UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENAN LEANDRO FERNANDES

Connection Method for Defeasible Description Logics

Recife

2024

RENAN LEANDRO FERNANDES

Connection Method for Defeasible Description Logics

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de doutor em Ciência da Computação.

Área de Concentração: Teoria da Computação

Orientador: Fred Freitas

Coorientador: Ivan Varzinczak

Recife

2024

.Catalogação de Publicação na Fonte. UFPE - Biblioteca Central

Fernandes, Renan Leandro.
   Connection Method for Defeasible Description Logics / Renan Leandro Fernandes. - Recife, 2024.
   98f.: il.

   Tese (Doutorado) - Universidade Federal de Pernambuco, Centro de Informática, Programa de Pós-Graduação em Ciências da Computação, 2024.
   Orientação: Frederico Luiz Gonçalves de Freitas.
   Coorientação: Ivan José Varzinczak.
   Inclui referências e apêndices.

   1. Connection method; 2. Description logics; 3. Defeasible reasoning. I. Freitas, Frederico Luiz Gonçalves de. II. Varzinczak, Ivan José. III. Título.

UFPE-Biblioteca Central

Dedico este trabalho aos que vieram antes de mim e desbravaram os caminhos para que eu pudesse trilhá-los. Aos meus pais e meus avós.

ACKNOWLEDGEMENTS

Estes cinco anos foram sem dúvida os anos mais difíceis da minha vida. Eu agradeço profundamente a todas as pessoas que passaram por ela neste período e contribuiram direta ou indiretamente para a conclusão deste doutorado.

Agradeço aos meus pais, que me ensinaram o valor da educação e do esforço. Nos meus piores momentos, me mostraram que eu nunca estive só.

Agradeço aos meus familiares pelo suporte e carinho. Em especial, ao meu irmão, por entender as minhas ausências. Agora terei mais tempo para ficar com meus sobrinhos!

Agradeço também à minha namorada Adriana. Dri, você me mostrou que eu deveria continuar, nos momentos em que eu achei que não conseguiria. Você abriu mão dos finais de semana, do seu tempo de qualidade, para estar ao meu lado nesses últimos meses em que fiquei trancado finalizando este trabalho. Muito obrigado por ter ficado ao meu lado durante todos esses momentos. Que possamos aproveitar nossos finais de semana agora.

Agradeço aos meus orientadores Fred e Ivan. Obrigado pela paciência, pelos ensinamentos e também pelo carinho com que me trataram durante este período. Espero que nossa parceira continue por muitos anos. Agradeço também a Pedro por ter sido um grande apoio e um excelente revisor deste trabalho.

Agradeço às pessoas que integram o Centro de Informática e a Universidade Federal de Pernambuco por todo o suporte e assistência.

Por fim, agradeço aos amigos. Muito obrigado pelo apoio e por me ajudarem a não desistir deste sonho.

RESUMO

A modelagem de exceções em ontologias, proporcionada através do uso de lógicas anuláveis, e o raciocínio em sua presença recebeu uma significativa atenção na última década. O desenvolvimento de métodos de prova para as Lógicas de Descrições (DLs) anuláveis, seguindo os métodos para as DLs clássicas, é principalmente baseado em tableaux semânticos. No entanto, a literatura apresenta sistemas de inferência alternativos igualmente viáveis para o desenvolvimento de provadores de teoremas automáticos, como o método de conexões. Este método consiste em um algoritmo de busca de prova orientado a um objetivo, buscando por conexões (pares de literais complementares) em conjuntos de cláusulas de literais, chamada de matriz. Esta tese apresenta um método de conexões para uma família de DLs tolerante a exceções. O trabalho apresenta as seguintes contribuições: (i) definição de uma representação matricial de uma base de conhecimento que estabelece condições para que um dado axioma seja provado pela matriz; (ii) definição de uma condição de bloqueio na presença de operadores de tipicalidade; (iii) fornecimento de um vínculo entre as estruturas matriciais do método proposto e a semântica de DLs anuláveis; (iv) provas de corretude, completude e terminação para o sistema de inferência proposto, dependendo apenas da semântica das lógicas de descrições anuláveis; e (v) uma arquitetura de provadores de métodos de conexões polimórficos, o PolyCoP, desenvolvido para a linguagem $\mathcal{ALCH}^{\bullet}$. Tal arquitetura pode abranger possivelmente qualquer outra lógica, com modificações sutis em seus métodos e classes.

Palavras-chaves: Método de conexões. Lógicas de descrição. Raciocínio não-monotônico.

ABSTRACT

The modelling of exceptions in ontologies, provided through defeasible logics, and the reasoning behind their presence have received significant attention in the last decade. The development of proof methods for defeasible Description Logics (DLs), following the methods for classical DLs, is mainly based on semantic tableaux. However, the literature offers equally viable alternatives for developing automatic theorem provers, such as the connection method. This method consists of a goal-oriented proof search algorithm for connections (pairs of complementary literals) in sets of literal clauses called a matrix. This thesis presents a connection method for a family of exception-tolerant DLs. The work presents the following contributions: (i) definition of a matrix representation of a knowledge base that establishes conditions for a given axiom to be provable by the matrix; (ii) definition of a blocking condition in the presence of typicality operators; (iii) providing a bond between the matrix structures of the proposed method and the semantics of defeasible DLs; (iv) proofs of correctness, completeness and termination for the proposed inference system, grounded only on the semantics of defeasible description logics; and (v) an architecture of polymorphic connection method provers, PolyCoP, developed for the $\mathcal{ALCH}^{\bullet}$language. Such an architecture can encompass any other logic, with subtle modifications in its methods and classes.

Keywords: Connection method. Description logics. Defeasible reasoning.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\square$	End of proof
$\blacksquare$	End of definition
$\triangle$	End of example

CONTENTS

1 INTRODUCTION

Modelling exceptions in ontologies and reasoning in their presence has been an active area over the past decade. Among the emblematic approaches put forward in the literature feature, Giordano et al.'s description logics of typicality (GIORDANO et al., 2007; GIORDANO et al., 2009; GIORDANO et al., 2015), Britz et al.'s defeasible subsumption relations (BRITZ; HEIDEMA; MEYER, 2008; BRITZ et al., 2021), Bonatti et al.'s light-weight DLs of normality (BONATTI; FAELLA; SAURO, 2011; BONATTI et al., 2015; BONATTI; SAURO, 2017), Bozzato et al.'s reduction to Datalog (BOZZATO; EITER; SERAFINI, 2018), besides Casini and Straccia's seminal work on the computational counterpart of non-monotonic entailment in DLs (CASINI; STRACCIA, 2010; CASINI; STRACCIA, 2013) along with its implementation (CASINI et al., 2015). These investigations have given rise to a whole family of defeasible description logics of varying expressive power and with the ability to handle exceptions at both the *modelling* and the *reasoning* levels in a number of ways (BONATTI, 2019; BRITZ; VARZINCZAK, 2017b; CASINI et al., 2014; CASINI; STRACCIA; MEYER, 2019; PENSEL; TURHAN, 2018).

One of the interesting characteristics of some of the approaches mentioned above is the fact that depending on the underlying DL that is assumed and given certain conditions on how exceptionality (or typicality) is expressed, the kind of non-monotonic reasoning that is performed can be reduced to (a polynomial number of calls to) classical entailment check. Therefore, studying automated deduction for the various flavours of defeasible DLs and its potential reduction to classical reasoning remains a relevant and active research topic in logic-based artificial intelligence.

The development of proof methods for defeasible description logics has followed those for classical DLs. As a result, semantic tableaux are the basis of the overwhelming majority of existing decision procedures for reasoning with defeasible ontologies (BRITZ; VARZINCZAK, 2017b; BRITZ; VARZINCZAK, 2019; GIORDANO et al., 2009; GIORDANO et al., 2013; VARZINCZAK, 2018). Despite the commonly extolled virtues of tableau systems, there are equally viable alternatives in the literature on automated theorem proving (ATP). One example is the connection method (CM) (BIBEL, 1987), initially defined by W. Bibel in the late '70s, which earned a good reputation in the field of ATP in the '80s and '90s. In particular, recent research has revived the connection method in the context of (classical) modal and description logics (OTTEN, 2012; OTTEN, 2022; FREITAS; OTTEN, 2016; FREITAS; VARZINCZAK, 2018).

CM's main internal structure is a matrix representation of the knowledge base and its associated query. Such representation allows for a more parsimonious usage of memory during proof search. Indeed, unlike tableaux and resolution, the connection method does not create intermediate clauses or sentences along the way, keeping its search space confined to the boundaries of the matrix it started. The first connection calculus for (classical) description logics, $\mathcal{ALC}\ \theta - CM$ (FREITAS; OTTEN, 2016), incorporates several features of most DL proof systems, such as blocking, lack of variables and Skolem functions. Moreover, a C++ implementation of the method, RACCOON (FILHO; FREITAS; OTTEN, 2017), has been developed.[1] Worthy of mention is the fact that, despite incorporating *none* of the optimisations commonly embedded in DL tableaux systems, RACCOON performed competitively in reasoning over $\mathcal{ALC}$ ontologies when compared to cutting-edge highly-optimised tableau-based reasoners, which had ranked high in DL reasoners' past competitions.[2]

1.1 SCOPE OF THE THESIS

As mentioned in the previous section, some efforts have been put forward to find ways to model exceptions in DLs. Among the defeasible logics present in the literature, $\mathcal{ALCH}^{\bullet}$ was the one that glimpsed our attention the most because (i) it has greater expressiveness in the way it represents concepts and axioms in opposition to the others - it allows, for example, the use of a typicality operator on both sides of subsumption - and (ii) it has extensions (BRITZ; VARZINCZAK, 2017b; BRITZ; VARZINCZAK, 2017a) already published, allowing us to advance our proof method for greater expressiveness in the future.

In this direction, this work focuses on the defeasible DL $\mathcal{ALCH}^{\bullet}$ (VARZINCZAK, 2018), an extension of $\mathcal{ALCH}$ with typicality operators on complex concepts and role names. Intuitively, the first one denotes the most typical objects in the class, and the other represents the most typical relationship instances. We give more detail about the language in Chapter 4.

Although most decision procedures for reasoning with ontologies rely on tableaux, this thesis focuses instead on the connection method. To the best of our knowledge, only two connection methods have been proposed for description logics: $\mathcal{ALC}\ \theta - CM$ (FREITAS; OTTEN, 2016), $\mathcal{ALCHQ}^{=}\ \theta - CM$ (FREITAS; VARZINCZAK, 2018), and $\mathcal{ALC}^{\bullet}\ \theta - CM$ (FERNANDES; FREITAS; VARZINCZAK, 2021).

[1] <https://github.com/dmfilho/raccoon>

[2] See <https://goo.gl/V9Ewkv> for details on the comparison.

$\mathcal{ALC}$ $\theta-CM$ (FREITAS; OTTEN, 2016) works over $\mathcal{ALC}$, introducing blocking and replacing the usage of Skolem terms and unification by θ-substitutions, which prevent connections in the same way as unification. $\mathcal{ALCHQ}^{=}$ $\theta - CM$ (FREITAS; VARZINCZAK, 2018) handles equality and number restrictions for concept and role names, the connectives for $\mathcal{ALCHQ}^{=}$.

$\mathcal{ALC}$ $\theta-CM$ and $\mathcal{ALCHQ}^{=}$ $\theta-CM$ soundness and completeness proofs take advantage of the functional equivalence between them and the connection method for first-order logic. However, we advocate the inclusion of proofs that depend only on DL, as demonstrated by Baader et al. for the tableau method (BAADER; NUTT, 2007). Therefore, we construct the proofs in this thesis without using the abovementioned equivalences. Such demonstrations can be the foundation for more expressive languages, providing more understandable theorems with less dependency on other proof theory backgrounds as first-order logic.

Furthermore, such demonstrations, less dependent on other logics or previous knowledge, have educational value and can provide new researchers with core information and tips for implementing new reasoners and optimisations in the near future, promoting a sense of learning and growth in the community. Therefore, we focus on providing soundness and completeness proofs for our connection method based only on defeasible DL semantics.

1.2 OBJECTIVES

The main goal of this thesis is **to propose a sound and complete connection method for $\mathcal{ALCH}^{\bullet}$, whose proofs are grounded on description logic proof theory**. Hereafter, we outline the specific objectives of this research:

1. Conceiving a matrix representation considering the typicality relation between individuals and pairs of individuals of the ontology;
2. Proposing a matrix characterisation showing that a matrix is complementary iff the ontology which generates it is valid;
3. Defining a sound and complete connection calculus for $\mathcal{ALCH}^{\bullet}$;
4. Defining an algorithm based on the connection calculus for $\mathcal{ALCH}^{\bullet}$ such that it is sound, complete and terminates in all cases;
5. Proposing self-contained demonstrations of our method, grounded on the semantics of description logic; and

6. Implementing a prover for $\mathcal{ALCH}^{\bullet}$.

1.3 RESULTS

We started our research by proposing a connection method à la Bibel for an exception-tolerant extension of $\mathcal{ALC}$ (FERNANDES; FREITAS; VARZINCZAK, 2021). As for the language, we assumed a fragment of $\mathcal{ALCH}^{\bullet}$, i.e. $\mathcal{ALC}$ extended with a typicality operator for concepts. We revisited the definition of a matrix representation of a knowledge base and established the conditions for a given axiom to be provable. In particular, we have shown how term substitution is dealt with and defined a suitable blocking condition in the presence of typicality operators.

Then, the main idea was to extend the proposed connection method to handle the typical operator on role names. However, dealing with variables and relations was a complex and non-intuitive approach for defining the method, as proposed previously for $\mathcal{ALC}^{\bullet}$. We improved the proposed representation by including preferential relations as literals in the matrix. This differs from the previous version for $\mathcal{ALC}^{\bullet}$, which treated the relation as an auxiliary structure to the matrix. Our modification allowed us to demonstrate our approach's soundness, completeness, and termination, grounded on DL.

Finally, we developed a connection prover for $\mathcal{ALCH}^{\bullet}$, called PolyCoP. Despite its primary purpose, we went further to design PolyCoP to be user-friendly, providing a polymorphic prover framework built on top of design patterns. This design separates the logical language-related code from the structural connection method, allowing one to focus only on the (rather slight) differences in literal representation, unification, and blocking strategies from each logic. This approach is intended to help logic beginners learn the connection method straightforwardly, making the use and the building of connection provers an intuitive experience. Furthermore, our framework is educational. The code, as proposed, provides a way to clearly check the difference between the connection method and the other inference methods.

One of PolyCoP's main features is its tree representation of the proof. Following the principles of polymorphism and reuse that guide PolyCoP, it is possible to read the proof through this tree and to develop different outputs according to the user's needs. We developed a code capable of representing the proof through LaTeX systems to demonstrate this feature. In this way, one can effectively execute, search for, and visualize the proofs.

1.4 ROADMAP

The remaining chapters of this thesis proposal are organized as follows:

- **Chapter 2 - Description logics:** we introduce the main features of description logics. We show the components, the languages, the semantics and important reasoning tasks;
- **Chapter 3 - Connection methods:** we show connection methods principles and they implementation for propositional, first-order, and a family of description logics;
- **Chapter 4 - Defeasible description logics:** we cover the language $\mathcal{ALCH}^{\bullet}$ (VARZINCZAK, 2018) which we focus in this thesis. We also show related work in defeasible description logics' field;
- **Chapter 5 - A connection method for a defeasible extension of $\mathcal{ALCH}$:** we present our connection method;
- **Chapter 6 - PolyCoP:** we show our implementation of connection calculus for $\mathcal{ALCH}^{\bullet}$ and other logic;
- **Chapter 7 - Related work:** we outline the related work of our proposal.
- **Chapter 8 - Conclusion:** we conclude the thesis, discussing foreseen contributions and future work.

Part I

Preliminaries

2 DESCRIPTION LOGICS

Description logics are formalisms to describe domains of interest based on their elements (BAADER et al., 2017). They are the core of the foundation of Semantics Web's Ontology Language (BERNERS-LEE; HENDLER; LASSILA, 2001).

In DLs, the key components are the elements (individuals), sets of elements (concepts) and sets for binary relations of elements (roles). Hereafter, we assume finite sets of concepts, roles, and individual names, denoted resp., with C, R, and I. We denote atomic concepts with $A, B, \ldots$, with $r, s, \ldots$ role names, and with $a, b, \ldots$ individual names.

2.1 CONCEPTS AND THEIR SEMANTICS

We have a formal language for each DL that allows us to build concepts and role descriptions. One well-known Description Logic is the $\mathcal{ALC}$ that allows representation of complex concepts built using the constructors $\neg$ (complement), $\sqcap$ (conjunction), $\sqcup$ (disjunction), $\forall$ (universal restriction) and $\exists$ (existential restriction). Complex concepts of $\mathcal{ALC}$ are denoted with $D, E, \ldots$ and are built according to the following grammar:

$$D, E ::= \mathrm{C} \mid \top \mid \bot \mid \neg D \mid D \sqcap E \mid D \sqcup E \mid \forall \mathrm{R}.D \mid \exists \mathrm{R}.D$$

Example 2.1. The following expressions are $\mathcal{ALC}$ concepts:

- $A \sqcap B \sqcup \exists r.C$;
- $B \sqcup \neg \forall r.(\exists s(D \sqcap \neg A))$;

△

In this chapter, we revisit the Description Logic $\mathcal{ALCH}$, an extension of $\mathcal{ALC}$ that allows representation for role hierarchies. The semantics of $\mathcal{ALCH}$ is the Tarskian semantics and is defined as follows.

Definition 2.1 (Semantics)**.** An *interpretation* is a structure $\mathcal{I} := \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ where $\Delta^{\mathcal{I}}$ is the domain of interpretation, and $\cdot^{\mathcal{I}}$ is the interpretation function that maps:

- concept names A to subsets $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ of the domain interpretation,
- role names r to binary relations $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ of the domain interpretation and

- individual names a to elements $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ of the domain interpretation.

The interpretation $\mathcal{I}$ interprets $\mathcal{ALCH}$ concepts in the following way:

$$\top^{\mathcal{I}} \stackrel{\text{def}}{=} \Delta^{\mathcal{I}}$$

$$\bot^{\mathcal{I}} \stackrel{\text{def}}{=} \emptyset$$

$$(D \sqcap E)^{\mathcal{I}} \stackrel{\text{def}}{=} D^{\mathcal{I}} \cap E^{\mathcal{I}}$$

$$(D \sqcup E)^{\mathcal{I}} \stackrel{\text{def}}{=} D^{\mathcal{I}} \cup E^{\mathcal{I}}$$

$$(\neg D)^{\mathcal{I}} \stackrel{\text{def}}{=} \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}}$$

$$(\forall r.D)^{\mathcal{I}} \stackrel{\text{def}}{=} \{a, b \in \Delta^{\mathcal{I}} \mid \text{for every } b, \text{ if } (a,b) \in r^{\mathcal{I}}, \text{ then } b \in D^{\mathcal{I}}\}$$

$$(\exists r.D)^{\mathcal{I}} \stackrel{\text{def}}{=} \{a, b \in \Delta^{\mathcal{I}} \mid \text{for some } b, (a,b) \in r^{\mathcal{I}} \text{ and } b \in D^{\mathcal{I}}\}$$

■

2.2 AXIOMS

In Description Logics, the facts are expressed as axioms. Those facts are separate in TBox (for general concept inclusions), RBox (for role inclusion axioms) and ABox (for assertions).

Definition 2.2 (General concept inclusion)**.** Let D, E be $\mathcal{ALCH}$ concepts. An axiom $D \sqsubseteq E$ is called a *general concept inclusion* (GCI). ■

Definition 2.3 (Role inclusion axiom)**.** Let r, s be role names. An axiom of the form $r \sqsubseteq s$ is called a *role inclusion axiom* (RIA). ■

Definition 2.4 (Concept assertion)**.** Let D be an $\mathcal{ALCH}$ concept and a be an individual name. An axiom of the form $D(a)$ is called a *concept assertion*. ■

Definition 2.5 (Concept assertion)**.** Let R be an $\mathcal{ALCH}$ role and a, b be individual names. An axiom of the form $R(a, b)$ is called a *role assertion*. ■

Definition 2.6 (TBox, RBox, ABox and knowledge base)**.** A *TBox* $\mathcal{T}$ is a set of GCIs. A *RBox* $\mathcal{R}$ is a set of RIAs. An *ABox* $\mathcal{A}$ is a set of (concept and role) assertions. We denote $\mathcal{K} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$ as a *knowledge base* (KB). ■

Example 2.2. Let $\mathrm{C} = \{\mathrm{Animal}, \mathrm{Cat}, \mathrm{CatOwner}, \mathrm{CatLover}\}$ be the set of concepts denoting animals, cats, people who own cats, and people who love cats, respectively; $\mathrm{R} =$

$\{\mathrm{hasPet}, \mathrm{hasCat}\}$ be the set of roles denoting who has a pet and who has a cat respectively; and $\mathrm{I} = \{\mathrm{renan}, \mathrm{darwin}\}$ denoting elements in this domain.

With $\mathcal{T}$, we define the following TBox:

$$\mathcal{T} = \left\{ \begin{array}{c} \exists \mathrm{hasCat}.\mathrm{Cat} \sqsubseteq \mathrm{CatOwner}, \\ \mathrm{CatLover} \sqsubseteq \exists \mathrm{hasCat}.\mathrm{Animal}, \\ \mathrm{CatLover} \sqsubseteq \forall \mathrm{hasPet}.\mathrm{Cat} \end{array} \right\}$$

representing, respectively, that "who has a cat is a cat owner", "cat lover is who has an animal as a cat", and "cat lover is someone who only has cats as a pet". With $\mathcal{R}$, we define the following RBox:

$$\mathcal{R} = \left\{ \mathrm{hasCat} \sqsubseteq \mathrm{hasPet} \right\}$$

representing that "who has a cat is someone who has a pet". Lastly, with $\mathcal{A}$, we define the following ABox:

$$\mathcal{A} = \left\{ \begin{array}{c} \mathrm{Cat}(\mathrm{darwin}), \\ (\mathrm{Animal} \sqcap \neg \mathrm{Cat})(\mathrm{renan}), \\ \mathrm{hasCat}(\mathrm{renan}, \mathrm{darwin}) \end{array} \right\}$$

representing, respectively, that "darwin is a cat", "renan is an animal and is not a cat", and "renan has darwin as his cat". In general, a role assertion is read assuming that the first argument represents the subject and the last represents the object of a sentence. △

Definition 2.7 (Satisfaction)**.** Let $\mathcal{I}$ be an interpretation; D, E be $\mathcal{ALCH}$ concepts; r, s be role names; and a, b be individual names. The satisfaction relation $\Vdash$ is defined as follows:

- $\mathcal{I} \Vdash D \sqsubseteq E$ if $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$;
- $\mathcal{I} \Vdash r \sqsubseteq s$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$;
- $\mathcal{I} \Vdash D(a)$ if $a^{\mathcal{I}} \in D^{\mathcal{I}}$; and
- $\mathcal{I} \Vdash r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$.

■

If an interpretation $\mathcal{I}$ satisfies every GCI (RIA) in a TBox $\mathcal{T}$ (RBox $\mathcal{R}$) we say that $\mathcal{I}$ is a model of $\mathcal{T}$ ($\mathcal{R}$). Thus, if an interpretation $\mathcal{I}$ satisfies $\mathcal{K}$ we say $\mathcal{I}$ is a model of $\mathcal{K}$.

2.3 REASONING

The main advantage of DLs as a logic-based formalism is the capability to entail new facts from those modelled. The common reasoning tasks for DLs are checking the satisfiability of concepts, subsumption of concepts and roles, and consistency of knowledge bases.

Definition 2.8 (Entailment)**.** Let $\mathcal{KB}$ be an $\mathcal{ALCH}$ knowledge base and α be an axiom (GCI, RIA or an assertion). We say $\mathcal{KB}$ *entails* α, denoted with $\mathcal{KB} \models \alpha$, if $\mathcal{I} \Vdash \mathcal{KB}$ only if $\mathcal{I} \Vdash \alpha$, for every interpretation $\mathcal{I}$. ■

Example 2.3. Assume $\mathcal{K} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$ be a knowledge base composed by the TBox, RBox and ABox from Example 2.2. We have that:

- $\mathcal{K} \models \mathrm{Animal}(\mathrm{renan})$ since renan is an animal and not a cat;
- $\mathcal{K} \models \mathrm{hasPet}(\mathrm{renan}, \mathrm{darwin})$ since $\mathrm{hasCat} \sqsubseteq \mathrm{hasPet} \in \mathcal{R}$ and $\mathrm{hasCat}(\mathrm{renan}, \mathrm{darwin}) \in \mathcal{A}$;
- $\mathcal{K} \models \mathrm{CatOwner}(\mathrm{renan})$ since who has a cat is a cat owner and renan has a cat (darwin);
- $\mathcal{K} \not\models \mathrm{CatLover}(\mathrm{renan})$ since is not sure that every pet of renan is a cat.

△

Definition 2.9 (Basic reasoning tasks)**.** Let $\mathcal{K} = \mathcal{T} \cup \mathcal{R} \cup \mathcal{A}$ be a knowledge base; D and E be $\mathcal{ALCH}$ concepts; r and s be role names; and a and b be individual names. We say that:

- D is *satisfiable* with respect to $\mathcal{T}$ if there exists an interpretation $\mathcal{I}$ s.t. $a^{\mathcal{I}} \in D^{\mathcal{I}}$, for some $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$;
- D is *subsumed* by E with respect to $\mathcal{T}$ if $D^{\mathcal{I}} \subseteq E^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{T}$;
- r is *subsumed* by s with respect to $\mathcal{R}$ if $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$, for every model $\mathcal{I}$ of $\mathcal{R}$;
- $\mathcal{K}$ is *consistent* if there exists a model of $\mathcal{K}$;
- $\mathcal{K}$ is *valid* if each interpretation $\mathcal{I}$ is a model of $\mathcal{K}$.

■

With those basic reasoning tasks, we can specify more complex reasoning tasks as *classification* of a TBox, *instance retrieval* and *realisation* (BAADER et al., 2017).

2.4 CONCLUNDING REMARKS

Description logics are well-known formalisms and the core of Semantic Web (BERNERS-LEE; HENDLER; LASSILA, 2001). They are built with elements, concepts, and roles categorised by their power of expressiveness.

In the next chapter, we discuss a reasoning method for description logic called the connection method. It is a direct proof method that can check whether a fact is entailed from a knowledge base.

3 A CONNECTION METHOD FOR $\mathcal{ALCH}$

In this chapter, we revisit the core definitions and theorems of a connection method for description logics. In order to illustrate the target language $\mathcal{ALCH}$ we present a simpler version of $\mathcal{ALCHQ}^{=}\ \theta - CM$ (FREITAS; VARZINCZAK, 2018), excluding cardinality restrictions and equalities of terms.

The connection method is a direct method in the sense that it proves the validity of formulae instead of searching for inconsistencies, as in refutation methods, such as resolution and tableaux. To check whether $\{\alpha, \beta\} \models \gamma$ in first-order logic (FOL), the validity of the formula $\models (\alpha \wedge \beta) \rightarrow \gamma$ must be proven, as the well-known Deduction theorem establishes. Therefore, the formula $\neg\alpha \vee \neg\beta \vee \gamma$ (a disjuction) must be valid. Thus, the effects for the validity procedure are: (*i*) the formula to be proven must be in Disjunctive Normal Form (DNF); (*ii*) premises are negated (*iii*) free variables are existentially quantified; (*iv*) FOL Skolemization applies over universally-quantified variables; and (*v*) the conclusion is not negated (FREITAS; OTTEN, 2016).

3.1 NORMAL FORM

A connection method represents a formula as a matrix. However, normal forms are required to represent those formulas on the matrix. The connection method for FOL (BIBEL, 1987) requires formulas in the Disjunctive Normal Form (DNF) to represent them. The connection method for $\mathcal{ALC}$ requires axioms in two-lined disjunctive normal form. This form easily converts axioms to matrices, avoiding complex translation and unnecessary repetition of literals. However, the translation still exponentially increases the matrix representation.

Definition 3.1 (Pure conjunction)**.** An $\mathcal{ALCH}$ concept is a *pure conjunction* if it is defined by the following grammar:

$$\hat{E} ::= \mathrm{C} \mid \neg\mathrm{C} \mid \hat{E} \sqcap \hat{E} \mid \exists \mathrm{R}.\hat{E}$$

■

Example 3.1 (Pure conjunction)**.** The following examples are pure conjunctions:

- A
- $\neg A$

- $A \sqcap B \sqcap (C \sqcap \exists r.(D \sqcap E))$
- $\exists r.(A \sqcap \exists s.B)$

On the other hand, the following examples are not pure conjunctions:

- $A \sqcup B$
- $A \sqcap B \sqcap (C \sqcap \forall r.(D \sqcap E))$
- $\forall r.A$

△

Definition 3.2 (Pure disjunction)**.** An $\mathcal{ALCH}$ concept is a *pure disjunction* if it is defined by the following grammar:

$$\check{E} ::= \mathrm{C} \mid \neg\mathrm{C} \mid \check{E} \sqcup \check{E} \mid \forall \mathrm{R}.\check{E}$$

■

Example 3.2 (Pure disjunction)**.** The following examples are pure disjunctions:

- A
- $\neg A$
- $A \sqcup B \sqcup (C \sqcup \forall r.(D \sqcup E))$
- $\forall r.(A \sqcup \forall s.B)$

On the other hand, the following examples are not pure disjunctions:

- $A \sqcap B$
- $A \sqcup B \sqcup (C \sqcup \exists r.(D \sqcup E))$
- $\exists r.A$

△

The primary reason for defining purity in conceptual expressions is to prevent operators representing disjunctions from combining with operators representing conjunctions. The definitions lead to literal concepts, such as A or $\neg A$, classified as pure disjunctions and conjunctions. We allow such classification because their representation does not impact the complexity of generating the corresponding matrices.

Definition 3.3 (Two-lined disjunctive normal form)**.** Let A be a concept name or its negation; D and E be $\mathcal{ALCH}$ concepts; and a be an individual name. A GCI is in *two-lined disjunctive normal form* (two-lined DNF) if it is in one of the following normal forms:[1]

- $\hat{D} \sqsubseteq \check{E}$;
- $A \sqsubseteq \hat{D}$;
- $\check{D} \sqsubseteq A$;

A concept assertion is in *two-lined disjunctive normal form* if its concept is a literal. ■

Example 3.3. Let $\mathrm{C} = \{A, B, C\}$, $\mathrm{R} = \{r, s\}$ and $\mathrm{I} = \{a, b, c\}$ be the set of concepts, the set of role names and the set of individual names, respectively. We have that:

- A is a pure conjunction and a pure disjunction;
- $A \sqcap (B \sqcap C)$ is a pure conjunction;
- $A \sqcap (B \sqcup C)$ is not a pure conjunction, i.e., is an impure conjunction;
- The GCI $A \sqsubseteq \exists r.(B \sqcap C)$ is in two-lined DNF;
- The assertion $(B \sqcup C)(a)$ is not in two-lined DNF.

△

Example 3.4. Let $\mathcal{K} = \mathcal{T} \cup \mathcal{R}$ be the following knowledge base:

$$\mathcal{T} = \left\{ \begin{array}{c} \exists \mathrm{hasCat.Cat} \sqsubseteq \mathrm{CatOwner}, \\ \mathrm{CatLover} \sqsubseteq \exists \mathrm{hasCat.Animal}, \\ \mathrm{CatLover} \sqsubseteq \forall \mathrm{hasPet.Cat} \end{array} \right\}$$

$$\mathcal{R} = \left\{ \mathrm{hasCat} \sqsubseteq \mathrm{hasPet} \right\}$$

The knowledge base $\mathcal{K}$ is already in two-line DNF. △

[1] We omit RBox axioms since in $\mathcal{ALCH}$ they are already in two-lined DNF.

3.2 MATRIX REPRESENTATION

The matrix representation of a formula (or a set of axioms) is the clausal form of it. A clause is a conjunction of literals, and a matrix is a disjunction of clauses. [2]

Definition 3.4 (Literal)**.** Let C be a set of concept names, R be a set of role names, and I be a set of individual names. A *literal* in $\mathcal{ALCH}$ is defined as follows:

$$L ::= \mathrm{C} \mid \mathrm{R} \mid \mathrm{C(I)} \mid \mathrm{R(I,I)} \mid \neg L$$

■

In addition to the representation indicated by the grammar above, we can associate literals with horizontal or vertical lines. This occurs when the literal represents restrictions of the form $\forall r.D$ or $\exists r.D$. A vertical line is added to the literals to map existential restriction and show their relationship. Similarly, a horizontal line is added to the related literals when mapping universal restriction. This is because universal restrictions have OR-based semantics, and since the matrix represents the disjunctions in separate columns, these literals are also separated into distinct clauses.

Definition 3.5 (Matrix representation)**.** Let A be an atomic concept or its negation; D be a concept (possibly complex); r and s be role names; a and b be individual names; $\hat{D} = D_1 \sqcap \cdots \sqcap D_n$ be a pure conjunction; and $\check{E} = E_1 \sqcup \cdots \sqcup E_m$ be a pure disjunction. The matrix and graphical representation of two-lined disjunctive normal forms are shown in Table 1. ■

The Two-lined DNF allows pure conjunctions and pure disjunctions, so a recursive matrix representation for complex concepts is needed. Table 2 shows the sub-cases and their respective matrix and graphical representation.

Example 3.5 (Matrix representation of $\mathcal{ALCH}$ concepts)**.** Assume $\mathcal{K}$ is the knowledge base of Example 3.4. The matrix representation of $\mathcal{K}$ is $\{\{\underline{\mathrm{hasCat}, \mathrm{Cat}}, \neg\mathrm{CatOwner}\}, \{\mathrm{CatLover}, \underline{\neg\mathrm{hasCat}^1}\}, \{\mathrm{CatLover}, \underline{\neg\mathrm{Animal}^1}\}, \{\mathrm{CatLover}, \underline{\mathrm{hasPet}, \neg\mathrm{Cat}}\}, \{\mathrm{hasCat}, \neg\mathrm{hasPet}\}\}$ and its graphical representation is shown in Figure 1. △

[2] As connection method is a direct method, this notion of clauses and matrix is dual to refutation methods as resolution or tableaux where a clause is a disjunction of literals.

Table 1 – Normal forms and their respective matrix and graphical representation.

Normal form	Matrix representation	Graphical representation
$\hat{D} \sqsubseteq \check{E}$	$\{\{D_1, \cdots, D_n, \neg E_1, \cdots, \neg E_m\}\}$	$\begin{bmatrix} D_1 \\ \vdots \\ D_n \\ \neg E_1 \\ \vdots \\ \neg E_m \end{bmatrix}$
$A \sqsubseteq \hat{D}$	$\bigcup_{i=1}^{n} \{\{A, \neg D_i\}\}$	$\begin{bmatrix} A & \cdots & A \\ D_1 & \cdots & D_n \end{bmatrix}$
$\check{E} \sqsubseteq A$	$\bigcup_{i=1}^{m} \{\{E_i, \neg A\}\}$	$\begin{bmatrix} E_1 & \cdots & E_m \\ \neg A & \cdots & \neg A \end{bmatrix}$
$r \sqsubseteq s$	$\{\{r, \neg s\}\}$	$\begin{bmatrix} r \\ \neg s \end{bmatrix}$
$D(a)$	$\{\{D(a)\}\}$	$\begin{bmatrix} \neg D(a) \end{bmatrix}$
$r(a, b)$	$\{\{r(a, b)\}\}$	$\begin{bmatrix} \neg r(a, b) \end{bmatrix}$

$$\left[\begin{array}{ccccc} \text{hasCat} & \text{CatLover} & \text{CatLover} & \text{CatLover} & \text{hasCat} \\ \text{Cat} & \neg\text{hasCat} & \neg\text{Animal} & \text{hasPet} & \neg\text{hasPet} \\ \neg\text{CatOwner} & & & \neg\text{Cat} & \end{array}\right]$$

Figure 1 – Matrix representation of $\mathcal{K}$ (Example 3.4).

As mentioned before, the connection method is a direct proof method and, as such, represents axioms of the knowledge base (the premises of the formula) negated. Hence, concept expressions E of subsumption axioms $D \sqsubseteq E$ are negated in the matrix representation. Furthermore, role subsumptions and assertions follow the same principle of being negated.

Table 2 – Recursive sub-cases of concepts in a two-lined DNF axiom. Let E_i be a concept (possibly complex) in a pure conjunction, pure disjunction, or a restriction as the filler of it.

Sub-case	Matrix representation	Graphical representation
$E_i = A$	$\{\{\cdots, A, \cdots\}\}$	$\begin{bmatrix} \vdots \\ A \\ \vdots \end{bmatrix}$
$E_i = \exists r.A$	$\{\{\cdots, \underline{r}, \underline{A}, \cdots\}\}$	$\begin{bmatrix} \vdots \\ r \\ A \\ \vdots \end{bmatrix}$
$E_i = \forall r.A$	$\{\{\cdots, \underline{r}^j, \cdots\} \cdots \{\cdots, \underline{A}^j, \cdots\}\}$	$\begin{bmatrix} \cdots & \cdots \\ \underline{r} & \underline{\neg A} \end{bmatrix}$

3.3 CHARACTERISATION OF VALIDITY

The goal of the connection method is to prove that a matrix is valid. The connection method for FOL does it by looking for complementary predicates over the whole matrix. In our case, the connection method for $\mathcal{ALCH}$ looks for complementary literals over the whole matrix. To perform that search, we must change our perspective, looking at the matrix horizontally, searching for complementary literals on its paths.

Definition 3.6 (Query)**.** Let $\mathcal{KB}$ be a knowledge base and α be an $\mathcal{ALCH}$ axiom. A *query* α against a knowledge base $\mathcal{KB}$ is a set for which the entailment $\mathcal{KB} \models \alpha$ should be proven. ■

Example 3.6 (Matrix representation of a query)**.** Assume that $\mathcal{K}$ is the knowledge base of Example 3.4. Let $\mathrm{CatLover} \sqsubseteq \mathrm{CatOwner}$ be a query against $\mathcal{K}$. In order to prove that $\mathcal{K} \models \mathrm{CatLover} \sqsubseteq \mathrm{CatOwner}$ the matrix shown in Figure 2 must be proven. △

Definition 3.7 (Path)**.** A *path* through a matrix is a set composed by one literal from each clause of the matrix. ■

Definition 3.8 (Connection)**.** A *connection* is a pair of literals with the same concept/role, but different polarities. ■

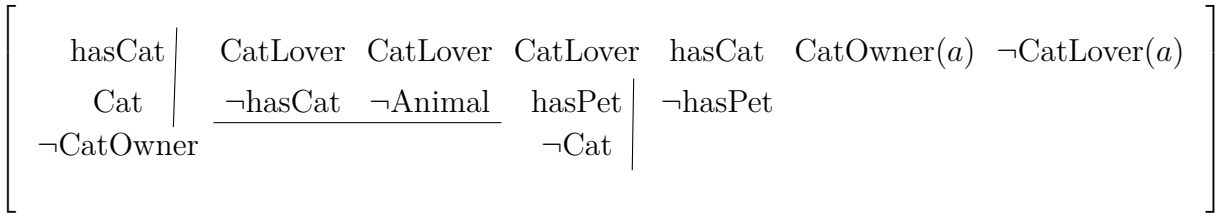


Figure 2 – Matrix representation of $\mathcal{K}$ and a query against it.

Definition 3.9 (Path, matrix and literal complementary)**.** A path is called *complementary* if it contains a connection (as a subset). A matrix is called *complementary* if each of its paths is complementary. The complement $\overline{L}$ of a literal L is $\neg A(a)$ if $L = A(a)$, and it is $A(a)$ if $L = \neg A(a)$ ■

Definition 3.10 (θ-substitution)**.** A θ-substitution is a term unification that assigns each variable to an individual or another variable. ■

Definition 3.11 (θ-complementary connection)**.** A θ-complementary connection is a pair of literals $\{A(x), \neg A(y)\}$ or $\{r(x, z), \neg r(y, k)\}$, with $\theta(x) = \theta(y)$ and $\theta(z) = \theta(k)$. ■

Definition 3.12 (Multiplicity)**.** Let M be a matrix and C be a clause in M. The *multiplicity* is a function $\mu : M \longrightarrow \mathbb{N}$ that assigns to each clause in M a natural number denoting the number of copies of that clause in the matrix. ■

With M^{μ}, we define the union of the matrix M and its μ clause copies. With x^{μ}, we define the term x from the μ-th copy of a clause.

So, the intuition is if we find a θ-complementary connection in a path, then the whole path is valid and we do not need to use it further. If every path in a matrix has a θ-complementary connection, then we are able to establish a relation between the matrix and the axioms which generate it with the following theorem:

Theorem 3.1 (Matrix characterization)**.** *Let $\mathcal{K}$ be a knowledge base, α be an axiom, and M be the matrix representation of the query α against $\mathcal{K}$. We have that $\mathcal{K} \models \alpha$ iff there exists a multiplicity μ, an admissible θ-substitution and a set of connections S such that every path through M^{μ} contains a θ-complementary connection $\{\theta(L_1), \theta(L_2)\} \in S$.*

Example 3.7. In the proof, we use the matrix M from Example 3.6 to show that $\mathcal{K} \models \alpha$. We demonstrate this by establishing a connection for each path of M as shown in Figure 3.

We begin the proof with the sixth clause, $\{\mathrm{CatOwner}(a)\}$. We connect the literal $\mathrm{CatOwner}(a)$ with $\neg\mathrm{CatOwner}$ from the first clause. Since no individual exists in the literal, we substitute with a. From the first clause, the remaining underlined literals are left unchanged. Therefore, no individual is assigned when we connect Cat with $\neg\mathrm{Cat}$ in the fourth clause. We then continue the proof by connecting hasPet with $\neg\mathrm{hasPet}$ from the fifth clause.

It is important to note that there are still some literals to be checked, such as hasCat from the fifth clause, $\mathrm{CatLover}$ from the fourth clause, and hasCat from the first clause. When we connect a literal to its complementary one in a clause, we must prove that the remaining literals of the former clause and the connected clause also have a connection.

The proof continues as we make connections 4, 5, 6, and 7, shown in Figure 3. The matrix is valid once there are no remaining literals; therefore, $\mathcal{K} \models \alpha$. △

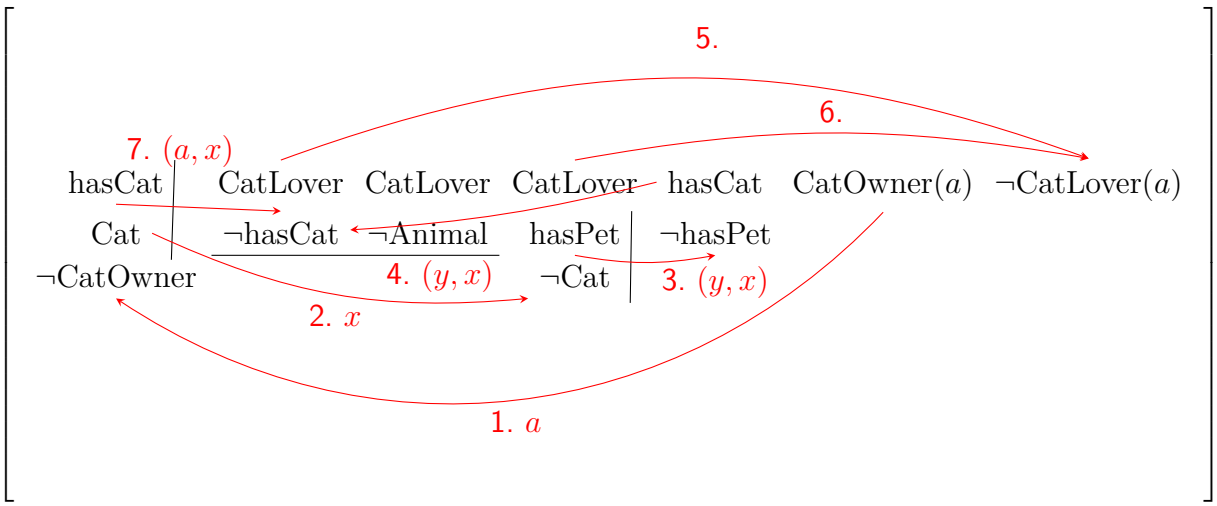


Figure 3 – Proof in matrix-style of M from Example (3.6).

3.4 THE CALCULUS

Even we establish the relation between connections in the matrix and the validity of axioms that generate it, the matrix characterisation does not afford a procedure that looks for connections in a matrix.

Thus, we define a proof calculus to provide a direct proof procedure. The basic structure (i.e., word) of the proof calculus is a triple $\langle C, M, Path\rangle$, where (sub-)clause C is the open

sub-goal, M is the matrix, and $Path$ is the active (sub-)path The rules of the calculus are applied in an analytical, bottom-up way. The rules and axiom are shown in Figure 4.

Definition 3.13 (Set of atomic concepts)**.** Let C be the set of concept names of a vocabulary. The set of atomic concepts of C is $\mathrm{A_C} \stackrel{\text{def}}{=} \mathrm{C} \cup \{\neg A \mid A \in \mathrm{C}\}$. ■

Definition 3.14 (Set of concepts)**.** The *set of concepts* $\tau(x, Path)$ of a term x in a path $Path$ that contains all atomic concepts instantiated by x is defined as $\tau(x, Path) \stackrel{\text{def}}{=} \{D \in \mathrm{A_C} \mid D(x) \in Path\}$. ■

Definition 3.15 (Skolem condition)**.** Let $\#\{\cdot\}$ be the cardinality of a set and C be a set of concept names. A path $Path$ is ensured by the *Skolem condition* if at most one atomic name is underlined for each term in graphical matrix representation. This condition is formally defined as $\forall i \forall a \#\{\underline{E^i} \in \mathrm{C} \mid \underline{E^i(a)} \in Path\} \leq 1$. ■

The $\mathcal{ALC}$ fragment of DL is finite. Then, every search performed in the connection method must terminate, even when cycles occur. The method detects cycles by analyzing the clauses copied during the proof. Therefore, we formally define a blocking condition for the calculus.

Definition 3.16 (Blocking condition)**.** Given a path $Path$, a θ-substitution, a multiplicity μ, a new individual $\theta(x^{\mu})$, and the previous individual $\theta(x^{\mu-1})$; the *blocking condition* holds if $\tau(\theta(x^{\mu}), Path) \subseteq \tau(\theta(x^{\mu-1}), Path)$. ■

$$\textit{Axiom (Ax)}\ \frac{}{\{\}, M, Path}$$

$$\textit{Start Rule (St)}\ \frac{C_1, M, \{\}}{\epsilon, M, \epsilon}, \text{with } C_1 \in M$$

$$\textit{Reduction Rule (Red)}\ \frac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}},\ \text{with } L_1 = \overline{L_2} \text{ or } L_1 = \top(a)$$

$$\textit{Extension Rule (Ext)}\ \frac{C_2 \setminus \{L_2\}, M, Path \cup \{L_1\} \cup \{C_2\} \qquad C, M, Path}{C \cup \{L_1\}, M, Path},$$
$$\text{with } L_1 = \overline{L_2} \text{ and} C_2 \in M$$

$$\textit{Copy Rule (Cop)}\ \frac{C \cup \{L_1\}, M \cup \{C_2^{\mu}\}, Path}{C \cup \{L_1\}, M, Path},$$
where C_2^{μ} is a copy of $C_1, L_2 \in C_2^{\mu}, \theta(L_1) = \theta(\overline{L_2}),$ and the blocking condition holds

Figure 4 – The $\mathcal{ALCH}$ connection calculus.

In the proof procedure, we start without a specific goal or path (represented by ε) and apply the *Start rule* to some clause in M. If this results in an empty set as the goal, we close that branch with the *Axiom rule*. If a branch remains open after starting with a clause, we backtrack and choose another clause in M to apply the *Start rule*.

Different rules connect literals to the goal in different ways during the proof. The *Reduction rule* is applied when a complementary literal is already on the active path. This means that the path contains a complementary set. The *Extension rule* is triggered when another clause with a complementary literal is found. The proof is then split into two parts to check the remaining goals (without that complementary literal) and the new goal (the clause found).

A significant aspect of this method is the introduction of the *Copy rule*. This rule allows for creating a new clause from another clause in the matrix, provided that the blocking condition is satisfied. Moreover, it is essential to follow each instance of copying with an Extension or Reduction rule to ensure the termination of the calculus.

Figure 5 shows a proof for M from Example 3.6 in calculus-style.

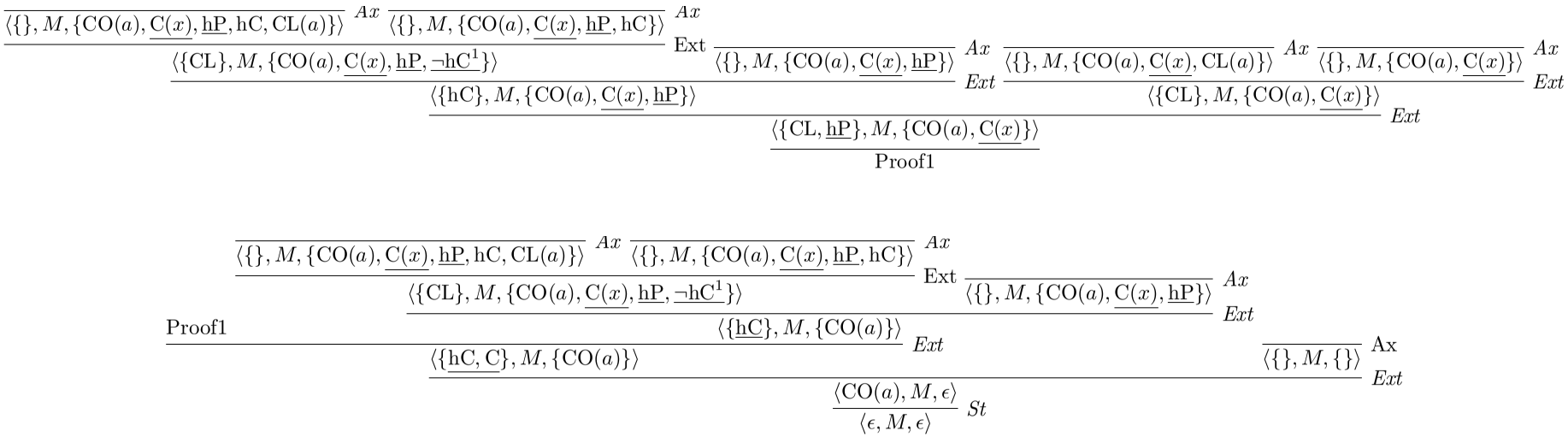


Figure 5 – Proof of matrix M from Example 3.6 in calculus-style.

3.5 CONCLUDING REMARKS

In this chapter, we revisited a connection method for description logics. In particular, we are interested in a fragment of $\mathcal{ALCHQ}^{=}\ \theta - CM$ (FREITAS; VARZINCZAK, 2018) that coincides with our target language $\mathcal{ALCH}$. The connection method demonstrates a knowledge base entails a query if their matrix representation has connections for each path of the matrix. Furthermore, we show an $\mathcal{ALCH}$ connection calculus that provides a procedure to look for connections in a matrix. It is a decision procedure and its demonstrations are available at <https://cin.ufpe.br/~fred/RR.pdf>.

4 DEFEASIBLE DESCRIPTION LOGICS

In this chapter, we revisit $\mathcal{ALCH}^{\bullet}$, an extension of description logics for non-monotonic reasoning. It extends the classical DL $\mathcal{ALCH}$ representing and reasoning for typical instances of concepts and relations. The core of this chapter draws on the paper "A Note on a Description Logic of Concept and Role Typicality for Defeasible Reasoning Over Ontologies", wrote by Ivan Varzinczak (VARZINCZAK, 2018).

The defeasible DL $\mathcal{ALCH}^{\bullet}$ (VARZINCZAK, 2018) is an extension of $\mathcal{ALCH}$ with typicality operators on complex concepts and role names. Intuitively, a typical concept expression denotes the most typical (alias normal) objects in the class, whereas a typical role expression denotes the most typical instances of the relationship represented by it.

4.1 THE LANGUAGE OF $\mathcal{ALCH}^{\bullet}$'S CONCEPTS

We assume finite sets of concept, role, and individual names, denoted, resp., with C, R, and I. With $A, B, \ldots$ we denote atomic concepts, with $r, s, \ldots$ role names, and with $a, b, \ldots$ individual names. Complex roles of $\mathcal{ALCH}^{\bullet}$ are denoted with $R, S, \ldots$ and defined by the rule:

$$R ::= \mathrm{R} \mid \bullet R$$

Complex concepts of $\mathcal{ALCH}^{\bullet}$ are denoted with $D, E, \ldots$ and are built according to the following grammar:

$$D ::= \mathrm{C} \mid \top \mid \bot \mid \neg D \mid D \sqcap D \mid D \sqcup D \mid \forall R.D \mid \exists R.D \mid \bullet D$$

Example 4.1 (Wizarding-world scenario)**.** Assume we are interested in modeling facts about the wizarding-world and its wonderful features. We have the atomic concepts $\mathrm{C} = \{\mathrm{Muggle}, \mathrm{Wizard}, \mathrm{PureBloodWizard}\}$ representing, respectively, the class of muggles, wizards, and pure blood wizards. As for the set of atomic roles, we have the set $\mathrm{R} = \{\mathrm{marriedTo}, \mathrm{hasPartner}\}$, representing, a marriage and a partnership between two people (wizards or muggles). The set of individuals I is $\{\mathrm{hermione}, \mathrm{ronWeasley}\}$. The complex concept $\bullet\mathrm{Muggle}$ denotes the set of typical (normal) muggles and $\bullet\mathrm{hasPartner}$ denotes a typical partnership relation. $\triangle$

The semantics of $\mathcal{ALCH}^{\bullet}$ extends that of classical $\mathcal{ALCH}$ and is in terms of partially-ordered structures called bi-ordered interpretations. Before introducing these, we recall a few notions.

Definition 4.1 (Strict partial order)**.** Let $<$ be a binary relation on a given set X. We say that $<$ is a *strict partial order* if it is irreflexive and transitive. ■

Definition 4.2 (Minimal elements)**.** Let $<$ be a strict partial order. If $<$ is a strict partial order on a given set X, with $\min_< X \stackrel{\text{def}}{=} \{x \in X \mid \text{there is no } y \in X \text{ s.t. } y < x\}$ we denote the *minimal elements* of X w.r.t. $<$. ■

Definition 4.3 (Well-founded)**.** Let $<$ be a strict partial order. We say $<$ is *well-founded* on a given set X if for every $\emptyset \neq X' \subseteq X$, we have $\min_< X' \neq \emptyset$. ■

Definition 4.4 (Bi-ordered interpretation)**.** An $\mathcal{ALCH}^\bullet$ *bi-ordered interpretation* is a tuple $\mathcal{O} \stackrel{\text{def}}{=} \langle \Delta^\mathcal{O}, \cdot^\mathcal{O}, <^\mathcal{O}, \ll^\mathcal{O} \rangle$ such that:

- $\langle \Delta^\mathcal{O}, \cdot^\mathcal{O} \rangle$ is a classical $\mathcal{ALCH}$ interpretation;
- $<^\mathcal{O} \subseteq \Delta^\mathcal{O} \times \Delta^\mathcal{O}$; and
- $\ll^\mathcal{O} \subseteq (\Delta^\mathcal{O} \times \Delta^\mathcal{O}) \times (\Delta^\mathcal{O} \times \Delta^\mathcal{O})$.

where both $<^\mathcal{O}$ and $\ll^\mathcal{O}$ are well-founded strict partial orders. ■

Given $\mathcal{O} = \langle \Delta^\mathcal{O}, \cdot^\mathcal{O}, <^\mathcal{O}, \ll^\mathcal{O} \rangle$, the intuition of $\Delta^\mathcal{O}$ and $\cdot^\mathcal{O}$ is the same as in a standard $\mathcal{ALCH}$ interpretation. The intuition underlying the orderings $<^\mathcal{O}$ and $\ll^\mathcal{O}$ is that they play the role of *preference relations* (or *normality* orderings): the objects (resp. pairs) that are lower down in the ordering $<^\mathcal{O}$ (resp. $\ll^\mathcal{O}$) are deemed more normal (or typical) than those higher up in $<^\mathcal{O}$ (resp. $\ll^\mathcal{O}$). Within the context of (the interpretation of) a concept C (resp. role R), $<^\mathcal{O}$ (resp. $\ll^\mathcal{O}$) therefore allows us to single out the most normal representatives falling under C (resp. R), which is the intuition of the semantics of concepts (resp. roles) of the form $\bullet C$ (resp. $\bullet R$):

Definition 4.5 (Semantics of $\mathcal{ALCH}^\bullet$)**.** A bi-ordered interpretation $\mathcal{O} = \langle \Delta^\mathcal{O}, \cdot^\mathcal{O}, <^\mathcal{O}, \ll^\mathcal{O} \rangle$

interprets $\mathcal{ALCH}^{\bullet}$ concepts in the following way:

$$\top^{\mathcal{O}} \stackrel{\text{def}}{=} \Delta^{\mathcal{O}}$$

$$\bot^{\mathcal{O}} \stackrel{\text{def}}{=} \emptyset$$

$$(D \sqcap E)^{\mathcal{O}} \stackrel{\text{def}}{=} D^{\mathcal{O}} \cap E^{\mathcal{O}}$$

$$(D \sqcup E)^{\mathcal{O}} \stackrel{\text{def}}{=} D^{\mathcal{O}} \cup E^{\mathcal{O}}$$

$$(\neg D)^{\mathcal{O}} \stackrel{\text{def}}{=} \Delta^{\mathcal{O}} \setminus D^{\mathcal{O}}$$

$$(\forall R.D)^{\mathcal{O}} \stackrel{\text{def}}{=} \{a \in \Delta^{\mathcal{O}} \mid \forall b.(a,b) \in R^{\mathcal{O}} \longrightarrow b \in D^{\mathcal{O}}\}$$

$$(\exists R.D)^{\mathcal{O}} \stackrel{\text{def}}{=} \{a \in \Delta^{\mathcal{O}} \mid \exists b.(a,b) \in R^{\mathcal{O}} \wedge b \in D^{\mathcal{O}}\}$$

$$(\bullet C)^{\mathcal{O}} \stackrel{\text{def}}{=} \min_{<^{\mathcal{O}}} C^{\mathcal{O}}$$

$$(\bullet R)^{\mathcal{O}} \stackrel{\text{def}}{=} \min_{\ll^{\mathcal{O}}} R^{\mathcal{O}}$$

■

Hence, to be a typical element of a concept (resp. role) amounts to being one of the most preferred elements in the interpretation of that concept (resp. role). It is easy to see that the typicality operators are both *idempotent*.

Example 4.2 (Semantics)**.** Let $\mathcal{O}$ be a bi-ordered interpretation such that:

- $A^{\mathcal{O}} = \{x_2, x_3\}$;
- $B^{\mathcal{O}} = \{x_1, x_2, x_3\}$;
- $r^{\mathcal{O}} = \{(x_1, x_2), (x_1, x_3)\}$;
- $<^{\mathcal{O}} = \{(x_1, x_2), (x_2, x_3)\}$;
- $\ll^{\mathcal{O}} = \{((x_1, x_2), (x_1, x_3))\}$.

Then, we have $(\bullet A)^{\mathcal{O}} = \{x_2\}$, $(\bullet B)^{\mathcal{O}} = \{x_1\}$, and $(\bullet r)^{\mathcal{O}} = \{(x_1, x_2)\}$. △

4.2 REASONING

The definitions of axiom, GCI, assertion, TBox, RBox (allowing for role subsumption axioms of the form $R \sqsubseteq S$) and ABox are as in the classical $\mathcal{ALCH}$ case. If $\mathcal{T}$, $\mathcal{R}$ and $\mathcal{A}$ are, respectively, a TBox, an RBox and an ABox, with $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ we denote henceforth a *knowledge base* (alias ontology), frequently abbreviated as KB.

Definition 4.6 (Satisfaction)**.** Let $\mathcal{O}$ be an bi-ordered interpretation; D, E be $\mathcal{ALCH}^{\bullet}$ concepts; R, S be roles; and a, b be individual names. The satisfaction relation $\Vdash$ is defined as follows:

- $\mathcal{O} \Vdash D \sqsubseteq E$ if $D^{\mathcal{O}} \subseteq E^{\mathcal{O}}$;
- $\mathcal{O} \Vdash R \sqsubseteq S$ if $R^{\mathcal{O}} \subseteq S^{\mathcal{O}}$;
- $\mathcal{O} \Vdash D(a)$ if $a^{\mathcal{O}} \in D^{\mathcal{O}}$; and
- $\mathcal{O} \Vdash R(a, b)$ if $(a^{\mathcal{O}}, b^{\mathcal{O}}) \in R^{\mathcal{O}}$.

■

Definition 4.7 (Preferentially entailment)**.** Let $\mathcal{KB}$ be an $\mathcal{ALCH}^{\bullet}$ knowledge base and α be an axiom. We say $\mathcal{KB}$ *preferentially entails* α, denoted with $\mathcal{KB} \models \alpha$, if $\mathcal{O} \Vdash \mathcal{KB}$ only if $\mathcal{O} \Vdash \alpha$, for every bi-ordered interpretation $\mathcal{O}$. ■

Example 4.3 (Wizarding-World Scenario)**.** Below we have an example of an $\mathcal{ALCH}^{\bullet}$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ for the wizarding world scenario.

$$\mathcal{T} = \left\{ \begin{array}{c} \bullet\mathrm{Muggle} \sqsubseteq \neg\mathrm{Wizard}, \\ \mathrm{PureBloodWizard} \sqsubseteq \forall \mathrm{hasPartner}.\mathrm{Wizard} \end{array} \right\}$$

$$\mathcal{R} = \left\{ \mathrm{marriedTo} \sqsubseteq \bullet\mathrm{hasPartner} \right\}$$

$$\mathcal{A} = \left\{ \begin{array}{c} \mathrm{Muggle}(\mathrm{hermione}), \\ \mathrm{PureBloodWizard}(\mathrm{ronWeasley}), \\ \mathrm{marriedTo}(\mathrm{ronWeasley}, \mathrm{hermione}) \end{array} \right\}$$

△

4.3 CONCLUDING REMARKS

In this chapter we discuss $\mathcal{ALCH}^{\bullet}$, a defeasible extension of $\mathcal{ALCH}$. The intuition is to represent most typical (normal) instances and pairs of instances in a strict partial order. As concept expression, we denote the most typical instances of a concept or a role with the typicality operator $\bullet$.

In the next chapter, we present a connection method for a defeasible extension of $\mathcal{ALC}$ (FERNANDES; FREITAS; VARZINCZAK, 2021), presented in the Proceedings of the 34th International Workshop on Description Logics (DL 2021)

Part II

Contributions

5 A CONNECTION METHOD FOR A DEFEASIBLE EXTENSION OF $\mathcal{ALCH}^{\bullet}$

This chapter introduces a method for reasoning in the description logic $\mathcal{ALCH}^{\bullet}$. The method follows a clausal approach, normalizing the input into matrix form and facilitating the search for proofs.

One of the main challenges in reasoning within $\mathcal{ALCH}^{\bullet}$ comes from the presence of the typicality operator, which applies to concepts and roles. These operators add complexity by representing both positive and negative literals internally. The proposed method addresses this challenge by providing a structured way to handle the typicality operator, allowing for reasoning within the defeasible description logic.

In that regard, (i) we use the language of $\mathcal{ALCH}$ extended with a typicality operator on concepts and another one on roles; (ii) we revisit the definition of a matrix representation of a knowledge base and establish the conditions for a given axiom to be provable from this matrix with a new normal form; (iii) we show how to handle term unification and define an adequate blocking condition in the presence of typicality operators; and (iv) we establish correctness, completeness, and termination relying only on defeasible description logic semantics.

Assume we want to model facts about the wizarding world using description logic. The following formulas represent the scenario using the logic $\mathcal{ALCH}^{\bullet}$. In this example, we use the typicality operator on concepts and roles, which adds complexity to the representation while keeping the example simple enough for clear understanding.

We aim to model that someone married to a pure-blood wizard is considered a typical wizard and that someone who typically loves another is married to that person. Additionally, we want to represent that Harry Potter typically loves Ginny Weasley, a pure-blood wizard. Below is an $\mathcal{ALCH}^{\bullet}$ knowledge base that describes this scenario in DL:

Example 5.1 (Wizarding-World Scenario (2))**.**

$$\mathcal{T} = \left\{ \exists \mathrm{marriedTo}.\mathrm{PureBloodWizard} \sqsubseteq \bullet \mathrm{Wizard} \right\}$$

$$\mathcal{R} = \left\{ \bullet \mathrm{loves} \sqsubseteq \mathrm{marriedTo} \right\}$$

$$\mathcal{A} = \left\{ \begin{array}{l} \bullet \mathrm{loves}(\mathrm{harryPotter}, \mathrm{ginWeasley}) \\ \mathrm{PureBloodWizard}(\mathrm{ginWeasley}) \end{array} \right\}$$

△

5.1 NORMAL FORM

The first step in our approach is to normalize the knowledge base. This step is crucial because it simplifies the axioms and allows them to be represented as a matrix.

Similar to previous methods for $\mathcal{ALC}$ (FREITAS; OTTEN, 2016) and $\mathcal{ALCHQ}^{=}$(FREITAS; VARZINCZAK, 2018), our normal form uses pure disjunctions and conjunctions.

Definition 5.1 (Pure disjunction and pure conjunction)**.** A *pure disjunction* ($\check{D}$) and a *pure conjunction* ($\hat{E}$) are recursively defined as follows:

$$\check{D} ::= \mathrm{C} \mid \neg\mathrm{C} \mid \check{D} \sqcup \check{D} \mid \forall \mathrm{R}.\mathrm{C} \qquad \hat{E} ::= \mathrm{C} \mid \neg\mathrm{C} \mid \hat{E} \sqcap \hat{E} \mid \exists \mathrm{R}.\mathrm{C}$$

■

However, for $\mathcal{ALCH}^{\bullet}$, we need extra attention for the typicality operators, as they introduce both positive and negative literals into the same axiom's semantics. This complexity is addressed by transforming the knowledge base into Bi-Typicality Normal Form (BTNF), where the typicality operator is limited to atomic concepts and roles.

The normalization process eliminates complex axioms, especially those with nested operators, making it easier to translate the axioms into a matrix representation.

Definition 5.2 (Bi-Typicality Normal Form)**.** Let $\hat{E}$ be a pure conjunction, $\check{D}$ be a pure disjunction, r and s be role names, A and B be atomic concepts, and a and b be individual names. A TBox, RBox or ABox axiom is in *bi-typicality normal form* (BTNF) if it is in one of the forms shown in Table 3.

Table 3 – Axioms in Bi-Typicality Normal Form (BTNF).

TBox	ABox	RBox
$\hat{E} \sqsubseteq \check{D}$	$A(a)$	$\bullet r \sqsubseteq s$
$A \sqsubseteq \exists r.B$	$\neg A(a)$	$r \sqsubseteq \bullet s$
$\forall r.A \sqsubseteq B$	$r(a,b)$	
$A \sqsubseteq \bullet B$	$\neg r(a,b)$	
$A \sqsubseteq \neg\bullet B$		
$\neg\bullet A \sqsubseteq B$		
$\bullet A \sqsubseteq B$		

A knowledge base is in BTNF if all its axioms are in BTNF. ■

The knowledge base in Example 5.1 is not in TNF once the TBox contains a non-literal concept with a typical concept and the ABox contains a role assertion with a typical role. We can obtain a semantically equivalent version by replacing the typical role with a new role name and adding a role subsumption between the new role name and $\bullet\mathrm{loves}$.

Example 5.2 (Bi-typicality normal form)**.** The following knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ is in BTNF:

$$\mathcal{T} = \left\{ \begin{array}{c} \exists \mathrm{marriedTo}.\mathrm{PureBloodWizard} \sqsubseteq N \\ N \sqsubseteq \bullet\mathrm{Wizard} \\ \bullet\mathrm{Wizard} \sqsubseteq N \end{array} \right\}$$

$$\mathcal{R} = \left\{ \begin{array}{c} \bullet\mathrm{loves} \sqsubseteq \mathrm{marriedTo} \\ N_r \sqsubseteq \bullet\mathrm{loves} \end{array} \right\}$$

$$\mathcal{A} = \left\{ \begin{array}{c} N_r(\mathrm{harryPotter}, \mathrm{ginWeasley}) \\ \mathrm{PureBloodWizard}(\mathrm{ginWeasley}) \end{array} \right\}$$

△

It is important to note that not all knowledge bases are in BTNF. Converting such bases into others in BTNF will be necessary. However, one must ensure that these entail the same as the originals.

Theorem 5.1. *Given a knowledge base $\mathcal{K}$, a decision procedure exists to obtain a BTNF of $\mathcal{K}$.*

We provide a decision procedure in Appendix A that converts any knowledge base to a semantically equivalent version in BTNF.

Dealing with complex axioms in the TBox presents a challenge when converting a knowledge base to BTNF. The normalization process can create many new axioms as the original expressions are simplified. This requires careful attention because an overly complex TBox can significantly increase the number of clauses in the matrix. Despite this, the advantages of working with a simplified, normalized base usually outweigh the challenges, especially regarding clarity and computational feasibility.

5.2 MATRIX CHARACTERISATION

DL reasoning methods generally do not use variables in their proof structures. Instead, such methods build models, adopting the instantiation of individuals for TBox axioms or creating new individuals when applying the existential quantifier, for example.

In our case, we followed the same approach for first-order logic and used variables. Similar to Otten's comparison in the modal case (OTTEN, 2022), approaches that do not use variables need to guess the correct sequence of rule application and individual (constant) instantiation. The use of variables, in this case, is helpful as it allows us to delay the choice of which individual to instantiate until the method can fully prove a branch, thereby enhancing the effectiveness of our method.

The vocabulary to build matrices contains a set $\mathtt{V}$ for variables; a set $\mathtt{A}$ for new individuals; and the previously defined sets C, R, and I. With $\mathtt{T} = \mathtt{V} \cup \mathtt{A} \cup \mathrm{I}$ we represent the set of *terms*. Besides, we represent the relation between terms and pairs of terms with $<$ and $\ll$, respectively. They are the syntactic counterparts of the preference relations on objects and pairs of objects in the semantics.

Definition 5.3 (Literal)**.** A *literal* in $\mathcal{ALCH}^\bullet$ is defined as follows:

$$L ::= \mathrm{C}(\mathtt{T}) \mid \mathrm{R}(\mathtt{T},\mathtt{T}) \mid \mathtt{T} < \mathtt{T} \mid (\mathtt{T},\mathtt{T}) \ll (\mathtt{T},\mathtt{T}) \mid \neg L$$

■

Example 5.3 (Literal)**.** Given $\mathtt{V} = \{x\}$, $\mathtt{A} = \{a'\}$, $\mathtt{I} = \{\mathrm{ginWeasley}\}$, $\mathrm{R} = \{\mathrm{loves}\}$, and $\mathrm{C} = \{\mathrm{Wizard},\ \mathrm{PureBloodWizard}\}$, the following cases are literals in $\mathcal{ALCH}^\bullet$:

- $\mathrm{Wizard}(\mathrm{ginWeasley})$
- $\neg\mathrm{loves}(x, a')$
- $(\mathrm{ginWeasley}, \mathrm{ginWeasley}) \ll (x, a')$
- $\neg(\mathrm{ginWeasley} < x)$

△

As usual, we define a clause and a matrix in the $\mathcal{ALCH}^\bullet$ case.

Definition 5.4 (Clause)**.** A *clause* is a set of literals. ■

Definition 5.5 (Matrix)**.** A *matrix* is a set of clauses. ■

The matrix representation of TBox axioms in BTNF is shown in Table 4. Analogously, the matrix representation of RBox axioms and ABox assertions in BTNF are shown in Table 5.

Table 4 – Matrix representation of TBox axioms.

Axiom (α)	Visual representation ($M(\alpha)$)
$\hat{E} \sqsubseteq \check{D}$	$\begin{bmatrix} E_1(x) \\ \cdots \\ E_n(x) \\ \neg D_1(x) \\ \cdots \\ \neg D_m(x) \end{bmatrix}$
$A \sqsubseteq \exists r.B$	$\begin{bmatrix} A(x) & A(x) \\ \neg r(x, a_x) & \neg B(a_x) \end{bmatrix}$
$\forall r.A \sqsubseteq B$	$\begin{bmatrix} \neg r(x, a_x) & A(a_x) \\ \neg B(x) & \neg B(x) \end{bmatrix}$
$A \sqsubseteq \neg\bullet B$	$\begin{bmatrix} A(x) & A(x) & A(x) \\ B(x) & B(x) & B(x) \\ \neg(a_x < x) & \neg B(a_x) & B(y) \\ & & (y < a_x) \end{bmatrix}$
$\bullet A \sqsubseteq B$	$\begin{bmatrix} A(x) & A(x) & A(x) \\ \neg(a_x < x) & \neg A(a_x) & A(y) \\ \neg B(x) & \neg B(x) & (y < a_x) \\ & & \neg B(x) \end{bmatrix}$
$A \sqsubseteq \bullet B$	$\begin{bmatrix} A(x) & A(x) \\ \neg B(x) & (y < x) \\ & B(y) \end{bmatrix}$
$\neg\bullet A \sqsubseteq B$	$\begin{bmatrix} \neg A(x) & A(y) \\ \neg B(x) & \neg(y < x) \\ & \neg B(x) \end{bmatrix}$

Table 5 – Matrix representation for RBox and ABox axioms.

Axiom	Visual representation
$\bullet r \sqsubseteq s$	$\begin{bmatrix} r(x,y) & r(x,y) & r(x,y) \\ \neg((a_x,b_y) \ll (x,y)) & \neg r(a_x,b_y) & r(z,k) \\ \neg s(x,y) & \neg s(x,y) & (z,k) \ll (a_x,b_y) \\ & & \neg s(x,y) \end{bmatrix}$
$r \sqsubseteq \bullet s$	$\begin{bmatrix} r(x,y) & r(x,y) \\ \neg s(x,y) & s(z,k) \\ & (z,k) \ll (x,y) \end{bmatrix}$
$A(a)$	$\begin{bmatrix} \neg A(a) \end{bmatrix}$
$\neg A(a)$	$\begin{bmatrix} A(a) \end{bmatrix}$
$r(a,b)$	$\begin{bmatrix} \neg r(a,b) \end{bmatrix}$
$\neg r(a,b)$	$\begin{bmatrix} r(a,b) \end{bmatrix}$

Please note that in our matrix representation, we do not use underline as in the case of the DL $\mathcal{ALCH}$. In Chapter 3, underlines were necessary to indicate which terms of a clause should be considered during a connection. However, such information is not required here since the terms are explicitly presented.

Example 5.4 (Matrix representation of a TBox)**.** Given the TBox in BTNF from Example 5.2 and following Table 4, the matrix representation[1] M is

$$\begin{bmatrix} \mathrm{mT}(x_0,x_1) & N(x_2) & N(x_3) & \neg N(x_5) & \neg N(x_6) \\ \mathrm{PBW}(x_1) & \neg \mathrm{W}(x_2) & \mathrm{W}(x_4) & \mathrm{W}(x_5) & \mathrm{W}(x_6) \\ \neg N(x_0) & & (x_4 < x_3) & \neg(c_{x_5} < x_5) & \neg \mathrm{W}(c_{x_6}) \end{bmatrix}$$

△

[1] We abbreviate the concept and role names for clarity in the example.

The matrix is a structural representation of the knowledge base. In that sense, there is no relationship between the literals and the $\mathcal{ALCH}^{\bullet}$ semantics.

Definition 5.6 (Path)**.** Given a matrix M, a *path* is a set containing exactly one literal from each clause of M. ■

We recall that the order of literals does not matter in a path because it is a set.

Definition 5.7 (Term substitution)**.** A *term substitution* is a function $\sigma : \mathtt{V} \longrightarrow \mathtt{A} \cup \mathrm{I}$ that maps variables to (possibly new) individuals. We extend it to literals, clauses and matrices as follows:

1. $\sigma(A(t)) = A(\sigma(t))$, for all $A \in \mathtt{C}$
2. $\sigma(r(t,u)) = r(\sigma(t), \sigma(u))$, for all $r \in \mathtt{R}$
3. $\sigma(t < u) = \sigma(t) < \sigma(u)$
4. $\sigma((t,u) \ll (v,k)) = (\sigma(t), \sigma(u)) \ll (\sigma(v), \sigma(k))$
5. $\sigma(\neg L) = \neg(\sigma(L))$, for all literal L
6. $\sigma(\{L_1, \cdots, L_n\}) = \{\sigma(L_1), \cdots, \sigma(L_n)\}$, for all $1 \leq i \leq n$ and for all literal L_i
7. $\sigma(\{C_1, \cdots, C_m\}) = \{\sigma(C_1), \cdots, \sigma(C_m)\}$, for all $1 \leq j \leq m$ and for all clause C_j

■

Example 5.5. Given the set of variables $\mathtt{V} = \{x, y, z\}$, the set of individuals $\mathrm{I} = \{\mathrm{harryPotter}, \mathrm{ginWeasley}\}$, and $\mathtt{A} = \{c, d\}$. We define a substitution σ such that:

$$\sigma = \{x \setminus \mathrm{harryPotter}, y \setminus \mathrm{ginWeasley}, z \setminus c\}$$

Applying σ to different elements, we have that:

- $\sigma(\mathrm{Wizard}(x)) = \mathrm{Wizard}(\sigma(x)) = \mathrm{Wizard}(\mathrm{harryPotter})$
- $\sigma(\mathrm{loves}(x,y)) = \mathrm{loves}(\sigma(x), \sigma(y)) = \mathrm{loves}(\mathrm{harryPotter}, \mathrm{ginWeasley})$
- $\sigma(x < y) = \sigma(x) < \sigma(y) = \mathrm{harryPotter} < \mathrm{ginWeasley}$
- $\sigma((x,y) \ll (z,x)) = (\sigma(x), \sigma(y)) \ll (\sigma(z), \sigma(x)) = (\mathrm{harryPotter}, \mathrm{ginWeasley}) \ll (c, \mathrm{harryPotter})$

- $\sigma(\neg\text{Wizard}(x)) = \neg(\sigma(\text{Wizard}(x))) = \neg\text{Wizard}(\text{harryPotter})$
- $\sigma(\{\text{Wizard}(x), \neg\text{PureBloodWizard}(y), \text{loves}(y, z)\}) =$
  $\{\text{Wizard}(\text{harryPotter}), \neg\text{PureBloodWizard}(\text{ginWeasley}), \text{loves}(\text{ginWeasley}, c)\}$
- $\sigma(\{\{\text{Wizard}(x), \text{PureBloodWizard}(y)\}, \{\text{loves}(x, z), \neg\text{Wizard}(z)\}\}) =$
  $\{\{\text{Wizard}(\text{harryPotter}), \text{PureBloodWizard}(\text{ginWeasley})\},$
  $\{\text{loves}(\text{harryPotter}, c), \neg\text{PureBloodWizard}(c)\}\}$

△

We establish the bond between matrices and semantics by associating the paths of the matrix with the bi-ordered interpretations, thus defining the notion of agreement.

Definition 5.8 (Path agreement to an interpretation)**.** Let t, u, v, and z be terms; $\mathcal{I}$ be a bi-ordered interpretation, σ be a term substitution, and p be a path. We say that $\mathcal{I}$ *agrees* with p w.r.t. σ, denoted as $\mathcal{I} \approx_\sigma p$, if:

1. $\sigma(t)^\mathcal{I} \in A^\mathcal{I}$ if $\neg A(t) \in p$;
2. $(\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}) \in r^\mathcal{I}$ if $\neg r(t, u) \in p$;
3. $(\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}) \in <^\mathcal{I}$ if $\neg(t < u) \in p$;
4. $((\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}), (\sigma(v)^\mathcal{I}, \sigma(z)^\mathcal{I})) \in \ll^\mathcal{I}$ if $\neg((t, u) \ll (v, z)) \in p$;
5. $\sigma(t)^\mathcal{I} \in (\overline{A})^\mathcal{I}$ if $A(t) \in p$;
6. $(\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}) \in (\overline{r})^\mathcal{I}$ if $r(t, u) \in p$;
7. $(\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}) \in (\overline{<})^\mathcal{I}$ if $(t < u) \in p$; and
8. $((\sigma(t)^\mathcal{I}, \sigma(u)^\mathcal{I}), (\sigma(v)^\mathcal{I}, \sigma(z)^\mathcal{I})) \in (\overline{\ll})^\mathcal{I}$ if $((t, u) \ll (v, z)) \in p$.

■

Definition 5.9 (Matrix agreement to an interpretation)**.** Given a bi-ordered interpretation $\mathcal{I}$ and a matrix M. We say that $\mathcal{I}$ *agrees* with M, or just $\mathcal{I} \approx M$ if, for each term substitution σ, a path p in M exists, such that $\mathcal{I} \approx_\sigma p$. ■

Now, we have a bond between matrices and $\mathcal{ALCH}^\bullet$ semantics in the sense that

Lemma 5.1. *Given two matrices M_1, M_2 and a bi-ordered interpretation $\mathcal{I}$, then $\mathcal{I} \approx M_1$ and $\mathcal{I} \approx M_2$ iff $\mathcal{I} \approx M_1 \cup M_2$.*

Proof. We prove this lemma by showing its contrapositive.

$\Rightarrow$. Assume that $\mathcal{I} \not\approx M_1 \cup M_2$. Then, exists a term substitution σ such that $\mathcal{I} \not\approx_\sigma p$, for all path p in $M_1 \cup M_2$. Besides, a path of $M_1 \cup M_2$ is the union of a path of each matrix M_1 and M_2, and σ is a term substitution for M_1 and M_2. Hence, there exists a term substitution σ that either $\mathcal{I} \not\approx_\sigma M_1$ or $\mathcal{I} \not\approx_\sigma M_2$.

$\Leftarrow$. It is analogous to the other part and left to the reader. □

Theorem 5.2. *Given a bi-ordered interpretation $\mathcal{I}$ and a BTNF axiom α, $\mathcal{I} \Vdash \alpha$ iff $\mathcal{I} \approx M(\alpha)$.*

Proof. The proof of this theorem comes from the close relationship between the paths of $M(\alpha)$ and the semantics of α. □

Example 5.6. To illustrate the result of Theorem 5.2, consider the axiom $\mathrm{PureBloodWizard} \sqsubseteq \bullet\mathrm{Wizard}$. Its matrix representation is:

$$\begin{bmatrix} \mathrm{PureBloodWizard}(x) & \mathrm{PureBloodWizard}(x) \\ \neg\mathsf{Wizard}(x) & \mathsf{Wizard}(y) \\ & (y < x) \end{bmatrix}$$

So, an interpretation $\mathcal{I}$ satisfies $\mathrm{PureBloodWizard} \sqsubseteq \bullet\mathrm{Wizard}$ iff $\mathrm{PureBloodWizard}^{\mathcal{I}} \subseteq min_{<^{\mathcal{I}}}(\mathrm{Wizard}^{\mathcal{I}})$. As such, for each element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ either:

1. $a^{\mathcal{I}}$ is an element of $(\neg\mathrm{PureBloodWizard})^{\mathcal{I}}$;
2. $a^{\mathcal{I}}$ is an element of $\mathrm{Wizard}^{\mathcal{I}}$ and there is no $b^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ s.t. $b^{\mathcal{I}} \in \mathrm{Wizard}^{\mathcal{I}}$ and $b^{\mathcal{I}} <^{\mathcal{I}} a^{\mathcal{I}}$

The paths containing $\mathrm{PureBloodWizard}(x)$ will agree with $\mathcal{I}$ for the first condition elements. To the other elements that hold the second condition, the paths containing $\neg\mathrm{Wizard}(x)$ and either $B(y)$ or $(y < x)$ will agree with $\mathcal{I}$.

△

Theorem 5.2 states a correspondence between matrices and their former axioms. However, TBox and Rbox axioms, by the semantics of $\mathcal{ALCH}^\bullet$, can be reused for other individuals besides the unified until this point. So, instead of using clauses once, we copy them to use again in further connections.

Definition 5.10 (Multiplicity)**.** The *multiplicity* is a function μ that maps for each clause of a matrix a natural number denoting the number of copies of such clause. ■

Definition 5.11 (Copy)**.** Given two clauses C_1 and C_2, we say that C_2 is a *copy* of C_1 if they have the same literals but new variables. ■

With M^μ we define the matrix M and its clause copies. Variables and new individual names are replaced by non-used variables and new individual names during the copy.

Definition 5.12 (i-th copy)**.** Given a matrix M, a clause $C \in M$, and a multiplicity μ, with C^i we denote the *i-th copy* of $C \in M^\mu$, where $1 \leq i \leq \mu(C)$. ■

Example 5.7 (Copy)**.** Let x and y be variables, a be an individual name, and M be the matrix as follows:

$$\begin{bmatrix} A(x) & B(y) \\ \neg r(x,a) & \\ \neg B(a) & \end{bmatrix}$$

Then, the clauses $\{A(x'), \neg r(x',a), \neg B(a)\}$ and $\{B(y')\}$ are *copies* of the first and second clause of M, where x' and y' are copies of x and y, respectively. Besides, for this example, we have that

- $\mu(\{A(x), \neg r(x,a), \neg B(a)\}) \stackrel{\text{def}}{=} 1$; and
- $\mu(\{B(y)\}) \stackrel{\text{def}}{=} 1$.

Finally, $\{B(y')\}$ is the *first copy* of $\{B(y)\}$ and M^μ is defined as

$$\begin{bmatrix} A(x) & B(y) & A(x') & B(y') \\ \neg r(x,a) & & \neg r(x',a) & \\ \neg B(a) & & \neg B(a) & \end{bmatrix}$$

△

Lemma 5.2. *Given a bi-ordered interpretation $\mathcal{I}$ and a BTNF axiom α, then $\mathcal{I} \approx M(\alpha)$ iff for each multiplicity μ, $\mathcal{I} \approx M(\alpha)^\mu$.*

Proof. We prove this lemma by showing its contrapositive.

$\Rightarrow$. Assume there exists a multiplicity μ such that $\mathcal{I} \not\approx M(\alpha)^{\mu}$. Then, also exists a term substitution σ that $\mathcal{I} \not\approx_{\sigma} p$, for all path p in $M(\alpha)^{\mu}$. By Lemma 5.1 and the fact that $M(\alpha)^{\mu} = \bigcup_{\mu}^{i=1} M(\alpha)^{i}$, at least one copy $M(\alpha)^{j}$ must not agree with $\mathcal{I}$. Hence, a term substitution σ' that coincides with the substitutions of $M(\alpha)^{j}$ not agree with $\mathcal{I}$. Therefore, $\mathcal{I} \not\approx M(\alpha)$.

$\Leftarrow$. Assume a term substitution σ exists such that $\mathcal{I} \not\approx_{\sigma} p$, for all path p in M. Hence, a term substitution σ' that contains the substitutions of σ cannot agree with $\mathcal{I}$ for some multiplicity μ. To illustrate this fact, consider a term substitution for a given individual a and a variable x that $\mathcal{I} \not\approx M(\alpha)$. Every term substitution that replaces x or its copies with a cannot agree with $\mathcal{I}$; otherwise, $\mathcal{I} \approx M(\alpha)$. Therefore, exists a multiplicity μ such that $\mathcal{I} \not\approx M(\alpha)^{\mu}$. □

Definition 5.13 (σ-complementary)**.** Given a literal L, a path p and a term substitution σ, we say that p is σ-*complementary* if $\{\sigma(L), \sigma(\neg L)\} \in p$. ■

We say that a matrix M is σ-complementary if every path in M is σ-complementary.

Example 5.8 (σ-complementary path)**.** Let M be the matrix as follows:

$$\begin{bmatrix} A(x) & B(y) \\ \neg r(x,a) & \\ \neg B(a) & \end{bmatrix}$$

If we assume σ as $\{y \setminus a\}$, then the path $\{\neg B(a), B(y)\}$ is σ-complementary, once $\sigma(B(y)) = B(a)$. Otherwise, assuming $\sigma \stackrel{\text{def}}{=} \{y \setminus b\}$, then the path $\{\neg B(a), B(y)\}$ is not σ-complementary, once $\sigma(B(y)) = B(b)$. △

Theorem 5.3 (Matrix characterisation)**.** *Given a knowledge base $\mathcal{K}$, $\mathcal{K} \models \bot$ iff there exists a multiplicity μ and a term substitution σ such that $M(\mathcal{K})^{\mu}$ is σ-complementary.*

Proof. We prove this theorem by showing its contrapositive.

$\Rightarrow$. Assume there is no multiplicity μ and term substitution σ such that $M(\mathcal{K})^{\mu}$ is σ-complementary. Then, for all multiplicity μ and all term substitution σ, a path p exists in the matrix with no complementary set. As such, p must agree with some interpretation $\mathcal{I}$. By Lemma 5.1, Theorem 5.2, and Lemma 5.2, $\mathcal{I} \approx M(\alpha)$, for all $\alpha \in \mathcal{K}$. Hence, $\mathcal{K} \not\models \bot$.

$\Leftarrow$. Assume $\mathcal{K} \not\models \bot$. Then, a bi-ordered interpretation $\mathcal{I}$ exists, that $\mathcal{I} \Vdash \mathcal{K}$. By Theorem 5.2, Lemma 5.1 and Lemma 5.2, for each multiplicity μ and each term substitution σ, a path p exists in $M(\mathcal{K})^{\mu}$ that agrees with $\mathcal{I}$. Hence, no complementary set exists in p. □

5.3 BLOCKING

Blocking is a mechanism that interrupts the search for proof when cycles are detected within that proof branch. This feature helps to prevent repetitive searches and avoids unnecessary iterations, which could otherwise result in an infinite loop.

The blocking notion is inspired by the one proposed in the $\mathcal{ALCH}^{\bullet}$ tableaux (VARZINCZAK, 2018), where a term is blocked by the ABox when some conditions are met. Here, instead of being blocked by the ABox, the term is blocked by the active path of the proof.

Definition 5.14 (r-descendant)**.** Given a path p, a term substitution σ, a role name r or its negation, and terms t, u; t is a *r-predecessor* of u, or u is a *r-successor* of t w.r.t. p and σ, if $r(\sigma(t), \sigma(u)) \in p$. The transitive closure of *r-predecessor* and *r-successor* are called *r-ancestor* and *r-descendant*, respectively. ■

Example 5.9 (r-descendant)**.** Let x y, a, b, and c be terms; σ be defined as $\{x \setminus a, y \setminus b\}$; r be a role name; and $\{r(x, y), r(y, c)\}$ be the (active) path. Then, once $\sigma(x) = a$ and $\sigma(y) = b$:

- a is a r-ancestor of b and c;
- b is a r-successor of a and is a r-predecessor of c; and
- c is a r-descendant of b and a.

△

Definition 5.15 ($<$-descendant)**.** Given a path p, a term substitution σ, $<$ or its negation, and terms t, u; t is a $<$-*predecessor* of u, or u is a $<$-*successor* of t w.r.t. p and σ, if $< (\sigma(t), \sigma(u)) \in p$. The transitive closure of $<$-*predecessor* and $<$-*successor* are called $<$-*ancestor* and $<$-*descendant*, respectively. ■

Example 5.10 ($<$-descendant)**.** Let x y, a, b, and c be terms; σ be defined as $\{x \setminus a, y \setminus b\}$; and $\{x < y, y < c)\}$ be the (active) path. Then, once $\sigma(x) = a$ and $\sigma(y) = b$:

- a is a $<$-ancestor of b and c;

- b is a $<$-successor of a and is a $<$-predecessor of c; and
- c is a $<$-descendant of b, and a.

△

Definition 5.16 (Set of concepts)**.** Given a term t, a path p, and a term substitution σ, the *set of concepts* of t w.r.t. the path p, denoted by $\tau_p^\sigma(t)$ is

$$\tau_p^\sigma(t) = \{D \mid \text{for all } D(\sigma(t)) \in \sigma(p)\}$$

■

Definition 5.17 (Blocked term)**.** Given two terms t, u, a path p and a term substitution σ, we say that t is *blocked* by u w.r.t. p and σ:

1. if u is a r-ancestor or $<$-descendant of t, and
2. $\tau_p^\sigma(t) \subseteq \tau_p^\sigma(u)$.

■

Example 5.11 (Blocked term)**.** Let $\mathrm{harryPotter}$ (h), $\mathrm{ginWeasley}$ (g) and $\mathrm{ronWeasley}$ (r) be terms; Wizard (W) and $\mathrm{PureBloodWizard}$ (PBW) be concept names; loves (l) be a role name; and $p = \{\mathrm{l(h,g)}, \mathrm{h} < \mathrm{r}, \mathrm{W(h)}, \mathrm{W(g)}, \mathrm{PBW(r)}\}$ be the (active) path. Then, g is blocked by h, once h is a l-ancestor of g and

$$\tau_p^\sigma(\mathrm{ginWeasley}) = \{\mathrm{Wizard}\} \subseteq \{\mathrm{Wizard}\} = \tau_p^\sigma(\mathrm{harryPotter}).$$

However, $\mathrm{ronWeasley}$ is not blocked by $\mathrm{harryPotter}$, once

$$\tau_p^\sigma(\mathrm{ronWeasley}) = \{\mathrm{PureBloodWizard}\} \not\subseteq \{\mathrm{Wizard}\} = \tau_p^\sigma(\mathrm{harryPotter}).$$

△

When blocking a term, we consider its related concepts, its position within the roles, and its place in the typicality hierarchy. We focus on these cases because they are where the connection method can create new individuals.

The underlying principle is that if a term is an r-descendant of another term and possesses the same concepts (or even fewer) than its predecessor, the proof path has already confirmed the potential connections the term could establish. In this scenario, we would be in a loop.

This same principle applies to the typicality hierarchy. Suppose a term is a $<$-ancestor of another term, and its concepts are at most the same as those of its successor. In that case, duplicate instances already exist in that branch of the proof, and their potential connections have already been verified.

The definition of blocking aligns with the principles proposed by Varzinczak for the tableau method in $\mathcal{ALCH}^{\bullet}$. However, the connection method incorporates additional concepts, such as copying clauses during the search process. Hence, we introduce next the concepts of literal and clause blocking.

Definition 5.18 (Blocked literal)**.** Given a literal L, a path p and a term substitution σ, we say that L is *blocked* w.r.t. p and σ if:

1. $\sigma(L) \in \sigma(p)$; or
2. $L \in C^n$ and there exists a copied new individual t_n in L, such that either t_{n-i} is blocked or $\tau^{\sigma}_{p\cup\{L\}}(t_n) \subseteq \tau^{\sigma}_{p\cup\{L\}}(t_{n-i})$, for some clause $C \in M^{\mu}$, some $2 < n \leq \mu(C)$, and some $1 \leq i < n$.

■

Definition 5.19 (Blocked clause)**.** We say that a *clause is blocked* w.r.t. a path p and a term substitution σ if some literal $L \in C$ is blocked w.r.t. p. ■

5.4 CONNECTION CALCULUS

We define the connection method using a formal calculus, as illustrated in Figure 6. The core structure consists of a triple $\langle C, M, Path\rangle$, where C represents the goal (the set of literals), M is the matrix, and *Path* denotes the active path of the proof, which indicates the current line of reasoning being sought.

The proof begins without a defined goal or path (represented by ε) by applying the *Start rule* to a clause in M. If the proof arrives at an empty set for the goal, that branch is closed using the *Axiom rule*. However, backtracking occurs if a branch remains open for a particular starting clause, prompting the proof search to select another clause from M to apply the *Start rule* again.

Calculus connects literals to the goal differently depending on the rule applied during the proof search. The first rule is called the *Reduction Rule*. This rule is used when a complemen-

$$\textbf{\textit{Axiom (Ax)}}\ \frac{}{\{\}, M, Path}$$

$$\textbf{\textit{Start rule (St)}}\ \frac{C_1, M, \{\}}{\varepsilon, M, \varepsilon},\ \text{with } C_1 \text{ being a blocking-free copy of some } C \in M$$

$$\textbf{\textit{Reduction rule (Red)}}\ \frac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}, \text{with } \sigma(L_1) = \sigma(\overline{L_2})$$

$$\textbf{\textit{Extension rule (Ext)}}\ \frac{C_1 \setminus \{L_2\}, M, Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path},\ \text{with } \sigma(L_1) = \sigma(\overline{L_2})$$
$$\text{and } L_2 \in C_1, C_1 \text{ being a blocking-free copy of some } C_2 \in M$$

Figure 6 – The calculus.

tary literal is already present on the active path, indicating that the path contains a set of complementary literals.

The second rule is known as the *Extension Rule*. This rule is activated when another clause containing a complementary literal is discovered. When this happens, the proof is divided into two parts: one part checks the remaining goals without considering the complementary literal, while the other part addresses the new goal from the clause that was found.

It is important to note that every calculus rule application must ensure the goal is not blocked by its path. If the goal is blocked, then no rule application should take place.

Definition 5.20 (Proof tree)**.** A proof tree is the tree representation of the applications of the calculus rules to a given matrix. ■

Definition 5.21 (Connection proof)**.** Given a derivation $Proof$ of a triple $\langle C, M, \mathit{Path}\rangle$, we say that $Proof$ is a *connection proof for* $\langle C, M, \mathit{Path}\rangle$, if there exists a multiplicity μ and a substitution σ such that every leaf of $Proof$ ends with an *Axiom*. ■

Example 5.12 (Proof tree)**.** Let M be a matrix as follows:

$$\begin{bmatrix} N(a) & \neg N(x_5) \\ & \mathrm{W}(x_5) \\ & \neg(c_{x_5} < x_5) \\ & \end{bmatrix}$$

A proof tree of the matrix M is shown in Figure 7. However, the proof tree of the Figure has a non-Axiom leaf. Hence, it is not a connection proof.

$$\dfrac{\dfrac{\langle\{W(x_5), \neg(c_{x_5} < x_5)\}, M, \{\}\rangle \; \overline{\langle\{\}, M, \{\}\rangle}\ \text{Ax}}{\langle\{N(a)\}, M, \{\}\rangle}\ \text{Ext}}{\langle\varepsilon, M, \varepsilon\rangle}\ St$$

Figure 7 – Example of a proof tree that is not a connection proof

△

The next step is establishing a link between the presented calculus and the matrix's characterisation. This will allow us to demonstrate that deriving a connection proof based on the calculus for a given matrix is equivalent to proving that the matrix is valid. However, to simplify this demonstration, we must first introduce the concept of relative paths.

Definition 5.22 (Relative paths)**.** Given two sets of literals C and S, and a matrix M, we define the *relative paths* of C, S and M as:

$$\varphi(M) \stackrel{\text{def}}{=} \{p \mid p \text{ is a path of } M\}$$
$$\varphi(M, S) \stackrel{\text{def}}{=} \{p \in \varphi(M) \mid S \subseteq p\}$$
$$\varphi(M, S, C) \stackrel{\text{def}}{=} \{p \in \varphi(M, S) \mid L \in p, \text{for some } L \in C\}$$

■

Lemma 5.3. *Given a triple $\langle C, M, \mathit{Path}\rangle$, if it has a connection proof, for some term substitution σ, then there exists a multiplicity μ for all path $p \in \varphi(M, \mathit{Path}, C)$, s.t. p is σ-complementary.*

Proof. We prove the lemma above by structural induction over the connection proofs. Since a connection proof can be a subtree of another connection proof, we can assume, as the Induction Hypothesis (IH), that if there exists a connection proof of a subtree for some σ, then there exists a multiplicity μ such that every path in it is σ-complementary. We demonstrate that the lemma holds for Axiom and the rules Reduction and Extension in the calculus as follows:

- **Axiom (Ax) :** Assume that $\overline{\{\}, M, \mathit{Path}}$ is a connection proof for $\langle\{\}, M, \mathit{Path}\rangle$. Therefore, $\varphi(M, \mathit{Path}, \emptyset) = \emptyset$, meaning that IH holds since there is no complementary path in the empty set;
- **Reduction (Red) :** Assume that $\dfrac{Proof}{C, M, \mathit{Path} \cup \{L_2\}}$ is a connection proof for $\langle C, M, \mathit{Path} \cup \{L_2\}\rangle$, for some term substitution σ. Then, the derivation $\dfrac{\dfrac{Proof}{C, M, \mathit{Path} \cup \{L_2\}}}{C \cup \{L_1\}, M, \mathit{Path} \cup \{L_2\}}$

is a connection proof, where $\sigma'(\sigma(L_1)) = \sigma'(\sigma(\overline{L_2}))$. By the IH, for some multiplicity μ, every path $p' \in \varphi(M^{\mu}, \mathit{Path} \cup \{L_2\}, C)$ is σ-complementary. Furthermore, every path $p'' \in \varphi(M^{\mu}, \mathit{Path} \cup \{L_2\}, \{L_1\})$ is σ'-complementary, since $\sigma'(\sigma(L_1)) = \sigma'(\sigma(\overline{L_2}))$. Let μ' be μ and σ'' be the composition of σ and σ'. As $\varphi(M^{\mu'}, \mathit{Path} \cup \{L_2\}, C \cup \{L_1\}) = \varphi(M^{\mu'}, \mathit{Path} \cup \{L_2\}, C) \cup \varphi(M^{\mu'}, \mathit{Path} \cup \{L_2\}, \{L_1\})$, we conclude that every path in $\varphi(M^{\mu'}, \mathit{Path} \cup \{L_2\}, C \cup \{L_1\})$ is σ''complementary;

- **Extension (Ex) :** Assume that $\dfrac{\mathit{Proof1}}{C_2 \setminus \{L_2\}, M, \mathit{Path} \cup \{L_1\}}$ is a connection proof for $\langle C_2 \setminus \{L_2\}, M, \mathit{Path} \cup \{L_1\}\rangle$, and $\dfrac{\mathit{Proof2}}{C, M, \mathit{Path}}$ is a connection proof for $\langle C, M, \mathit{Path}\rangle$, for some substitution σ. By the IH, there exists a multiplicity μ_1 such that every path in $\varphi(M^{\mu_1}, \mathit{Path} \cup \{L_1\}, C_2 \setminus \{L_2\})$ is σ-complementary, and there exists a multiplicity μ_2 such that every path in $\varphi(M^{\mu_2}, \mathit{Path}, C)$ is σ-complementary. Then, the derivation

$$\dfrac{\dfrac{\text{Proof1}}{C_2 \setminus \{L_2\}, M, \mathit{Path} \cup \{L_1\}} \quad \dfrac{\text{Proof2}}{C, M, \mathit{Path}}}{C \cup \{L_1\}, M, \mathit{Path}}$$

  with $\sigma'(\sigma(L_1)) = \sigma'(\sigma(\overline{L_2}))$ and C_2 as a copy of some clause $C_1 \in M$, is a connection proof for $\langle C \cup \{L_1\}, M, \mathit{Path}\rangle$. Let μ' be the combination of μ_1 and μ_2, and σ'' be the composition of σ and σ'. Hence, every path $p' \in \varphi(M^{\mu'}, \mathit{Path}, C)$ is σ''-complementary and every path $p'' \in \varphi(M^{\mu'}, \mathit{Path}, C_2 \setminus \{L_2\})$ is also σ''complementary. Moreover, once $\sigma''(L_1) = \sigma''(\overline{L_2})$, every path in $\varphi(M^{\mu'}, \mathit{Path} \cup \{L_1\}, \{L_2\})$ is complementary. As $C_2 \in M^{\mu'}$, we also have that $\varphi(M^{\mu'}, \mathit{Path} \cup \{L_1\}, C_2) = \varphi(M, \mathit{Path} \cup \{L_1\})$ and every path on it is complementary. Therefore, every path in $\varphi(M^{\mu'}, \mathit{Path}, C \cup \{L_1\})$ is σ''-complementary since $\varphi(M^{\mu'}, \mathit{Path}, C \cup \{L_1\}) = \varphi(M^{\mu'}, \mathit{Path} \cup \{L_1\}) \cup \varphi(M, \mathit{Path}, C)$.

□

Theorem 5.4 (Soundness of the calculus)**.** *Given a matrix M, if $\langle \varepsilon, M, \varepsilon\rangle$ has a connection proof with σ, then there exists a multiplicity μ such that every path in $\varphi(M^{\mu})$ is σ-complementary.*

Proof. We prove this theorem by contrapositive. If we assume there is no multiplicity, then no copies can occur, and the triple has no connection proof. Now, we assume there exists a multiplicity μ and a path in $\varphi(M^{\mu})$ that is not σ-complementary, for all σ. Let $\dfrac{\dfrac{\cdots}{\langle C_2, M, \{\}\rangle}}{\langle \varepsilon, M, \varepsilon\rangle}\,\mathit{St}$ be the derivation for M, with C_2 being a copy of some clause $C_1 \in M$. Then, $\varphi(M^{\mu}, \{\}, C_2)$,

is not σ-complementary as well. By Lemma 5.3's contrapositive, there is no connection proof for $\langle C_2, M, \{\}\rangle$, therefore, there is no connection proof for $\langle \varepsilon, M, \varepsilon\rangle$. □

Theorem 5.5 (Completeness of the calculus)**.** *Given a matrix M, if there exists a multiplicity μ and term substitution σ such that every path in $\varphi(M^\mu)$ is σ-complementary, then $\langle \varepsilon, M, \varepsilon\rangle$ has a connection proof.*

Proof. We also prove this theorem by the contrapositive. If there is no connection proof for $\langle \epsilon, M, \epsilon\rangle$, then there exists no multiplicity μ and term substitution σ such that every path in $\varphi(M^\mu)$ is σ-complementary. Thus, w.l.o.g. we assume there exists a saturated derivation branch of $\langle \epsilon, M, \epsilon, \rangle$ containing a leaf $\dfrac{\langle C, M, Path\rangle}{\cdots}$ such that there is no rule of the calculus to be applied. The clause C cannot be empty, or the Axiom rule would be applied. Besides, the clause C does not contain a blocking-free literal L such that its complement $\overline{L} \in Path$ w.r.t. σ, or the Reduction rule would be applied. Finally, there is no blocking-free copy clause C_2 of $C_1 \in M^\mu$ such that $\overline{L} \in C_2$, for some literal $L \in C$, or the Extension rule would be applied. Therefore, there exists a path $p \in \varphi(M^\mu, Path)$ over the matrix M such that p is not complementary for any multiplicity μ. □

The previous theorems state that the connection calculus is sound and complete, linking it and the matrix characterisation. However, we require an algorithm to ensure the connection method is a decision procedure for BTNF knowledge bases.

5.5 ALGORITHM AND IMPLEMENTATION

We present two algorithms that construct a proof within the connection calculus.

Algorithm 2, called **Prove**, recursively applies the rules of the calculus as outlined in Figure 6 to a given triple, verifying whether its derivations constitute connection proofs.

Algorithm 1, referred to as **Entail**, takes a matrix as input. It then applies the Start rule by repeatedly invoking **Prove** until it either returns $True$ for some clause in the matrix or $False$ if no such clause exists.

Algorithm 1 Entail

Require: A matrix M of an $\mathcal{ALCH}^{\bullet}$ knowledge base in BTNF and a concept assertion
Ensure: *True* if $\mathcal{K} \models A(a)$ or *False* otherwise

```
for each clause C ∈ M do
    if Prove(C, M, {}) is True then
        return True
    end if
end for
return False
```

Algorithm 2 Prove

Require: A (sub-)clause C, a matrix M and a (sub-)path $Path$
Ensure: *True* if there exists a connection proof for $\langle C, M, Path\rangle$ or *False* otherwise

```
if C = ∅ then
    return True
end if
for each rule R that is applicable to ⟨C, M, Path⟩ do
    for each triple ⟨C', M, Path'⟩ derived from applying R do
        if Prove(C', M, Path') is False then
            skip rule R
        end if
    end for
    return True
end for
return False
```

Algorithm **Prove** applies the calculus rules, checking whether a connection proof exists for the argument triple. It tries every applicable rule and, in a recursive way and verifies if its derived triples are connection proofs. If there is no more applicable rule or the derivations are not connection proofs, then it returns $False$. Otherwise, it returns $True$.

Theorem 5.6 (Termination)**.** *Given any $\mathcal{ALCH}^{\bullet}$ knowledge base $\mathcal{K}$, concept assertion $A(a)$, and their matrix representation M;* **Entail**(M) *terminates.*

Proof. The critical component that affects the algorithm's termination is when Extension rules are applied to clauses that create new individuals, such as $\bullet A \sqsubseteq B$, $A \sqsubseteq \neg\bullet B$, $\forall r.A \sqsubseteq B$ and $A \sqsubseteq \exists r.B$. In the other cases, once the number of concept names and individuals is finite, the number of different literals in the path will also be finite, too. These cases terminate when the set of distinct literals in the path are exhausted. Looking at the paths of the axioms mentioned earlier, every new individual is related to at least a concept, a role name or the $<$ relation for some path. As the number of concept names and role names is finite, at some point, those new individuals must either:

- be blocked by an r-ancestor term when the copied literal is in the form of $\mathrm{R}(\mathtt{T},\mathtt{T})$ or $<(\mathtt{T},\mathtt{T})$ and its set of concepts is subsumed by the set of the other term, or
- be blocked by a previously copied new individual when the copied literal is in the form of $\mathrm{C}(\mathtt{T})$ and its set of concepts is subsumed by the set of the previous term.

Therefore, the blocking will inevitably occur and the reasoning processing terminates for these cases too.

□

Theorem 5.7 (Soundness). *Given an $\mathcal{ALCH}^{\bullet}$ knowledge base $\mathcal{K}$, concept assertion $A(a)$, and their matrix representation M; if* **Entail**(M) *returns $True$, then $\mathcal{K} \models A(a)$.*

Proof. The proof is a direct consequence of the Matrix characterisation (Theorem 5.3) and the soundness of the calculus (Theorem 5.4). □

Theorem 5.8 (Completeness). *Given an $\mathcal{ALCH}^{\bullet}$ knowledge base $\mathcal{K}$, concept assertion $A(a)$, and their matrix representation M; if $\mathcal{K} \models A(a)$, then* **Entail**(M) *returns $True$*

Proof. The proof is a consequence of the Matrix characterisation (Theorem 5.3) and the completeness of the calculus (Theorem 5.5). □

5.6 EXAMPLE OF PROOF

This section presents an example to illustrate how the proposal checks whether a knowledge base entails an assertion. First, let $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ be the knowledge base of Example[2] 5.1, where

$$\mathcal{T} = \{\exists \mathrm{mT.PBW} \sqsubseteq \bullet\mathrm{W}\}, \mathcal{R} = \{\bullet\mathrm{l} \sqsubseteq \mathrm{mT}\}, \text{ and } \mathcal{A} = \{\bullet\mathrm{l}(\mathrm{h},\mathrm{g}), \mathrm{PBW}(\mathrm{g})\}. \tag{5.1}$$

We want to verify whether $\mathcal{K} \models \mathrm{W}(\mathrm{h})$, i.e., whether $\mathcal{K}$ entails that Harry Potter is a wizard. The first step is to transform $\mathcal{K}$ into BTNF. So, we add a new concept name N and a new role name N_r to convert $\mathcal{K}$ into $\mathcal{K}' = (\mathcal{T}', \mathcal{R}', \mathcal{A}')$ such that

$$\begin{aligned} &\mathcal{T}' = \{\exists \mathrm{mT.PBW} \sqsubseteq N, N \sqsubseteq \bullet\mathrm{W}, \bullet\mathrm{W} \sqsubseteq N\}, \\ &\mathcal{R}' = \{\bullet\mathrm{l} \sqsubseteq \mathrm{mT}, N_r \sqsubseteq \bullet\mathrm{l}\}, \text{ and } \mathcal{A}' = \{N_r(\mathrm{h},\mathrm{g}), \mathrm{PBW}(\mathrm{g})\}. \end{aligned} \tag{5.2}$$

[2] To improve readability, we will use abbreviations for the names mentioned.

Now, we can represent the knowledge base as a matrix. Figure 8 shows the matrix representation of $\mathcal{K}'$, according to Tables 4 and 5. The Figure also displays $\mathrm{W}(\mathrm{h})$, the assertion we want to deduce. It remains unchanged as the connection method checks validity instead of refutational approaches such as Tableaux.

We may not need to use all the matrix clauses in every proof, similar to how knowledge bases can include axioms that do not impact a query's entailment. As a result, we will now omit unnecessary clauses and rearrange the order of clauses and literals to provide clarity.

We perform the connection method in Figure 9. Red curves between literals represent the application of the Extension or Reduction rules. The last clause is a copy of clause 6, necessary because two literals require the same literals of clause 6. We illustrate that by adding the symbol $'$ after the variable, i.e., x'_{17} is a copy of x_{17}. The goal is to test whether every clause associated with a connection is fully connected; otherwise, the derivation is not a connection proof. The reasoning starts with $\mathrm{W}(\mathrm{h})$, the possible entailed assertion, connecting it to a complementary literal $\neg\mathrm{W}(\mathrm{h})$ in the third clause and unifying $\sigma = \{x_2 \setminus \mathrm{h}\}$. Continuing the proof, $N(x_2)$, the other literal in the third clause, must be proven. We connect $N(x_2)$ with $\neg N(x_0)$ in the fourth clause, where $\sigma(x_0) = \mathrm{h}$. The other connections follow the same approach. The ninth connection (9) is settled by the reduction rule, instead of the extension rule, given we already have its σ-complementary literal in the active path. Thus, as every clause associated with a connection is fully connected, we prove that $\mathcal{K} \models \mathrm{W}(\mathrm{h})$.

Another useful representation of the connection method is the sequent-style derivation tree. It shows the triples during proof search and applies the rules until no rule can be further triggered. If every leaf is an Axiom, i.e., the subgoal is fully proven, then the proof tree is a connection proof of M. Figure 10 deploys the examples's sequent-style proof.

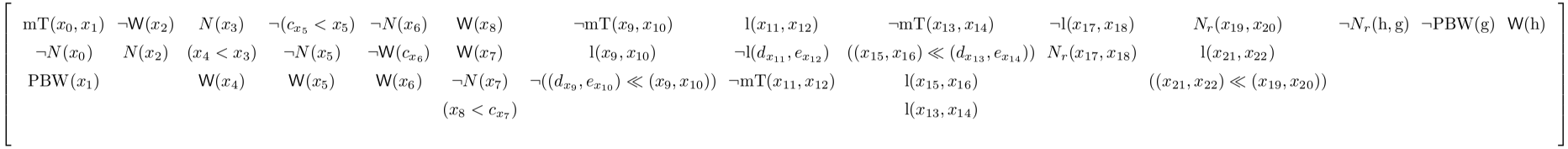
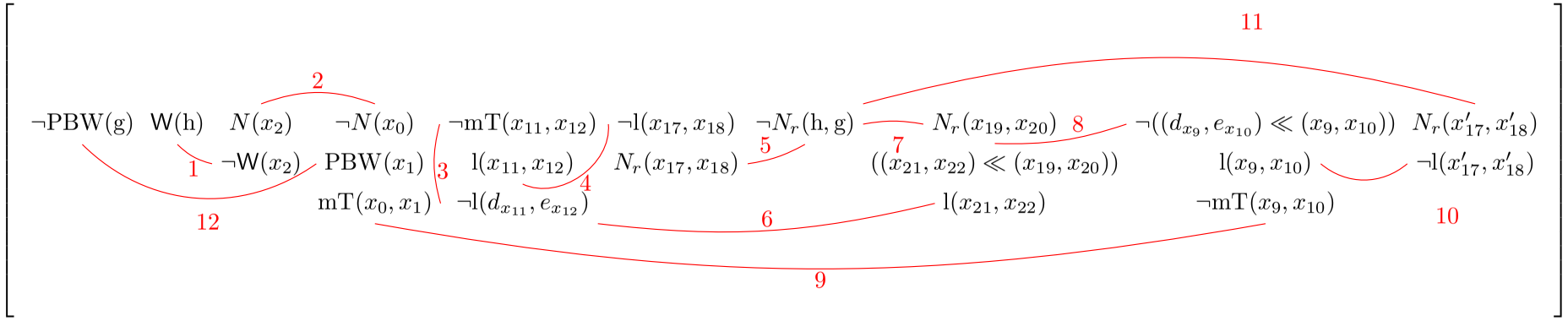


Figure 8 – Matrix representation of $\mathcal{K}'$ and $\mathrm{W}(\mathrm{h})$.



$1.\sigma_1 = \{x_2 \setminus \mathrm{h}\}$ $2.\sigma_2 = \sigma_1 \cup \{x_0 \setminus \mathrm{h}\}$ $3.\sigma_3 = \sigma_2\{x_{11} \setminus \mathrm{h}, x_{12} \setminus x_1\}$ $4.\sigma_4 = \sigma_3\{x_{17} \setminus \mathrm{h}, x_{18} \setminus x_1\}$

$5.\sigma_5 = \sigma_4\{x_1 \setminus \mathrm{g}\}$ $6.\sigma_6 = \sigma_5\{x_{21} \setminus d_{\mathrm{h}}, x_{22} \setminus e_{\mathrm{g}}\}$ $7.\sigma_7 = \sigma_6\{x_{19} \setminus \mathrm{h}, x_{20} \setminus \mathrm{g}\}$ $8.\sigma_8 = \sigma_7\{x_9 \setminus \mathrm{h}, x_{10} \setminus \mathrm{g}\}$

9.Reduction with 3 $10.\sigma_9 = \sigma_8\{x'_{17} \setminus \mathrm{h}, x'_{18} \setminus \mathrm{g}\}$

Figure 9 – Proof of connection method in the matrix style. The proof starts with the connection 1 and so on. Enumerated substitutions below the matrix show the substitution in each step (connection).

$$\dfrac{\dfrac{\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}}),\mathrm{l}(x_9,x_{10}),N_r(x'_{17},x'_{18})\}}\,\mathrm{Ax}\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}}),\mathrm{l}(x_9,x_{10})\}}\,\mathrm{Ax}}{\{N_r(x'_{17},x'_{18})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}}),\mathrm{l}(x_9,x_{10})\}}\,Ext}{\text{Proof3}}$$

$$\dfrac{\dfrac{\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}}),N_r(x_{19},x_{20})\}}\,\mathrm{Ax}\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,\mathrm{Ax}}{\{N_r(x_{19},x_{20})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,Ext}{\text{Proof2}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{Proof3}\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,\mathrm{Ax}}{\{\mathrm{l}(x_9,x_{10})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,Ext}{\{\neg\mathrm{mT}(x_9,x_{10}),\mathrm{l}(x_9,x_{10})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),((x_{21},x_{22})\ll(x_{19},x_{20})),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,Red\quad\text{Proof2}}{\{((x_{21},x_{22})\ll(x_{19},x_{20})),N_r(x_{19},x_{20})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\}}\,Ext\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2)\}}\,\mathrm{Ax}}{\{\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2)\}}\,Ext}{\text{Proof1}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),\mathrm{l}(x_{11},x_{12}),N(x_2),N_r(x_{17},x_{18})\}}\,\mathrm{Ax}\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),\mathrm{l}(x_{11},x_{12}),N(x_2)\}}\,\mathrm{Ax}}{\{N_r(x_{17},x_{18})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),\mathrm{l}(x_{11},x_{12}),N(x_2)\}}\,Ext\quad\text{Proof1}}{\{\mathrm{l}(x_{11},x_{12}),\neg\mathrm{l}(d_{x_{11}},e_{x_{12}})\},M,\{\mathsf{W}(\mathrm{h}),\mathrm{mT}(x_0,x_1),N(x_2)\}}\,Ext\quad\dfrac{\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),N(x_2),\mathrm{PBW}(x_1)\}}\,\mathrm{Ax}\quad\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h}),N(x_2)\}}\,\mathrm{Ax}}{\{\mathrm{PBW}(x_1)\},M,\{\mathsf{W}(\mathrm{h}),N(x_2)\}}\,Ext}{\{\mathrm{mT}(x_0,x_1),\mathrm{PBW}(x_1)\},M,\{\mathsf{W}(\mathrm{h}),N(x_2)\}}\,Ext\quad\dfrac{\dfrac{}{\{\},M,\{\mathsf{W}(\mathrm{h})\}}\,\mathrm{Ax}}{\{N(x_2)\},M,\{\mathsf{W}(\mathrm{h})\}}\,Ext\quad\dfrac{}{\{\},M,\{\}}\,\mathrm{Ax}}{\{\mathsf{W}(\mathrm{h})\},M,\{\}}\,Ext}{\varepsilon,M,\varepsilon}\,St$$

Figure 10 – The connection method in sequent-style. It starts from the bottom to the top. As every leaf is an Axiom, we call this proof derivation a connection proof.

5.7 COMPLEXITY

In addition to the results presented in this chapter, it is important to investigate the complexity of the proposed algorithm. To achieve this, we will base our analysis on the proof tree that arises from applying the calculus rules illustrated in Figure 6.

In the worst-case scenario, each application of a rule (Extension) leads to the proof being divided into two parts. As a result, with each successive application of the rule, the proof continues to split into two. Therefore, the proof tree can be visualized as a binary tree. Figure 11 illustrates the sequence of applying the extension rules.

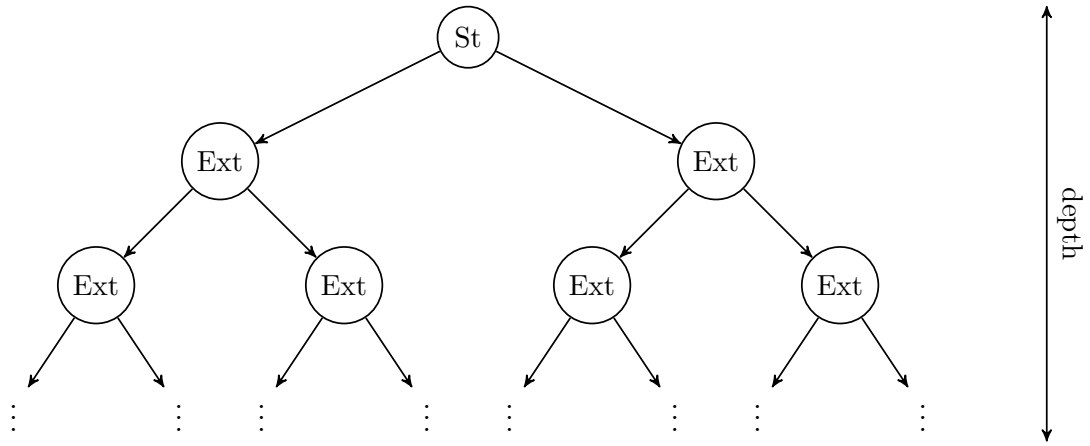


Figure 11 – Proof tree expansion after (Extension) rule applications.

A key element in understanding the algorithm's complexity is identifying the maximum number of times a clause can copy before it gets blocked. We will classify the results based on whether the matrix is cyclic.

Lemma 5.4 (Maximum multiplicity (acyclic case))**.** *Given an acyclic matrix M, the maximum multiplicity μ_{max} of a clause in M is $(|\mathrm{I}| + |\mathrm{A}|)^4$, where I is the set of individual names of M and A is the set of new individuals of M.*

Proof. By Definition 5.7, the terms are substituted by individuals or new individuals only. Besides, by Definition 5.19, there is no duplicate of a literal in the path. Hence, the number of clause copies is bounded by the cardinality of both sets of individual names and new individuals.

Furthermore, the literals of the form $(x, y) \ll (z, k)$ contain the clause's maximum number of terms (four). Therefore, the maximum multiplicity of a clause is $(|\mathrm{I}| + |\mathrm{A}|)^4$. □

Lemma 5.5 (Maximum multiplicity (cyclic case))**.** *Given a cyclic matrix M, the maximum multiplicity μ_{max} of a clause in M is $(|\mathrm{I}| + |M|2^{|\mathrm{C}|})^4$, where I is the set of individual names of M and C is the set of concept names M.*

Proof. By Definition 5.7, the terms are substituted by individuals or new individuals only. Besides, by Definition 5.17, terms must be blocked if their concepts are subset or equal to their $<$-descendants or r-ancestors. Once the new individual's creation is linked to role and typicality literals, the maximum number of new individuals is bounded by the power set of the concept names times the size of the matrix. Furthermore, the literals of the form $(x, y) \ll (z, k)$ contain the clause's maximum number of terms. Therefore, the maximum multiplicity of a clause is $(|\mathrm{I}| + |M|2^{|\mathrm{C}|})^4$. □

After establishing μ_{max}, we can determine the maximum depth of the proof tree.

Corollary 5.1. Given a matrix M and its maximum multiplicity μ_{max}, the maximum depth of a proof tree of M is $2^{|M|\mu_{max}}$.

Finally, when considering the maximum depth of the proof tree, we can also conduct an asymptotic analysis regarding the number of clauses in the matrix. Since the sets of individual names, new individual names, and concept names are proportional fractions of the size of the matrix, we can conclude that the complexity of the algorithm is $O(2^{2^{n^4}})$, where n is the size of the matrix (input). This result aligns with the conjecture presented by Varzinczak (VARZINCZAK, 2018), which states that his algorithm also exhibits double exponential complexity.

6 POLYCOP: A POLYMORPHIC CONNECTION IMPLEMENTATION

In the previous chapter, we introduced a method for connecting the defeasible logic $\mathcal{ALCH}^{\bullet}$. The next step is to create a prover based on this method.

Most - if not all - reasoning calculi (tableaux, resolution, natural deduction, etc) feature uniformity when employed in different logics; for instance, tableaux for first-order is equal to its propositional version, if not for the additional rules for quantification and the use of unification.

This gives rise to the idea of implementing a polymorphic reasoner, one that could work over an array of different logics, where each logic´s specifities is dealt properly: a connection in first-order must include unification spanning over the whole matrix, distinct forms of blocking, according to the DL fragment being dealt, etc. In a field cramped with the needs for efficiency, to the best of our knowledge, no polymorphic reasoner - whose main pro is being reusable and emphasizing the subtle differences in inference among logics - has been proposed yet.

Considering these factors, we have developed PolyCoP, a polymorphic and reusable prover capable of handling potentially any logic. We focus on maintaining a single, reusable main algorithm responsible for proof search. Furthermore, new code related to various existing logics can be added to PolyCoP, allowing it to function as a prover for these logics.

Our approach differs from Otten's family of provers, as the source codes vary despite similarities in implementations. Any improvements or changes to the family's code would require modifying the source code of each prover.

Thus, the objectives of this work are:

1. to build a connection prover for $\mathcal{ALCH}^{\bullet}$;
2. to develop a polymorphic connection method prover capable of proving numerous logics by reusing its core implementation and coding the slight differences locally; and
3. to implement a simple prover/framework allowing beginners to understand and operate a connection method prover.

On the other hand, up to this point, this framework does not aim to provide an optimized and highly competitive prover, as our goal is to show a simpler and more general version of a connection prover.

PolyCoP isolates the code related to the proof search and general method from the code related to the logical language, such as:

- how literals can be complementary;
- how complementary literals connect;
- how a clause is copied; and
- when the proof must be blocked.

The logical-related specifications aforementioned are represented as strategies. Figure 12 gives a glimpse of PolyCoP 's capabilities. The idea is that developers who want to design a prover for a given logic implement only the strategies' interfaces. In other words, when implementing a specific logic connection reasoner, the code related to the proof is reused. In Figure 12, such code is represented by the UniqueProver class.

The developer may focus only on a code fragment related to the given logical language (for propositional logic or any other logic), such as how literals are complementary, connected, clauses are copied and the inference is blocked.

The prover's input is a formula that needs to be mapped to a matrix by some user-specific code — a mapper. The matrix must be linked to strategies to connect, copy, or block the literals of the matrix according to the developer's specification. Those strategies establish how the prover utilizes the literals in the matrix.

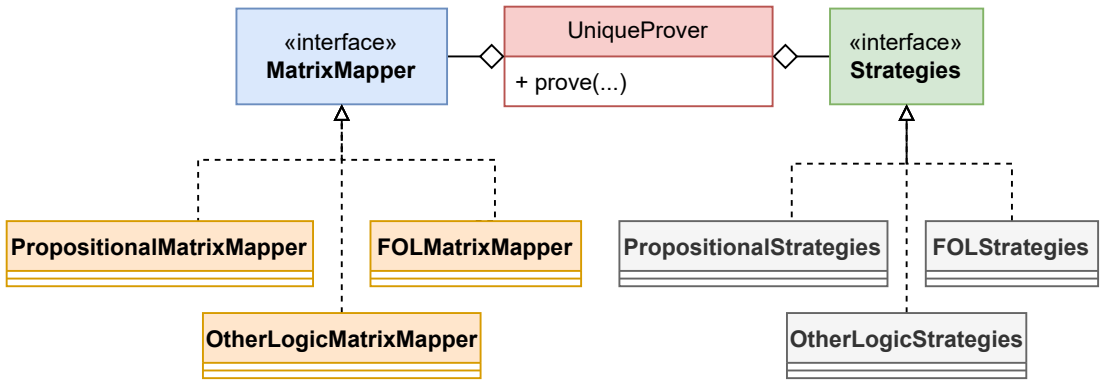


Figure 12 – UML diagram of PolyCoP.

6.1 A GENERAL CONNECTION METHOD ALGORITHM

This section describes the algorithm for conducting proof search in PolyCoP. In the previous chapter, we introduced an algorithm to determine whether a knowledge base in BTNF entails a

query. This algorithm relies on specific rules for that logic, similar to other proving algorithms. To create an algorithm that can be applied to a different logic, we need to separate the code related to the logic from the one related to the proof.

We start defining a notation to illustrate our connection method's generalisation better. With $\mathcal{L}$, we define the logical language; with $\Sigma_{\mathcal{L}}$, we define the set of possible literals in $\mathcal{L}$; with $\mathcal{P}(\Sigma_{\mathcal{L}})$, we represent the power set of possible literals, i.e., every set of literals is an element of $\mathcal{P}(\Sigma_{\mathcal{L}})$. With $\langle \sigma, S_\sigma \rangle$, we represent the unification strategy , where $\sigma : \Sigma_{\mathcal{L}} \longrightarrow \Sigma_{\mathcal{L}}$ is a function that maps a literal of the target logical language to a term substituted literal, and S_σ is the state of the unification strategy.

The representation of copy strategy is $\langle \phi, S_\phi \rangle$, where $\phi : \mathcal{P}(\Sigma_{\mathcal{L}}) \longrightarrow \mathcal{P}(\Sigma_{\mathcal{L}})$ is a function that copies a clause, i.e., maps a set of literals to a set of literals, and S_ϕ is the state of the copy strategy.

We represent a blocking strategy as a function $\upsilon : (S_\sigma)^* \times (S_\phi)^* \longrightarrow \{true, false\}$ that maps a pair of states (unification and copy) to $true$ if we want to block the proof and $false$, otherwise.

Algorithm 3 SimpleProver

Require: A matrix M
Ensure: *True* if M is spanning or *False* otherwise
1: creates the copy strategy $\langle \phi, S_\phi \rangle$
2: $S'_\phi \leftarrow S_\phi$
3: **for** each clause $C \in M$ **do**
4: $C' \leftarrow \phi(C)$
5: updates the state S_ϕ
6: $Proof \leftarrow$ **Prove**$(C', M, \{\})$
7: **if** $Proof$ is a connection proof **then**
8: **return** $\dfrac{Proof}{\varepsilon, M, \varepsilon}$ St
9: **end if**
10: $S_\phi \leftarrow S'_\phi$
11: **end for**
12: **return** $\dfrac{}{\varepsilon, M, \varepsilon}$

Algorithm 3 presents a simple general connection method. Initially, it builds a copy strategy instance and saves its initial state. The State design pattern consists of the general CM storing states of copy and connection strategies for recovery during backtracking and for future parallel approaches. The loop iterates over clauses and searches for a start clause with a proof tree that serves as a connection proof. Within the loop, the algorithm calls the Algorithm 4.

Algorithm 4 Prove

Require: A (sub-)clause C, a matrix M, a path $Path$, a copy strategy $\langle \phi, S_\phi \rangle$, a unification strategy $\langle \sigma, S_\sigma \rangle$, and a blocking function υ

Ensure: a proof for $C, M, Path$

1: **if** $C = \emptyset$ **then return** $\dfrac{}{C, M, Path}$ Ax
2: **end if**
3: $S'_\sigma \leftarrow S_\sigma$
4: gets some $L \in C$
5: **for** each literal $\overline{L} \in Path$ **do**
6: **if** $\{\sigma(L), \sigma(\overline{L})\}$ is a valid connection **then**
7: update the state S_σ
8: $Proof \leftarrow$ **Prove**$(C \setminus \{L\}, M, \{\})$
9: **if** $Proof$ is a connection proof **then**
10: **return** $\dfrac{Proof}{C, M, Path}$ Red
11: **end if**
12: $S_\sigma \leftarrow S'_\sigma$
13: **end if**
14: **end for**
15: $S'_\phi \leftarrow S_\phi$
16: **for** each clause $C_2 \in M$ **do**
17: $C'_2 \leftarrow \phi(C_2)$
18: update the state S_ϕ
19: **for** each literal $\overline{L} \in C'_2$ **do**
20: **if** $\{\sigma(L), \sigma(\overline{L})\}$ is a valid connection **then**
21: update the state S_σ
22: **if** $\upsilon(S_\sigma, S_\phi)$ **then**
23: $S_\sigma \leftarrow S'_\sigma$
24: skip $\overline{L}$ iteration
25: **end if**
26: $LeftProof \leftarrow$ **Prove**$(C'_2 \setminus \{\overline{L}\}, M, Path \cup \{L\})$
27: **if** $LeftProof$ is a connection proof **then**
28: $RightProof \leftarrow$ **Prove**$(C \setminus \{L\}, M, Path)$
29: **if** $RightProof$ is a connection proof **then**
30: **return** $\dfrac{LeftProof \quad RightProof}{C, M, Path}$ Ext
31: **end if**
32: **end if**
33: $S_\sigma \leftarrow S'_\sigma$
34: **end if**
35: **end for**
36: $S_\phi \leftarrow S'_\phi$
37: **end for**
38: **return** $\dfrac{}{\varepsilon, M, \varepsilon}$

We designed the Algorithm 4 to determine whether a tuple has a valid connection proof. It recursively applies the Axiom, Reduction, and Extension Rules. First, it checks if the subgoal is empty to apply the Axiom Rule. If the subgoal is not empty, *Prove* tries to apply the Reduction Rule, unifying the literal with complementary ones in the path. The algorithm also stores the connection state to restore to the previous state if necessary—the remaining part of the implementation attempts to apply the Extension Rule. In addition to the connection state, it stores the copy state of the copy strategy. The recursive call of the Extension rule recovers the copy state if the inner subproofs are not connection proofs. This behaviour ensures that backtracking works appropriately. The blocking function is only applied over Extension Rules, as it is the only rule that can lead to an infinite loop. The function looks at the state of connections and copies and blocks (or not) the recursive calls for inner scenarios.

Due to CM's uniformity (BIBEL, 2017), the calculi and the algorithm are quite analogous to the ones in Chapter 5. This general implementation's key distinction lies in this aspect: only one algorithm is required to carry out the proof in any logic, where its specificities must be defined in lower-level, overriding methods. Given that the language-dependent functions operate as detailed in the definitions and demonstrations of that chapter, this general prover also serves as a prover for $\mathcal{ALCH}^{\bullet}$.

6.2 IMPLEMENTATION

PolyCoP, the general and polymorphic connection method written in Java, is available at GitHub[1]. The architecture relies on Java features, such as interfaces and generics, and the design patterns(GAMMA et al., 1994) strategy and state. The CoP implementation calls the strategies methods to prove the formula using their interfaces. So, the PolyCoP can run its proof search using different logic and algorithms by changing the interface implementations.

6.2.1 Strategies

There are four different strategy interfaces: `LiteralHelperStrategy`, `ConnectionStrategy`, `CopyStrategy`, and `BlockingStrategy`.

The strategy called `LiteralHelperStrategy` allows developers to define complementary literals and implement the logic behind them. This is essential for the prover to find comple-

[1] <https://github.com/renanlf/polycop>

mentary literals of a given one. The implementation takes a given literal L and the matrix M as input. The developer's code must return a submatrix of clauses M' where each clause has a complementary literal of L. The meaning of complementary can change based on the logic being implemented.

The `ConnectionStrategy` plays a crucial role in maintaining the correct behaviour of the code. It returns a Boolean value indicating whether the connection is valid. The inputs are the literals to be connected, the active path, and the current state of the strategy. The developer must check the current state and the path and update the strategy's current state to ensure the correct behaviour of the code. The state update must create another object representation of it instead of changing the properties of the current state. This allows for fast restoration of previous states when backtracking occurs.

As the name suggests, the `CopyStrategy` deals with copying clauses in the matrix. Some logic may require copying clauses to be reused in another subproof later. The input to this strategy is the clause to be copied and the current state of the strategies. These inputs help the developer check whether a copy occurs.

Finally, the `BlockingStrategy` helps the developer determine when the proof needs to stop the search. It returns *true* when the proof needs to stop the search after a connection and *false* otherwise. The input to this strategy is the current state of the other strategies. The developer can use this strategy to check the soundness of the proof after a connection instead of checking it before the connection happens.

In Figure 13, we detailed the UML class diagram of PolyCoP.

6.2.2 Implementation for propositional logic

To illustrate the PolyCoP polymorphism, we show the implementation of the well-known propositional logic in this chapter. The main reason for using it as PolyCoP's starter logic is to focus the reader on the system features without requiring a background in other logic. Nevertheless, this decision does not restrict PolyCoP to simple logic. The code repository also contains other implementations for first-order and defeasible Description Logic.

Our implementation's input is files in the well-known CNF format. It represents the literals as integers, where a negative number $-N$ denotes the negation of the literal N. Because it is already a clausal representation, we straightforwardly translate the values to the matrix, multiplying them by -1, to obtain the DNF version. The intuition is that once we negate the

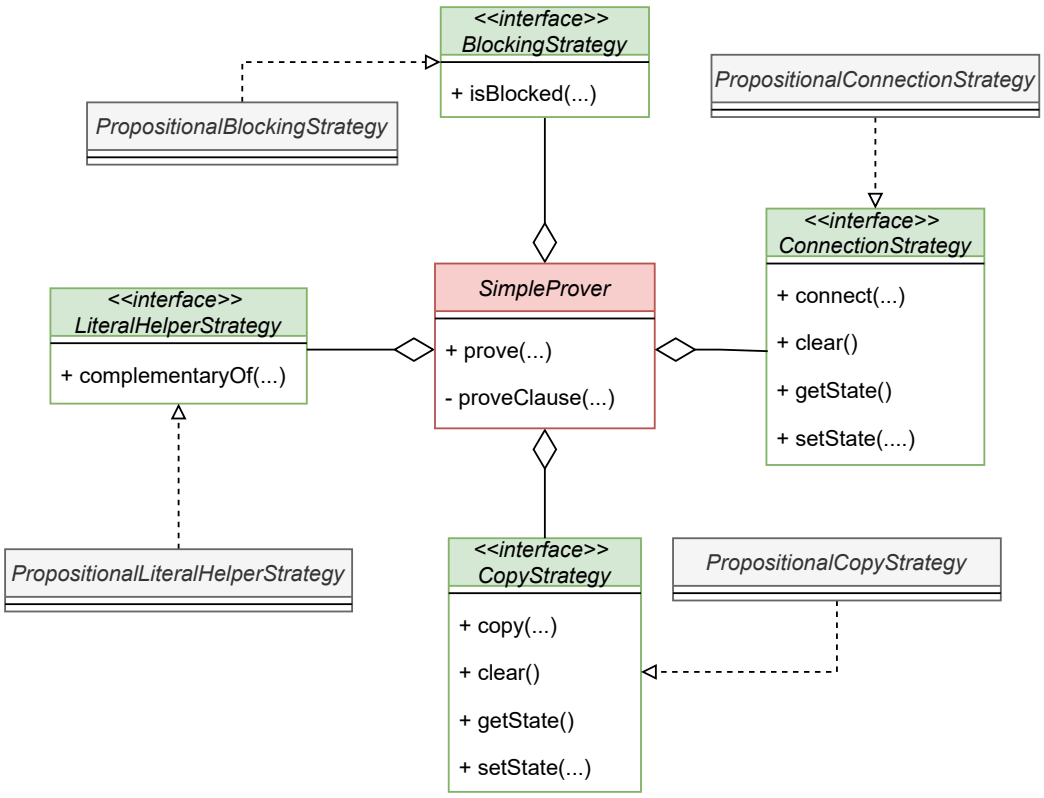


Figure 13 – Detailed UML class diagram of PolyCoP strategies and the prover

formula in CNF (because CM is a direct method), it is the same as changing the polarity of its literals.

Figure 14 shows a CNF file. The file's first line denotes the unsigned max number for a literal and the number of clauses. Each line represents a clause in the matrix from the second line to the end of the file. The 0 (zero) at the end of these lines denotes the end of the clause. So, once negative numbers are the negation of a literal, the line $-1\ 2\ 0$ represents a clause $\{\neg 1, 2\}$.

```
p cnf 3 4
1 0
-1 2 0
-2 -3 0
3 0
```

Figure 14 – CNF file.

Delving deeper into the data structures of PolyCoP, we use linked nodes to store the states through the search proof. The main reason to do so is that this approach optimizes memory usage when the proof tree becomes deeper.

To illustrate the memory benefits, assume the code stores a new object every time a proof rule is applied. So, proving some literal in a 10-depth proof tree in an immutable stateful scenario must contain states with 0 to 9 objects each. The code must store 37 references to represent those ten states.

However, the number of references needed decreases if the code uses linked nodes to store the states. The 10th state only needs to store one object's reference and the previous linked node's reference. The previous linked node also requires two references, and so on. The proof must store 18 references to represent the same ten states.

Furthermore, the connection proof result is a tree where each node represents a different tuple of the connection proof search. The root is the Start Rule, and each rule's application appends new children to the nodes. Such an approach is helpful for human-readable search proofs.

Our code provides an implementation that translates the connection proof tree as a LaTeX string, allowing the developer to represent the connection proof in a sequent-style.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\}, M, [1,2,-3]}\,\text{Ax} \quad \dfrac{}{\{\}, M, [1,2]}\,\text{Ax}}{[-3], M, [1,2]}\,Ext \quad \dfrac{}{\{\}, M, [1]}\,\text{Ax}}{[2], M, [1]}\,Ext \quad \dfrac{}{\{\}, M, []}\,\text{Ax}}{[1], M, []}\,Ext}{\varepsilon, M, \varepsilon}\,St$$

Figure 15 – Proof in sequent-style interpreted by a LaTeX system of PolyCoP generated proof.

Figure 16 shows a fragment of the code generated by PolyCoP. Utilizing PolyCoP, the developer can export it to a file and generate the corresponding sequent-style proof.

```
\documentclass[convert={outext=.eps, command=\unexpanded{pdftops -eps \infile}}]{standalone}
\usepackage{bussproofs} % for sequent-style proofs
\begin{document}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$\{\}, M, [1, 2, -3]$}
\AxiomC{}
\RightLabel{Ax}
\UnaryInfC{$\{\}, M, [1, 2]$}
\RightLabel{\textit{Ext}}
\BinaryInfC{$[-3], M, [1, 2]$}
```

Figure 16 – Fragment of code generated by PolyCoP of Example's proof.

Besides, implementing strategies for a logic turns PolyCoP into a prover of that logic. Thus, implementing strategies for first-order logic (fol), for example, becomes PolyCoP a first-order logic prover. The blocking strategy can be ignored due to the nature of the undecidable first-order logic formulae. One can also implement a blocking strategy that stops when it reaches some condition to prune the proof search. Those different strategies are possible in PolyCoP, and developers are welcome to try any envisioned logic.

Finally, for PolyCoP to be a $\mathcal{ALC}$(CAI; MING; LI, 2008) prover, for example, the blocking strategy must map to $true$ at some point once $\mathcal{ALC}$ is a decidable fragment of description logics.

6.2.3 $\mathcal{ALCH}^{\bullet}$'s implementation

The main design objective of our implementation was to establish a direct parallel between Java classes and the main concepts of the defeasible DL $\mathcal{ALCH}^{\bullet}$. To achieve this, we created a class called `ALCHbTerm` to represent a term, which serves as the parent class for `ALCHbVariable`, `ALCHbIndividual`, and `ALCHbUnaryIndividual`. These subclasses represent variables, individuals, and new individuals, respectively. Additionally, we built a Java interface to represent a literal, which serves as the parent class for the subclasses representing concepts, roles, ordered literals, and bi-ordered literals. The latter represents the typicality relation between objects and pairs of objects.

In terms of strategies, we implemented the Helper strategy to indicate when a literal complements another. Our implementation only checks the literal type and its polarity. For example, according to our helper strategy, $A(a)$ and $\neg A(b)$ are considered complementary, even if this is not the case. We did this to simplify the search and leave the responsibility of preventing such a connection to the connection strategy.

The connection strategy unifies the terms of a literal. Its responsibility is to check whether the literal types are the same and whether they can unify their terms while respecting the order in which they appear in the literal.

The copy strategy is responsible for creating copies. It creates new clauses where new variables replace their former counterparts. We use the symbol ' to denote a copy. For example, the variable x' is the first copy of the variable x, x'' is the second, and so on.

The connection and copy strategies assume states that help maintain relevant information for the proof. In our case, they store substitutions and copied variables. Besides, relying on

states allows us to use them in the blocking strategy. The blocking strategy acts similarly to the blocking condition described in the previous chapter. Its main difference is that it does not look for duplicated literals since the proof-related code already handles such repetitions.

6.2.3.1 Implementation of OWL fragment

The defeasible Description Logic $\mathcal{ALCH}^{\bullet}$ currently lacks a specification for modelling knowledge bases in a file format. However, its classical fragment $\mathcal{ALCH}$ can be represented using OWL files, Semantic Web specifications for ontologies (ANTONIOU et al., 2012). PolyCoP has been enhanced as a reasoner for this fragment $\mathcal{ALCH}$, allowing for the verification of consistency in OWL knowledge bases during execution.

We convert ontologies from OWL to literals using the OWL API (HORRIDGE; BECHHOFER, 2011). The OWL API is a Java Application Programming Interface that provides a straightforward connection between the OWL 2 specification and Java programming language objects. While other implementations, like Raccoon, develop their parsers from scratch, we utilize the OWL API to abstract the parsing process, allowing us to concentrate solely on the reasoner implementation.

To illustrate our implementation, consider the following ontology in a functional style:

```
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(:=<http://cin.ufpe.br/~dldmf/raccoon/test_cyclic_inconsistency002.owl#>)
Ontology(<http://cin.ufpe.br/~dldmf/raccoon/test_cyclic_inconsistency002.owl>
    SubClassOf(ObjectSomeValuesFrom(:R :A) :A)
    SubClassOf(:B :A)
    ObjectPropertyAssertion(:R :a :b)
    ObjectPropertyAssertion(:R :b :c)
    ObjectPropertyAssertion(:R :c :d)
    ClassAssertion(:B :d)
    ClassAssertion(ObjectComplementOf(:A) :a)
)
```

The OWL API parses the ontology into a Java object. Next, the `ALCHbOWLAPIMapper` translates the ontology into a matrix based on the Algorithm 5 provided in the Appendix. Figure 17 illustrates the resulting matrix after the mapping. The OWL API parses the ontology into a Java object. Next, the `ALCHbOWLAPIMapper` translates the ontology into a matrix based on the Algorithm 5 provided in the Appendix. Figure 17 illustrates the resulting matrix after the mapping.

$$\begin{bmatrix} \neg R(a,b) & \neg R(b,c) & \neg R(c,d) & A(x_2) & B(x_3) & \neg B(d) & A(a) \\ & & & R(x_1,x_2) & \neg A(x_3) & & \\ & & & \neg A(x_1) & & & \\ & & & & & & \end{bmatrix}$$

Figure 17 – Resulting matrix after the mapping of `ALCHbOWLAPIMapper`.

Finally, as demonstrated in the propositional case, our code implements the proof tree in sequent style, using a LaTeX string. Figure 18 illustrates the proof.

Therefore, the ontology is inconsistent once the proof is a connection proof.

Currently, we do not have performance benchmark tests. However, our initial tests were designed to evaluate the algorithm's correctness. For this purpose, we utilized 24 ontologies from the Raccoon repository as a reference. PolyCoP verified the consistency of all these ontologies, providing a reliable indicator that our implementation is on the right track. We plan to conduct experimental tests in the future.

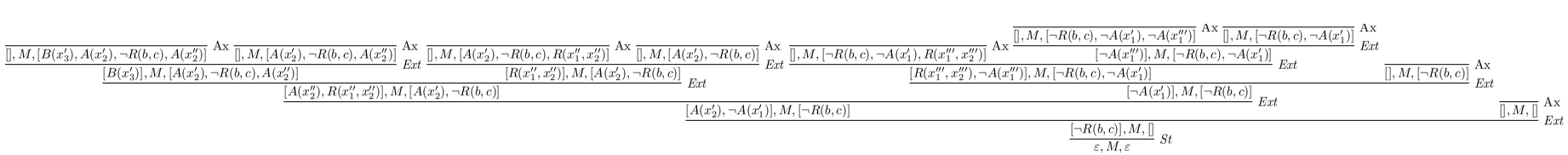


Figure 18 – Resulting connection proof.

6.3 CONCLUDING REMARKS

This chapter presented the PolyCoP, a polymorphic connection prover for $\mathcal{ALCH}^{\bullet}$ and beyond. What started as an implementation for our connection method became a general polymorphic CM prover that benefits from the uniformity of CMs to implement a single prover to possibly any logical language. We could not find any other polymorphic reasoning in the literature.

This is the first implementation of the general polymorphic connection prover. Further optimized implementations will reuse the Java strategy interfaces without any change.[2]

Take the lemmata optimization (OTTEN, 2010), for instance. It reuses subproofs of literals if they already have been proved in the proof search. We can provide such behaviour by adding a new structure to store proofs and look for them each time PolyCoP calls the proof method. If N logical language strategies were implemented, the optimization would improve the proof search for the N logical languages. Another easy win is to extend the implementation to a multi-thread approach once immutable states can be shared with less or no effort between the threads during proof search.

[2] We understand that, as a first version, a few changes may be necessary until a stable general connection prover is obtained.

7 RELATED WORK

In the previous chapters, we proposed a new connection method (CM) for $\mathcal{ALCH}^{\bullet}$ and its reasoner PolyCoP, a connection prover for any logic. PolyCoP is the first reasoner (implementation) for defeasible DLs based on a connection method. This chapter outlines other CMs and defeasible description logic reasoners related to our proposal.

7.1 DEFEASIBLE DESCRIPTION LOGICS

The first reasoning method for the defeasible DL $\mathcal{ALCH}^{\bullet}$ was a semantic tableau (VARZINCZAK, 2018). It incorporates the rules from previous DL tableaux and deals with the typical operators with novel rules. Such rules add new facts to the branch according to the $\mathcal{ALCH}^{\bullet}$ semantics. Both calculi share a similar blocking condition and preferential relation representation. The main difference between the tableau and the CM proposed here is the goal orientation of CMs. Each extension occurs through a connection between literals in CM. It is the same as applying a tableau rule only to derivations containing a clash. While CM requires such connections to apply for an extension, tableau may apply rules arbitrarily.

$\mathcal{ALC}+T$ (GIORDANO et al., 2009) is another defeasible DL. It denotes typical concepts of C as a new concept constructor $T(C)$. Instead of as the one for $\mathcal{ALCH}^{\bullet}$, the modal operators inspire the tableau rules of this method, and labels represent the preferential relation during the proof. The authors also proposed another semantic tableau for $\mathcal{ALC}+T^{min}$, which does reasoning decreasing "atypical" instances from the consequences. $\mathcal{ALCH}^{\bullet}$ is a more expressive defeasible DL once the typical operator can appear on the right side of axioms such as $A \sqsubseteq \bullet B$.

7.2 CONNECTION METHODS

The most prominent implementation of the connection method is leanCoP (OTTEN; BIBEL, 2003), proposed by J. Otten and W. Bibel in the 2000s. It is a first-order prover developed in Prolog, a well-known logic programming language that recursively finds the proof by applying Reduction and Extension rules in a clausal matrix. The Prolog internal structure and storage representation benefits the leanCoP implementation, allowing it to be a seven-line program. The only optimization in leanCoP's first version was the positive start clause technique, which,

as its name suggests, starts the proof search only with positive clauses. In further versions, such as leanCoP 2.0 (OTTEN, 2008), the prover was enhanced by other features, such as

1. **Regularity** - a condition that blocks a literal from occurring more than once in the path and reduces the search space;
2. **Lemmata** - a technique that reuses subproofs of a literal for the same one later in the proof;
3. **Restricted backtracking** - a technique that also reduces the search space, blocking clauses from being used such that a literal from them is already proven and
4. **Readable proofs** - an output parameter that allows users to check the resulting proof in different formats, including a natural language proof.

PolyCoP and leanCoP are based on very similar calculi. However, leanCoP is a way more mature prover, containing variants and a successful 20 years of research and experiments (OTTEN, 2023).

7.2.1 MleanCoP and ileanCoP

MleanCoP (OTTEN, 2014) and ileanCoP (OTTEN, 2008) are variants of leanCoP based on modal and intuitionistic connection calculi (OTTEN, 2012), respectively. The novelty in such calculi is using prefixes to represent the modal and intuitionistic characteristics. They compose the unifier with the classical term unification. Nonetheless, the prefix unification changes based on the modal logic wanted, turning on or off the unification between a constant prefix p and a word prefix Vu, according to the accessibility relation of such modal logic, for example.

PolyCoP and both provers are, in a sense, non-classical provers. However, mleanCoP and ileanCoP represent the relations of modal and intuitionistic logic in prefixes with a modal/intuitionistic unifier. We began the same way but discovered that using prefixes to represent typical instances should badly represent the description logics semantics. For example, to represent that no instance is more typical than a in concept C, we need to express it like $V : \neg C(a)$, where V is a prefix variable and can be unified with any other prefix. If we add the typical relation for pairs of instances, the problem increases, and the representation of such cases becomes non-intuitive.

7.2.2 NanoCoP

The connection methods mentioned above rely on a clausal representation of formulae. However, such representation modifies the original formulae, making returning a more human-readable proof harder. NanoCoP (OTTEN, 2016a) is a non-clausal first-order connection prover built on top of the non-clausal connection calculus (OTTEN, 2011). In addition to benefiting from the techniques mentioned above, it avoids redundant literals created by the clausal form. The significant difference from the previous calculi is that a matrix now comprises clauses, literals and other matrices. The calculus of nanoCoP adds a Decomposition rule to the earlier calculi, which deals with the submatrices and extracts the complementary literals from it. Later, Otten also developed further versions of nanoCoP for modal and intuitionistic logic (OTTEN, 2017; OTTEN, 2016b). PolyCoP is a clausal prover, but implementing a non-clausal PolyCoP is part of our future work agenda.

7.2.3 Raccoon

Raccoon (FILHO; FREITAS; OTTEN, 2017) is an $\mathcal{ALC}$ connection reasoner developed in C++ based on $\mathcal{ALC}$ $\theta - CM$(FREITAS; OTTEN, 2016). The leanCoP family of provers inspires Raccoon with competitive results against highly optimized reasoners. Besides the different DL, the novelty of PolyCoP against Raccoon is the abstraction between the proof search code and the logical representation of knowledge bases and their constructors. In a way, PolyCoP continues Raccoon's saga to implement the full description logics, as $\mathcal{ALCH}^{\bullet}$ is an extension of $\mathcal{ALCH}$, a more expressive description logic than $\mathcal{ALC}$.

7.2.4 Machine learning and connection method

In recent years, CMs combined with machine learning (ML) techniques have given rise to variants of leanCoP in a mixture of programming languages and approaches. rlCoP (KALISZYK et al., 2018) uses reinforcement learning to calculate probabilities of rules/steps application and estimate a heuristic for each proof state. Connect++ (HOLDEN, 2023), a C++ implementation of leanCoP, has as one of its primary goals a compiled program to use with machine learning approaches and a modifiable prover that other researchers can reuse. Connections (RØMMING; OTTEN; HOLDEN, 2023) is a Python library that maps connection calculi to Markov Decision

Procedures, providing a framework for ML approaches. Lastly, Bare Metal Tableaux Prover (KALISZYK, 2015) is a reimplementation of leanCoP in the programming language C. Its primary purpose is to be a low-level efficient connection tableau. So, the prover uses two stacks to store the proof state instead of implementing the calculus in a recursive way like the other provers. The left stack stores the extension rule left branch of the proof, and the right one stores the remaining (still to be proven) extension rule right step. In a sense, such provers aim to simplify access to parameters and proof states of connection provers such as leanCoP to use ML techniques. PolyCoP is built on top of such premises once it provides interfaces and allows reuse of the core proof-related implementation.

7.3 CONCLUDING REMARKS

In this chapter, we discuss the work related to our proposal. As mentioned earlier, the calculus proposed is the first connection method for $\mathcal{ALCH}^{\bullet}$. Furthermore, PolyCoP was the first to implement a reasoner for $\mathcal{ALCH}^{\bullet}$.

One challenge of being a pioneer is that no other studies exist to evaluate and compare results. Indeed, there is no defeasible knowledge base dataset in which we can evaluate reasoners such as PolyCoP. Such a lack of dataset inspired us to build a polymorphic connection prover. Thus, we could test and check the proof-related code correctness in other logic.

8 CONCLUSIONS

In this thesis, we have outlined a connection method for $\mathcal{ALCH}^{\bullet}$, which is a defeasible description logic that is at least as expressive as most other defeasible DLs studied in the existing literature. This calculus extends $\mathcal{ALC}$ $\theta - CM$ in two ways: (*i*) it aligns with the preferential-DL semantics developed by Britz et al. and by Giordano et al., which is widely accepted in the literature on reasoning with defeasible ontologies; and (*ii*) it relies on a custom matrix translation that we introduced to handle typicality in concepts and roles.

Regarding the implementation, we created PolyCoP, a connection prover for $\mathcal{ALCH}^{\bullet}$ and other logics. It is designed to implement a polymorphic version of the connection method, which abstracts the need to code proof-related algorithms. PolyCoP allows users to focus solely on the differences in literal representation, unification, and blocking strategies from each logic. This can help beginners in logic to learn the connection method easily and developers to manipulate our framework and build novel provers.

Furthermore, PolyCoP utilizes a proof tree in its proving algorithm, which can be useful for understanding how the proof occurred. The proof tree can also be visually represented using a LaTeX system.

Our objective is to propose a sound and complete connection method for $\mathcal{ALCH}^{\bullet}$, whose proofs are grounded on description logic proof theory. We have successfully achieved this objective. In Chapter 5, we presented a decision procedure based on the connection method, and demonstrations relied solely on the logic presented. Furthermore, in Chapter 6, we introduced PolyCoP as a reasoner for $\mathcal{ALCH}^{\bullet}$, which can also be used as a foundation for reasoning in virtually any other logic.

Below, we provide this thesis's limitations and future work.

8.1 PUBLICATIONS

The work detailed in this thesis led to the development of three new papers, which we submitted to international journals and conferences. We list the publications as follows:

1. **A connection method for a defeasible extension of** $\mathcal{ALC}$, presented in the Proceedings of the 34th International Workshop on Description Logics (DL 2021);

2. **Connection method for defeasible extension of $\mathcal{ALCH}$**, publication under review in the Journal of Automated Reasoning; and

3. **PolyCoP: a connection method for (possibly) any logical language**, an accepted paper for the Rule Challenge track of the 8th International Joint Conference on Rules and Reasoning (RuleML+RR 2024).

8.2 LIMITATIONS

- **Preferential entailment:** The CM in this work applies preferential reasoning, a Tarskian notion of logical consequence, and, therefore, monotonic. However, such reasoning is not always sufficient for defeasible reasoning with exceptions. Stronger, more venturous forms of entailment are often called for, such as the rational entailment (BRITZ et al., 2021; CASINI; STRACCIA, 2010; GIORDANO et al., 2015).

- **Single preference ordering:** We use a description logic language with only one preference ordering. Hence, it is impossible to represent scenarios where an object is more typical for one context but less typical for another. Therefore, a more expressive language representing a multi-preference of objects could solve this problem.

- **Optimizations:** PolyCoP is a polymorphic implementation designed to meet different logics and serve as a basis for new researchers or students. Even so, optimizations already established and used by other connection provers, such as Lemmata and Restricted backtracking, can be added.

- **Experimental results:** So far, our implementation is the first reasoner for $\mathcal{ALCH}^{\bullet}$. There are few existing knowledge bases for this logic, and there is no syntactic formalization of ontologies, as we see with OWL for classical DL. Therefore, conducting experiments and further analyzing PolyCoP for this logic is complex.

8.3 FUTURE WORK

- **Rational entailment:** The rational closure of a defeasible ontology, has been extensively studied in the context of defeasible $\mathcal{ALC}$. Therefore, enhancing the current CM with rational entailment is a promising advancement in our work.

- **Multiple preference relations:** Another addition to our work is conceiving multi-preferential relations to the CM. Thus, the notions of a typical object in one context and non-typical in others would be satisfied.

- **Non-clausal connection method:** Non-clausal provers, like Nano-cop (OTTEN, 2016a), have proven to be very effective for reasoning, especially in memory optimization, considering the number of literals presented in their clausal counterpart. Additionally, Description Logics (DLs) have an appeal for this representation, as their original axioms can be directly translated into a non-clausal matrix. So, we plan to implement a non-clausal version in PolyCoP.

- **Optimizations:** We have not yet incorporated several optimizations already used by other connection provers and specific optimizations for defeasible logic. For example, we could apply a reduction when the typicality relation in an active path infers a literal that exists in the objective clause. We intend to include these optimizations in the method and PolyCoP.

- **PolyCoP for other logic:** We intend for PolyCoP to serve as a connection method for all kinds of logic as possible. To achieve this, we will implement various logics, verify the patterns, and thus establish it as a universal reasoner.

REFERENCES

ANTONIOU, G.; GROTH, P.; HARMELEN, F. v. v.; HOEKSTRA, R. *A Semantic Web Primer*. [S.l.]: The MIT Press, 2012. ISBN 0262018284, 9780262018289.

BAADER, F.; HORROCKS, I.; LUTZ, C.; SATTLER, U. A basic description logic. In: _____. *An Introduction to Description Logic*. [S.l.]: Cambridge University Press, 2017. p. 10–49.

BAADER, F.; NUTT, W. Basic Description Logics. In: BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. (Ed.). *The Description Logic Handbook: Theory, implementation and applications*. [S.l.]: Cambridge University Press, 2007.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. *Sci. Am.*, v. 284, n. 5, p. 34–43, 2001.

BIBEL, W. *Automated Theorem Proving*. Wiesbaden: Vieweg Verlag, 1987.

BIBEL, W. A vision for automated deduction rooted in the connection method. In: SCHMIDT, R. A.; NALON, C. (Ed.). *Automated Reasoning with Analytic Tableaux and Related Methods - 26th International Conference, TABLEAUX 2017, Brasília, Brazil, September 25-28, 2017, Proceedings*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10501), p. 3–21.

BONATTI, P. Rational closure for all description logics. *Artificial Intelligence*, v. 274, p. 197–223, 2019.

BONATTI, P.; FAELLA, M.; PETROVA, I.; SAURO, L. A new semantics for overriding in description logics. *Artificial Intelligence*, v. 222, p. 1–48, 2015.

BONATTI, P.; FAELLA, M.; SAURO, L. Defeasible inclusions in low-complexity DLs. *Journal of Artificial Intelligence Research*, v. 42, p. 719–764, 2011.

BONATTI, P.; SAURO, L. On the logical properties of the nonmonotonic description logic DL^N. *Artificial Intelligence*, v. 248, p. 85–111, 2017.

BOZZATO, L.; EITER, T.; SERAFINI, L. Enhancing context knowledge repositories with justifiable exceptions. *Artificial Intelligence*, v. 257, p. 72–126, 2018.

BRITZ, K.; CASINI, G.; MEYER, T.; MOODLEY, K.; SATTLER, U.; VARZINCZAK, I. Principles of KLM-style defeasible description logics. *ACM Transactions on Computational Logic*, v. 22, n. 1, 2021.

BRITZ, K.; HEIDEMA, J.; MEYER, T. Semantic preferential subsumption. In: LANG, J.; BREWKA, G. (Ed.). *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR)*. [S.l.]: AAAI Press/MIT Press, 2008. p. 476–484.

BRITZ, K.; VARZINCZAK, I. Context-based defeasible subsumption for $d\mathcal{SROIQ}$. In: *Proceedings of the 13th International Symposium on Logical Formalizations of Commonsense Reasoning*. [S.l.: s.n.], 2017.

BRITZ, K.; VARZINCZAK, I. Toward defeasible $\mathcal{SROIQ}$. In: *Proceedings of the 30th International Workshop on Description Logics*. [S.l.: s.n.], 2017.

BRITZ, K.; VARZINCZAK, I. Preferential tableaux for contextual defeasible $\mathcal{ALC}$. In: CERRITO, S.; POPESCU, A. (Ed.). *Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX)*. [S.l.]: Springer, 2019. (LNCS, 11714), p. 39–57.

CAI, S.; MING, Z.; LI, S. BALC: A belief extension of description logic ALC. In: *9th International Conference for Young Computer Scientists*. Hunan, China: IEEE, 2008. p. 1711–1716.

CASINI, G.; MEYER, T.; MOODLEY, K.; NORTJÉ, R. Relevant closure: A new form of defeasible reasoning for description logics. In: FERMÉ, E.; LEITE, J. (Ed.). *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA)*. [S.l.]: Springer, 2014. (LNCS, 8761), p. 92–106.

CASINI, G.; MEYER, T.; MOODLEY, K.; SATTLER, U.; VARZINCZAK, I. Introducing defeasibility into OWL ontologies. In: ARENAS, M.; CORCHO, O.; SIMPERL, E.; STROHMAIER, M.; D'AQUIN, M.; SRINIVAS, K.; GROTH, P.; DUMONTIER, M.; HEFLIN, J.; THIRUNARAYAN, K.; STAAB, S. (Ed.). *Proceedings of the 14th International Semantic Web Conference (ISWC)*. [S.l.]: Springer, 2015. (LNCS, 9367), p. 409–426.

CASINI, G.; STRACCIA, U. Rational closure for defeasible description logics. In: JANHUNEN, T.; NIEMELÄ, I. (Ed.). *Proceedings of the 12th European Conference on Logics in Artificial Intelligence (JELIA)*. [S.l.]: Springer-Verlag, 2010. (LNCS, 6341), p. 77–90.

CASINI, G.; STRACCIA, U. Defeasible inheritance-based description logics. *Journal of Artificial Intelligence Research (JAIR)*, v. 48, p. 415–473, 2013.

CASINI, G.; STRACCIA, U.; MEYER, T. A polynomial time subsumption algorithm for nominal safe $\mathcal{ELO}_{\bot}$ under rational closure. *Information Sciences*, v. 501, p. 588–620, 2019.

FERNANDES, R.; FREITAS, F.; VARZINCZAK, I. A connection method for a defeasible extension of ALC. In: HOMOLA, M.; RYZHIKOV, V.; SCHMIDT, R. A. (Ed.). *Proceedings of the 34th International Workshop on Description Logics (DL 2021) part of Bratislava Knowledge September (BAKS 2021), Bratislava, Slovakia, September 19th to 22nd, 2021*. [S.l.]: CEUR-WS.org, 2021. (CEUR Workshop Proceedings, v. 2954).

FILHO, D. M.; FREITAS, F.; OTTEN, J. RACCOON: A Connection Reasoner for the Description Logic $\mathcal{ALC}$. In: *21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*. Maun, Botswana: EasyChair, 2017. v. 46, p. 200–211.

FREITAS, F.; OTTEN, J. A Connection Calculus for the Description Logic $\mathcal{ALC}$. In: *29th Canadian Conference on Artificial Intelligence*. Victoria, BC, Canadá: Springer, 2016. p. 243–256.

FREITAS, F.; VARZINCZAK, I. Cardinality restrictions within description logic connection calculi. In: BENZMÜLLER, C.; RICCA, F.; PARENT, X.; ROMAN, D. (Ed.). *Rules and Reasoning - Second International Joint Conference, RuleML+RR 2018, Luxembourg, September 18-21, 2018, Proceedings*. [S.l.]: Springer, 2018. (Lecture Notes in Computer Science, v. 11092), p. 65–80.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Pearson Education, 1994. (Addison-Wesley Professional Computing Series). ISBN 9780321700698.

GIORDANO, L.; GLIOZZI, V.; OLIVETTI, N.; POZZATO, G. Preferential description logics. In: DERSHOWITZ, N.; VORONKOV, A. (Ed.). *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*. [S.l.]: Springer, 2007. (LNAI, 4790), p. 257–272.

GIORDANO, L.; GLIOZZI, V.; OLIVETTI, N.; POZZATO, G. $\mathcal{ALC} + T$: a preferential extension of description logics. *Fundamenta Informaticae*, v. 96, n. 3, p. 341–372, 2009.

GIORDANO, L.; GLIOZZI, V.; OLIVETTI, N.; POZZATO, G. A non-monotonic description logic for reasoning about typicality. *Artificial Intelligence*, v. 195, p. 165–202, 2013.

GIORDANO, L.; GLIOZZI, V.; OLIVETTI, N.; POZZATO, G. Semantic characterization of rational closure: From propositional logic to description logics. *Artificial Intelligence*, v. 226, p. 1–33, 2015.

HOLDEN, S. B. Connect++: A new automated theorem prover based on the connection calculus. In: OTTEN, J.; BIBEL, W. (Ed.). *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi (AReCCa 2023) affiliated with the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2023), Prague, Czech Republic, September 18, 2023*. [S.l.]: CEUR-WS.org, 2023. (CEUR Workshop Proceedings, v. 3613), p. 95–106.

HORRIDGE, M.; BECHHOFER, S. The OWL API: A java API for OWL ontologies. *Semantic Web*, v. 2, n. 1, p. 11–21, 2011. Disponível em: <https://doi.org/10.3233/SW-2011-0025>.

KALISZYK, C. Efficient low-level connection tableaux. In: NIVELLE, H. de (Ed.). *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*. [S.l.]: Springer, 2015. (Lecture Notes in Computer Science, v. 9323), p. 102–111.

KALISZYK, C.; URBAN, J.; MICHALEWSKI, H.; OLSÁK, M. Reinforcement learning of theorem proving. *CoRR*, abs/1805.07563, 2018.

OTTEN, J. leancop 2.0and ileancop 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: ARMANDO, A.; BAUMGARTNER, P.; DOWEK, G. (Ed.). *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*. Springer, 2008. (Lecture Notes in Computer Science, v. 5195), p. 283–291. Disponível em: <https://doi.org/10.1007/978-3-540-71070-7_23>.

OTTEN, J. Restricting backtracking in connection calculi. *AI Commun.*, v. 23, n. 2-3, p. 159–182, 2010.

OTTEN, J. A non-clausal connection calculus. In: BRÜNNLER, K.; METCALFE, G. (Ed.). *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings*. [S.l.]: Springer, 2011. (Lecture Notes in Computer Science, v. 6793), p. 226–241.

OTTEN, J. Implementing connection calculi for first-order modal logics. In: KOROVIN, K.; SCHULZ, S.; TERNOVSKA, E. (Ed.). *IWIL 2012: The 9th International Workshop on the Implementation of Logics, Merida, Venezuela, March 10, 2012*. [S.l.]: EasyChair, 2012. (EPiC Series in Computing, v. 22), p. 18–32.

OTTEN, J. Mleancop: A connection prover for first-order modal logic. In: DEMRI, S.; KAPUR, D.; WEIDENBACH, C. (Ed.). *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*. Springer, 2014. (Lecture Notes in Computer Science, v. 8562), p. 269–276. Disponível em: <https://doi.org/10.1007/978-3-319-08587-6_20>.

OTTEN, J. nanocop: A non-clausal connection prover. In: OLIVETTI, N.; TIWARI, A. (Ed.). *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*. Springer, 2016. (Lecture Notes in Computer Science, v. 9706), p. 302–312. Disponível em: <https://doi.org/10.1007/978-3-319-40229-1_21>.

OTTEN, J. Non-clausal connection-based theorem proving in intuitionistic first-order logic. In: BENZMÜLLER, C.; OTTEN, J. (Ed.). *Proceedings of the 2nd International Workshop Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2016) affiliated with the International Joint Conference on Automated Reasoning (IJCAR 2016)., Coimbra, Portugal, July 1, 2016*. [S.l.]: CEUR-WS.org, 2016. (CEUR Workshop Proceedings, v. 1770), p. 9–20.

OTTEN, J. Non-clausal connection calculi for non-classical logics. In: SCHMIDT, R. A.; NALON, C. (Ed.). *Automated Reasoning with Analytic Tableaux and Related Methods - 26th International Conference, TABLEAUX 2017, Brasília, Brazil, September 25-28, 2017, Proceedings*. [S.l.]: Springer, 2017. (Lecture Notes in Computer Science, v. 10501), p. 209–227.

OTTEN, J. Advancing automated theorem proving for the modal logics D and S5. In: BENZMÜLLER, C.; OTTEN, J. (Ed.). *Proceedings of the 4th International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2022) affiliated with the 11th International Joint Conference on Automated Reasoning (IJCAR 2022), Haifa, Israel, August 11, 2022*. [S.l.]: CEUR-WS.org, 2022. (CEUR Workshop Proceedings, v. 3326), p. 81–91.

OTTEN, J. 20 years of leancop - an overview of the provers. In: OTTEN, J.; BIBEL, W. (Ed.). *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi (AReCCa 2023) affiliated with the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2023), Prague, Czech Republic, September 18, 2023*. [S.l.]: CEUR-WS.org, 2023. (CEUR Workshop Proceedings, v. 3613), p. 4–22.

OTTEN, J.; BIBEL, W. leancop: lean connection-based theorem proving. *J. Symb. Comput.*, v. 36, n. 1-2, p. 139–161, 2003.

PENSEL, M.; TURHAN, A.-Y. Reasoning in the defeasible description logic $\mathcal{EL}_\perp$ – computing standard inferences under rational and relevant semantics. *International Journal of Approximate Reasoning*, v. 112, p. 28–70, 2018.

RØMMING, F.; OTTEN, J.; HOLDEN, S. B. Connections: Markov decision processes for classical, intuitionistic and modal connection calculi. In: OTTEN, J.; BIBEL, W. (Ed.). *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi (AReCCa 2023) affiliated with the 32nd International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2023), Prague, Czech Republic, September 18, 2023*. [S.l.]: CEUR-WS.org, 2023. (CEUR Workshop Proceedings, v. 3613), p. 107–118.

VARZINCZAK, I. A note on a description logic of concept and role typicality for defeasible reasoning over ontologies. *Logica Universalis*, v. 12, n. 3-4, p. 297–325, 2018.

APPENDIX A – PROOF OF THEOREM 5.1

Definition A.1 (Signature)**.** Let $\mathcal{K}$ be a knowledge base. We denote the *signature* of $\mathcal{K}$, or $sig(\mathcal{K})$ as the list $(\mathrm{C}, \mathrm{R}, \mathrm{I})$, where C, R, and I are the set of concepts, roles, and individuals present in $\mathcal{K}$, respectively. ■

Definition A.2 (Conservative extension)**.** Let $\mathcal{K}$ and $\mathcal{K}'$ be knowledge bases. We say that $\mathcal{K}'$ is a *conservative extension* of $\mathcal{K}$ if

i. $sig(\mathcal{K}) \subseteq sig(\mathcal{K}')$,

ii. every model of $\mathcal{K}'$ is a model of $\mathcal{K}$, and

iii. for each model $\mathcal{I}$ of $\mathcal{K}$ there exists a model $\mathcal{I}'$ of $\mathcal{K}'$ such that $A^{\mathcal{I}} = A^{\mathcal{I}'}$, $r^{\mathcal{I}} = r^{\mathcal{I}'}$, and $a^{\mathcal{I}} = a^{\mathcal{I}'}$, for all concept names A, role names r, and individual names a from $sig(\mathcal{K})$, respectively.

■

Thus, a conservative extension $\mathcal{K}'$ entails nothing but what $\mathcal{K}$ already does w.r.t. $sig(\mathcal{K})$. Thereby, we can replace the knowledge bases with more useful conservative extensions in a normal form.

Apply, in Table 6 is a helper function that maps non-TNF axioms in TBox to a set of axioms closer to the BTNF. This way, a rule applications chain exists that translates α to a set of axioms in BTNF.

Lemma A.1. *Given a TBox axiom α that is not in TNF and a interpretation $\mathcal{I}$, if $\mathcal{I} \Vdash$ apply(α), then $\mathcal{I} \Vdash \alpha$.*

Proof. We prove this lemma by contrapositive. Assume $\mathcal{I} \not\Vdash \alpha$. Then:

- if $\alpha = (D \equiv E)$, then $D^{\mathcal{I}} \neq E^{\mathcal{I}}$. Hence, either $D^{\mathcal{I}} \not\subseteq E^{\mathcal{I}}$, or $E^{\mathcal{I}} \not\subseteq D^{\mathcal{I}}$. Therefore, $\mathcal{I} \not\Vdash$ *apply*$(D \equiv E)$;

- if $\alpha = (D \sqsubseteq \bullet E)$, then $D^{\mathcal{I}} \not\subseteq \mathit{min}_{<^{\mathcal{I}}} E^{\mathcal{I}}$. Hence, for every concept name N such that $N^{\mathcal{I}} = E^{\mathcal{I}}$, $D^{\mathcal{I}} \not\subseteq \mathit{min}_{<^{\mathcal{I}}} N^{\mathcal{I}}$. Therefore, $\mathcal{I} \not\Vdash$ *apply*$(D \sqsubseteq \bullet E)$;

- if $\alpha = (\neg\bullet A \sqsubseteq D)$, then $(\neg\bullet A)^{\mathcal{I}} \not\subseteq D$. Hence, for every concept name N such that $N^{\mathcal{I}} = D^{\mathcal{I}}$, $(\neg\bullet A)^{\mathcal{I}} \not\subseteq N$. Therefore, $\mathcal{I} \not\Vdash$ *apply*$(\neg\bullet A \sqsubseteq D)$;

Table 6 – Conservative transformation rules. Let D, E, and F be concepts; A be a concept name; N a new concept name; F be a non-literal concept; $\check{D}$ be a pure disjunction; $\hat{E}$ be a pure conjunction; and $\star \in \{\bullet, \neg\bullet\}$.

Rule	α	$apply(\alpha)$
1	$D \equiv E$	$\{D \sqsubseteq E, E \sqsubseteq D\}$
2	$D \sqsubseteq \bigsqcap_{i=1}^{n} E_i$	$\{D \sqsubseteq E_1, \cdots, D \sqsubseteq E_n\}$
3	$D \sqsubseteq \bigsqcup_{i=1}^{n} E_i \sqcup \forall r.F$	$\{D \sqsubseteq \bigsqcup_{i=1}^{n} E_i \sqcup \forall r.N \ , \ N \sqsubseteq F, F \sqsubseteq N\}$
4	$D \sqsubseteq \bigsqcup_{i=1}^{n} F_i \sqcup \hat{E}$	$\{D \sqsubseteq \bigsqcup_{i=1}^{n} F_i \sqcup N \ , \ N \sqsubseteq \hat{E}, \hat{E} \sqsubseteq N\}$
5	$D \sqsubseteq \exists r.F$	$\{D \sqsubseteq \exists r.N \ , \ N \sqsubseteq F, F \sqsubseteq N\}$
6	$D \sqsubseteq \exists r.A$	$\{N \sqsubseteq \exists r.A \ , \ N \sqsubseteq D, D \sqsubseteq N\}$
7	$D \sqsubseteq \star E$	$\{D \sqsubseteq \star N, N \sqsubseteq E, E \sqsubseteq N\}$
8	$D \sqsubseteq \star A$	$\{B \sqsubseteq \star A, B \sqsubseteq D, D \sqsubseteq B\}$
9	$\bigsqcup_{i=1}^{n} D_i \sqsubseteq E$	$\{D_1 \sqsubseteq E, \cdots, D_n \sqsubseteq E\}$
10	$\bigsqcap_{i=1}^{n} F_i \sqcap \check{D} \sqsubseteq E$	$\{\bigsqcap_{i=1}^{n} F_i \sqcap N \sqsubseteq E \ , \ \check{D} \sqsubseteq N, N \sqsubseteq \check{D}\}$
11	$\bigsqcap_{i=1}^{n} D_i \sqcap \exists r.F \sqsubseteq E$	$\{\bigsqcap_{i=1}^{n} D_i \sqcap \exists r.N \sqsubseteq E \ , \ F \sqsubseteq N, N \sqsubseteq F\}$
12	$\forall r.F \sqsubseteq E$	$\{\forall r.N \sqsubseteq E, F \sqsubseteq N, N \sqsubseteq F\}$
13	$\forall r.A \sqsubseteq E$	$\{\forall r.A \sqsubseteq N, E \sqsubseteq N, N \sqsubseteq E\}$
14	$\star D \sqsubseteq E$	$\{\star N \sqsubseteq E, N \sqsubseteq D, D \sqsubseteq N\}$
15	$\star A \sqsubseteq D$	$\{\star A \sqsubseteq B, B \sqsubseteq D, D \sqsubseteq B\}$

The remaining cases of Table 6 are left to the reader. □

As a result of the previous lemma, we have the following corollary:

Corollary A.1. Given a TBox axiom α not in TNF and an interpretation $\mathcal{I}$. If $\mathcal{I} \Vdash \alpha$, then there exists an interpretation $\mathcal{I}'$ such that $\mathcal{I}' \Vdash$ *apply*(α), where $A^{\mathcal{I}} = A^{\mathcal{I}'}$, $r^{\mathcal{I}} = r^{\mathcal{I}'}$, and $a^{\mathcal{I}} = a^{\mathcal{I}'}$, for all concept names A, role names r, and individual names a from $sig(\{\alpha\})$, respectively.

To illustrate that arbitrary knowledge bases can be translated to BTNF ones, Table 6 provides a set of transformation rules that translate a given axiom to a set of axioms closest to BTNF. The Algorithm 5 shows a decision procedure *btnf* that translates knowledge bases to BTNF ones.

Lemma A.2. *Given a knowledge base $\mathcal{K}$, btnf$(\mathcal{K})$ is a conservative extension of $\mathcal{K}$.*

Proof. We prove this proposition by contradiction. Assume *btnf*$(\mathcal{K})$ is not a conservative extension of $\mathcal{K}$. Then, it does not hold some condition of Definition A.2.

Algorithm 5 btnf

Require: An $\mathcal{ALCH}^\bullet$ knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$
Ensure: An $\mathcal{ALCH}^\bullet$ knowledge base $\mathcal{K}' = (\mathcal{T}', \mathcal{R}', \mathcal{A}')$ in BTNF

$\mathcal{T}', \mathcal{R}', \mathcal{A}' \leftarrow \emptyset$
while $\mathcal{A} \neq \emptyset$ **do**
  get some $\alpha \in \mathcal{A}$
  $\mathcal{A} = \mathcal{A} \setminus \{\alpha\}$
  if $\alpha = D(a)$ and D is a non-literal concept **then**
    $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{N(a)\}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{N \equiv D\}$
  else if $\alpha = R(a, b)$ and R is a non-literal role **then**
    $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{u(a, b)\}$
    $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{u \sqsubseteq R\}$
  else
    $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{\alpha\}$
  end if
end while
while $\mathcal{R} \neq \emptyset$ **do**
  get some $R \sqsubseteq S \in \mathcal{R}$
  $\mathcal{R} = \mathcal{R} \setminus \{R \sqsubseteq S\}$
  if $R = \bullet r$ and $S = \bullet s$ **then**
    $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{R \sqsubseteq N, N \sqsubseteq S\}$
  else
    $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{R \sqsubseteq S\}$
  end if
end while
while $\mathcal{T} \neq \emptyset$ **do**
  get some $\alpha \in \mathcal{T}$
  $\mathcal{T} = \mathcal{T} \setminus \{\alpha\}$
  if α is in BTNF **then**
    $\mathcal{T}' \leftarrow \mathcal{T}' \cup \{\alpha\}$
  else
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathsf{apply}(\alpha)$
  end if
end while
return $(\mathcal{T}', \mathcal{R}', \mathcal{A}')$

(i) $sig(\mathcal{K}) \subseteq sig(\textit{btnf}(\mathcal{K}))$. Once the algorithm only adds new concept names and role names to the knowledge base without removing any, then it holds that $sig(\mathcal{K}) \subseteq sig(\textit{btnf}(\mathcal{K}))$.

(ii) every model of *btnf*$(\mathcal{K})$ is a model of $\mathcal{K}$. We carefully check the lines that create new axioms to prove this condition. In the lines 6 and 7, the algorithm adds $\{N(a), N \equiv D\}$ when the ABox axiom contains a non-literal concept. A model that satisfies both also satisfies $D(a)$. In the lines 9 and 10, the algorithm adds $\{u(a), u \sqsubseteq D\}$ when the ABox axiom contains a non-literal role. If $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in u^{\mathcal{I}}$ and $u^{\mathcal{I}} \subseteq R^{\mathcal{I}}$, then $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, for every interpretation $\mathcal{I}$. Hence, a model that holds $\{u(a), u \sqsubseteq D\}$ also holds that $R(a, b)$. In the line 12, the algorithm adds the same assertion to the new ABox. Hence, the models trivially hold it. For the RBox, in the line 19, the algorithm adds $\{R \sqsubseteq N, N \sqsubseteq S\}$ when both sides of the subsumption contain a typical operator. By the semantics, $min_{\ll^{\mathcal{I}}} r^{\mathcal{I}} \subseteq N^{\mathcal{I}}$ and $N^{\mathcal{I}} \subseteq min_{\ll^{\mathcal{I}}} s^{\mathcal{I}}$, for every model $\mathcal{I}$. Hence, $min_{\ll^{\mathcal{I}}} r^{\mathcal{I}} \subseteq min_{\ll^{\mathcal{I}}} s^{\mathcal{I}}$. The algorithm adds the same assertion in the line 21. For the TBox, in the line 28, the algorithm adds the same axiom if it is in BTNF. In the line 30, by the Lemma A.1, we have that every model $\mathcal{I}$ that satisfies *apply*(α) also satisfies α.

(iii) for each model $\mathcal{I}$ of $\mathcal{K}$ there exists a model $\mathcal{I}'$ of *btnf*$(\mathcal{K})$ such that $A^{\mathcal{I}} = A^{\mathcal{I}'}$, $r^{\mathcal{I}} = r^{\mathcal{I}'}$, and $a^{\mathcal{I}} = a^{\mathcal{I}'}$, for all concept names A, role names r, and individual names a from $sig(\mathcal{K})$, respectively. Lastly, this condition also holds once the addition of new axioms preserves the models, if we consider *sig($\mathcal{K}$)*, only.

Therefore, all the conditions have been met, confirming that *btnf*$(\mathcal{K})$ serves as a conservative extension of $\mathcal{K}$. □

Lemma A.3. *Given a knowledge base $\mathcal{K}$, btnf$(\mathcal{K})$ is in BTNF.*

Proof. The loops for ABox (lines 2-14), RBox (lines 15-23), and TBox (lines 24-32) add to the new ABox, RBox, and TBox, respectively, only axioms in BTNF. Therefore, the output of *btnf* $(\mathcal{K})$ is a knowledge base in BTNF. □

The next step is to prove the algorithm terminates. The point of attention here is the transformations in the TBox. Ensure that these rules are applied consistently. Any sequence of applying the rules in the table will lead to the knowledge base being in BTNF.

To achieve this, we will define a measure of impurity for an axiom (as well as for a concept) below.

Definition A.3 (Number of impurities of a concept)**.** Given a concept (possibly complex) D, the measure $m(D)$ of D is defined as the *number of impure operators* in D. ■

Example A.1 (Number of impurities of a concept)**.** Let A, B, C be concept names; r be a role name; and $D = \bullet(A \sqcap \forall r.\neg(B \sqcap C))$ be a concept. The number of impurities $m(D)$ of D is *3*, once:

- the typicality operator is not attached to a concept name;
- the conjunction has a $\forall$ operator as its operand; and
- the filler of the operator $\forall$ is not a concept name.

△

Definition A.4 (Number of impurities of an axiom)**.** Given two concept names D, E, and a TBox axiom α. We define the measure $m(\alpha)$ as:

- $m(D \equiv E) \stackrel{\text{def}}{=} 2 + m(D) + m(E)$;
- $m(D \sqsubseteq E) \stackrel{\text{def}}{=} 1 + m(D) + m(E)$ if:
  1. D root operator is in $\{\forall, \exists, \bullet, \neg\bullet\}$ and E is a complex concept or
  2. D is a complex concept and E root operator is in $\{\forall, \exists, \bullet, \neg\bullet\}$; and
- $m(D \sqsubseteq E) \stackrel{\text{def}}{=} m(D) + m(E)$, otherwise.

■

Example A.2 (Number of impurities of an axiom)**.** Let A_1, A_2, A_3, be concept names; r be a role name; and $\alpha = \exists r.A_1 \sqsubseteq \bullet(A_2 \sqcap \forall r.\neg(A_3 \sqcap A_1))$ be an axiom. The number of impurities $m(\alpha)$ of α is *4*, once:

- the typicality operator is not attached to a concept name;
- the conjunction has a $\forall$ operator as its operand;
- the filler of the operator $\forall$ is not a concept name; and
- the root operator of the left side of the axiom is a $\exists$, and the right side is a complex concept.

△

Corollary A.2. Given a TBox axiom α, α is in BTNF iff $m(\alpha) = 0$.

Rule	$m(\alpha)$	$m(\beta)$, for all $\beta \in apply(\alpha)$
1	$2 + m(D) + m(E)$	$\{1 + m(D) + m(E), 1 + m(E) + m(D)\}$
2	$1 + m(D) + \sum_{i=1}^{n} m(E_i)$	$\{m(D) + m(E_1), \cdots, m(D) + m(E_n)\}$
3	$m(D) + \sum_{i=1}^{n} m(E_i) + 1 + m(F)$	$\{m(D) + \sum_{i=1}^{n} m(E_i), m(F), m(F)\}$
4	$m(D) + \sum_{i=1}^{n} m(F_i) + 1 + m(\hat{E})$	$\{m(D) + \sum_{i=1}^{n} m(F_i), m(\hat{E}), m(\hat{E})\}$
5	$1 + m(D) + m(F)$	$\{1 + m(D), m(F), m(F)\}$
6	$1 + m(D)$	$\{0, m(D), m(D)\}$
7	$1 + m(D) + m(E)$	$\{1 + m(D), m(E), m(E)\}$
8	$1 + m(D)$	$\{0, m(D), m(D)\}$
9	$\sum_{i=1}^{n} m(D_i) + m(E)$	$\{m(D_1) + m(E), \cdots, m(D_n) + m(E)\}$
10	$\sum_{i=1}^{n} m(F_i) + 1 + m(\check{D}) + m(E)$	$\{\sum_{i=1}^{n} m(F_i) + m(E), m(\check{D}), m(\check{D})\}$
11	$\sum_{i=1}^{n} m(F_i) + 1 + m(F) + m(E)$	$\{\sum_{i=1}^{n} m(F_i) + m(E), m(F), m(F)\}$
12	$1 + m(F) + m(E)$	$\{1 + m(E), m(F), m(F)\}$
13	$1 + m(E)$	$\{0, m(E), m(E)\}$
14	$1 + m(D) + m(E)$	$\{1 + m(E), m(D), m(D)\}$
15	$1 + m(D)$	$\{0, m(D), m(D)\}$

Table 7 – Number of impurities of each rule transformation.

Following the above definitions, we can now state that the rule set of Table 6 is terminating.

Lemma A.4 (Axiom transformation rules is terminating)**.** *The set of rules in Table 6 is terminating.*

Proof. We prove that the set of rules is terminating, showing that, for each case α in the table, $m(\alpha) > m(\beta)$, for all $\beta \in apply(\alpha)$. The measures are shown in Table 7.

□

Now, we can state the termination of the algorithm.

Lemma A.5 (Termination of *btnf*)**.** *Given a knowledge base $\mathcal{K}$, btnf($\mathcal{K}$) terminates.*

Proof. Let n_A and n_R be the size of the ABox and RBox of $\mathcal{K}$, respectively. For each iteration, the first two loops (lines 2-14 and 15-23) remove an axiom from ABox and RBox, respectively. So, there are $n_A + n_R$ iterations to end both loops.

In the last loop (lines 24-32), $apply$ adds new axioms to the TBox if α is not yet in BTNF. According to the Lemma A.4, any chain of rule applications is terminating. Therefore, the

algorithm terminates after $n_A + n_R + n_T$ iterations, where n_T is the number of iterations of the last loop. □

Theorem 5.1. Given a knowledge base $\mathcal{K}$, The algorithm *btnf* is a decision procedure to obtain a BTNF conservative extension of $\mathcal{K}$.

Proof. The proof of this theorem is a consequence of Lemmas A.2, A.3 and A.5. □