a 的先驱，而 a 不被封锁则说明 b' 不能被封锁，因此 b' 也不被封锁。

根据 A'' 的定义，易知：

$$\mathrm{con}_{A''}(a) = \mathrm{con}_{A'}(a)$$

将此等式记为性质 $(*)$。因为 A' 是无冲突的，所以 A'' 也是无冲突的：性质 $(*)$ 则表明若 A'' 包含冲突，那么 A' 也包含冲突。此外，若 A' 是完全的，那么 A'' 也是完全的，证明如下：

(1) 对于 $\sqcap$-规则：若 $a : C \sqcap D \in A''$，则由性质 $(*)$ 可得 $a : C \sqcap D \in A'$。又因为 A' 是完全的，则 $\{a : C, a : D\} \subseteq A'$，根据性质 $(*)$，得 $\{a : C, a : D\} \subseteq A''$。

(2) 对于 $\sqcup$-规则：若 $a : C \sqcup D \in A''$，则由性质 $(*)$ 可得 $a : C \sqcup D \in A'$。又因为 A' 是完全的，则 $\{a : X\} \subseteq A'$ 其中 $X \in \{C, D\}$，根据性质 $(*)$，得 $\{a : X\} \subseteq A''$ 其中 $X \in \{C, D\}$。

(3) 对于 $\sqsubseteq$-规则：若 $a : C \in A''$，则由性质 $(*)$ 可得 $a : C \in A'$。又因为 A' 是完全的，对所有概念 D，若 $\top \sqsubseteq D \in T$，则 $a : D \in A'$，根据性质 $(*)$，得 $a : D \in A''$。

(4) 对于 $\exists$-规则：若 $a : \exists r.C \in A''$，则 $a : \exists r.C \in A'$ 且 a 不在 A' 中被封锁。又因为 A' 是完全的，所以存在 b 使得 $\{(a, b) : r, b : C\} \subseteq A'$。那么存在两种情况：

-若 b 不被封锁，那么 $\{(a, b) : r, b : C\} \subseteq A''$。

-若 b 被封锁，其先驱 a 不被封锁说明 b 是被在 A' 中存在的 b 的其他先驱 b' 封锁了，且 b' 不被封锁。因此 $(a, b') : r \in A''$，且 $\mathrm{con}_{A'}(b) \subseteq \mathrm{con}_{A'}(b')$，再根据性质 $(*)$，得 $\{(a, b') : r, b' : C\} \subseteq A''$。

在这两种情形下，$\exists$-规则都不能在 A'' 中应用，即 A'' 完全。

(5) 对于 $\forall$-规则：若 $\{a : \forall r.C, (a, b') : r\} \subseteq A''$，则 $a : \forall r.C \in A'$ 且 a 与 b' 都不在 A' 中被封锁。那么存在两种情况：

-若 $(a, b') \in A'$，因为 A' 是完全的，则 $b' : C \in A'$。根据性质 $(*)$，得 $b' : C \in A''$。

-若 $(a, b') \notin A'$，因为 A' 是完全的，则存在 b 使 $(a, b) : r \in A'$，其中 b 在 A' 中被 b' 封锁，且 $b : C \in A'$。根据封锁的定义可知 $b' : C \in A'$，再根据性质 $(*)$，得 $b' : C \in A''$。

在这两种情形下，$\forall$-规则都不能在 A'' 中应用，即 A'' 完全。

根据上述证明，结合由 expand(K) 输出的 A' 是完全且无冲突的，我们可以推出 A'' 亦是完全且无冲突。与在引理 3.4 中的基本理念相同，下面我们为 A'' 构造解释 I：

$$\Delta^I = \{a \mid a\text{是}A''\text{中出现的个体名}\}$$
对任一在 A'' 中出现的个体名 a，$a^I = a$
对任一在 A'' 中的概念名 A，$A^I = \{a \mid A \in \mathrm{con}_{A''}(a)\}$
对任一在 A'' 中出现的关系 r，$r^I = \{(a, b) \mid (a, b) : r \in A''\}$

根据定义 2.5，一个解释 I 是 $K = (T, A)$ 的模型，则 I 需是 T 与 A 的模型。对于 I 是 A''(因此也是 A') 的模型的证明与在引理 3.4 中一致。即，以 A'' 是完全且无冲突的为前提，通过对概念的结构施归纳的方法证明：若 $a : C \in A''$，则 $a^I \in C^I$。根据定义 2.3，若 I 满足 T 中所有的 GCI，则 I 是 T 的模型。由于 T 是被范式化了的，那么对每一 GCI $\top \sqsubseteq D \in T$ 以及 A'' 中每一个体名 a，都有 $a : D \in A''$(否则 $\sqsubseteq$-规则能够应用，与 A'' 是完全的前提相矛盾)。因为 I 是 A'' 的模型，因此有 $a = a^I \in D^I$。又因为 a 是 Δ^I 中的任一元素，因此可得 $\Delta^I \subseteq D^I$，所以 I 也是 T 的模型。那么此解释 I 是 K 的模型，则 K 是一致的。 □

引理 3.11. （完全性）若 K 是一致的，那么 consistent(K) 输出“一致的”。

证明. 与引理 3.5 中的证明理念相同，对 expand 算法的递归使用确保了一致性。“封锁” 这一概念的引入并没有对完全性的证明造成影响，它只是确保了算法的终止（引理 3.9）。

因此具体证明过程的唯一区别在于 $\sqsubseteq$-规则的引入，而此规则保持一致性的证明十分显然：若 $a : C \in A$ 且 $\top \sqsubseteq D \in T$，那么根据定义 2.3 可知, 在任一 (T, A) 的模型 I 中都有 $a^I \in D^I$，因此，I 也是 $(T, A \cup \{a : D\})$ 的模型。 □

定理 3.12. *定义 3.8 中的 tableau 算法是对 $\mathcal{ALC}$ knowledge base 一致性的判定算法。*

证明. 任意一个 $\mathcal{ALC}$ KB 都能被等价地转换为范式化的 KB。定义 3.8 中的算法是范式化 $\mathcal{ALC}$ KB 一致性的判定算法直接由引理 3.9，3.10 与 3.11 推出。 □

4 结论

本文简略介绍了描述逻辑及知识表示与推理系统的重要理论概念及相关的实际应用。主要对 DL 的基本语言 $\mathcal{ALC}$ 进行了详细介绍，特别地，对在 $\mathcal{ALC}$ 的基础上利用 Tableau 算法进行推理的方法进行了详尽描述。

如前文所述，$\mathcal{ALC}$ 是最基本的描述语言，仅由简单的布尔算子作为概念构造算子，因此其所能表达的知识类型相当受限。在实际的 DL 系统应用中，$\mathcal{ALC}$ 这样简单的语言并不能满足实际 DL 系统的使用需求。但是，我们可以通过增加构造算子的方式对 $\mathcal{ALC}$ 的表达力进行扩张，例如：逆关系（r^-），即对关系 r，有关系 r^-，使对 $(a, b) : r$ 有 $(b, a) : r^-$；数量限制 $(\leq n\, r)$、$(\geq n\, r)$，即对概念的 $r-$ 后继进行数量上的限制等等。与此同时，对语言的表达力进行扩张，也就意味着增加了推理的计算复杂度以及知识库的规模。因此，对于每一个构造算子的增加，以及不同构造算子的组合，我们都需要对其推理效率以及完全性、可靠性问题进行分析讨论，从而在推理效率与语言表达力之间达成良好的权衡。

另外，本文所介绍的 Tableau 方法只是解决 DL 推理问题的一条路径，其他的还有：resolution 方法，自动机方法，编写查询方法等等。Tableau 方法在一阶逻辑、模态逻辑中应用经验丰富，自上世纪 90 年代末以来，人们就在 DL 领域对此方法其进行了广泛探索。许多成功的 DL 推理机 (如:RACER, FaCT++,Pellet, Hermit 等) 都利用了 Tableau 技术或其变体。

其优越性在于：Tableau 算法对命题封闭的 DL 有效 (即，运用全布尔运算符的 DL)，且能够在表达力极强的 DL 下保持完全性；此外，通过适当的优化，Tableau 算法在 DL 系统中的实践运行效率良好。

其局限性则主要在于：虽然高度优化的 Tableau 算法在 TBox 推理的应用中很成功，但是面对大体量的本体语言则收效甚微。并且，目前尚未发现有效率地处理大规模 ABox 的 Tableau 路径，也由此使表达力较弱的 DL 重现生机。

参考文献

[1] Ronald J. Brachman, Hector J. Levesque. *Knowledge Representation And Reasoning*. Morgan Kaufmann Publications, US, 2004:1-12.

[2] Franz Baader, Ian Horrocks, Carsten Lutz *et al. An Introduction to Description Logic*. Cambridge University Press, UK, 2017.

[3] Franz Baader, Diego Calvanese, Deborah L. McGuinness *et al. The Description Logic Handbook*. Cambridge University Press, UK, 2003:1-30.

[4] Franz Baader, Ian Horrocks, and Ulrike Sattler. *Chapter 3 Description Logics*. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation* Elsevier, 2007.

[5] Manfred Schmidt-Schauss and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence* 48(1):1–26, 1991.

[6] Kremen P. *Tableau algorithm for ALC*[J]. 2010-11-25. http://cw, felk. cvut. cz.

[7] Baader F, Sattler U. *An overview of tableau algorithms for description logics*[J]. Studia Logica, 2001, 69(1): 5-40.

[8] Baader, Franz, and Ulrike Sattler. *Tableau algorithms for description logics*. International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. Springer, Berlin, Heidelberg, 2000: 1-18.

[9] Donini F M, Massacci F. *EXPTIME tableaux for ALC*[J]. Artificial Intelligence, 2000,87-138.

[10] 跃兴.$\mathcal{ALC}$ 中的 Tableau 算法及其性质 [J]. 计算机应用与软件,2010,27(10):272-274.

[11] 梅婧, 林作铨. 从 $\mathcal{ALC}$ 到 $\mathcal{SHOQ}(\mathcal{D})$: 描述逻辑及其 Tableau 算法 [J]. 计算机科学, 2005(03): 1-11+35.