Optimised tableau algorithms for reasoning in the description logic ALC extended with link keys

Khadija Jradeh

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Informatique
Unité de recherche : Laboratoire d'Informatique de Grenoble

Algorithmes de tableau optimisés pour le raisonnement dans la logique de description ALC étendue avec des clés de liage

Optimised tableau algorithms for reasoning in the description logic ALC extended with link keys

Présentée par :

Khadija JRADEH

Direction de thèse :

Manuel ATENCIA ARCAS Université Grenoble Alpes	Directeur de thèse
Chan LE DUC Université Sorbonne Paris Nord	Co-directeur de thèse

Rapporteurs :

MADALINA CROITORU
Professeur des Universités, UNIVERSITE DE MONTPELLIER
NATHALIE PERNELLE
Professeur des Universités, UNIVERSITE SORBONNE PARIS NORD

Thèse soutenue publiquement le **12 juillet 2022**, devant le jury composé de :

MADALINA CROITORU Professeur des Universités, UNIVERSITE DE MONTPELLIER	Rapporteure
NATHALIE PERNELLE Professeur des Universités, UNIVERSITE SORBONNE PARIS NORD	Rapporteure
JERÔME GENSEL Professeur des Universités, UNIVERSITE GRENOBLE ALPES	Président
CASSIA TROJAHN DOS SANTOS Maître de conférences HDR, UNIVERSITE TOULOUSE 2 - JEAN JAURES	Examinatrice

Invités :

Contents

List of Algorithms

List of Figures

Acknowledgement

I like first to thank the reviewers, Mme Madalina Croitoru and Mme Nathalie Pernelle for their valuable reviews and comments. I would like to thank Mme Madalina Croitoru, also, for the support and the encouragement she gave me during the thesis follow-up meetings. I want to thank Mme Cassia Trojahn Dos Santos for being on the jury and for providing me the opportunity to do my post-doctoral research with her. Also, I am very thankful to the president of the jury M. Jérôme Gensel for all the interesting discussions.

I also like to express my sincere gratitude to my thesis director Manuel Atencia Arcas, for the encouragement, and the pieces of advice he has provided throughout my thesis in both scientific and personal manners. It was a great opportunity to work with you. I would like to thank my co-supervisor, Chan Le Duc, for the huge amount of knowledge he passed to me and for always insisting to bring out the best version of me.

I sincerely thank the mOeX team members for their support and encouragement. Especially, Jérôme Euzenat for his guidance, feedback, and comments which helped me to shape and improve my research. Also, I thank Jérôme David for all the help on both scientific and personal aspects. I also like to thank my colleagues and friends at mOeX, Yasser and Andreas, who made my journey richer and more joyful.

I am deeply grateful to my family and especially to my mother, Samira, and my father, Mahmoud, for their continuous care and love. Thank you for always being beside me no matter the distance that have separated between us. I would never have done it without you.

For my friends, Mayssam, Samarmar, and Zainab thank you for being my second family. I would never find friends as supportive and caring as you.

For my partner, Shadi, thank you for always being there for me. For your patience, support, and love which gave me the strength to carry on my pursuit of the Ph.D. degree.

Resumé

Les graphes de connaissances (KG) sont sans cesse utilisés par différentes organisations pour représenter des entités du monde réel sous la forme d'un graphe. Ils peuvent utiliser une couche ontologique pour décrire les classes et les propriétés des entités représentées. Les graphes de connaissances RDF sont des graphes de connaissances qui transmettent au modèle RDF. L'interconnexion des graphes de connaissances RDF consiste à identifier différents IRIs appartenant à différents graphes de connaissances RDF et faisant référence à la même entité du monde réel. Cela facilite l'intégration et l'interopérabilité des données en combinant différentes descriptions d'entités présentes dans différents graphes de connaissances.

Il existe différentes méthodes pour aborder la tâche d'interconnexion des graphes de connaissances RDF. Les clés de liage font partie de ces méthodes. Elles sont utilisées pour interconnecter des graphes de connaissances RDF décrits à l'aide de différentes ontologies. Les clés de liage spécifient les propriétés à comparer pour décider si deux entités appartenant à des classes différentes et présentes dans des graphes de connaissances différents sont les mêmes.

Les clés de liage peuvent être exprimées sous forme d'axiomes logiques, et il est donc possible de les combiner avec des ontologies et des alignements d'ontologies pour effectuer un raisonnement logique. Dans cette thèse, nous avons pour objectif d'étudier le problème du raisonnement avec des clés de liages. Pour étudier formellement ce problème, nous modélisons les graphes de connaissances RDF, les ontologies et les alignements d'ontologies en utilisant la logique de description $\mathcal{ALC}$. Nous choisissons la logique de description $\mathcal{ALC}$ comme langage de base pour le raisonnement. $\mathcal{ALC}$ couvre de nombreuses capacités de modélisation utilisées pour la représentation des connaissances et permet une extension plus facile à des logiques de description plus expressives. Nous étendons $\mathcal{ALC}$ avec des clés de liage et des égalités individuelles, la logique de description résultante est appelée $\mathcal{ALC}{+}\mathcal{LK}$. Nous montrons que l'implication des clés de liage peut être réduite à la vérification de la cohérence des clés de liage sans avoir besoin d'introduire la négation des clés de liage.

Ensuite, nous concevons un algorithme pour décider de la cohérence de l'ontologie $\mathcal{ALC}{+}\mathcal{LK}$. Nous avons prouvé que l'algorithme est correct, complet et qu'il se termine toujours. Cet algorithme s'exécute en 2EXPTIME. Cependant, il existe des algorithmes EXPTIME pour raisonner en $\mathcal{ALC}$ et les règles de complétion ajoutées pour traiter les clés de liage et les égalités ne nécessitent pas plus de puissance de calcul que celle de $\mathcal{ALC}$.

À la lumière de ce qui précède, nous concevons un algorithme correct, complet et optimal dans le pire des cas pour le raisonnement en $\mathcal{ALC}{+}\mathcal{LK}$. Cet algorithme est inspiré de l'algorithme du tableau comprimé, qui permet d'obtenir le résultat de complexité optimale EXPTIME. Cependant, cet algorithme a un comportement non dirigé qui entrave son implémentation.

Enfin et surtout, nous proposons un algorithme de tableau correct, complet et optimal dans le pire des cas pour le raisonnement dans la logique de description $\mathcal{ALC}$ avec des individus et des clés de liaison. Cet algorithme, contrairement à celui non dirigé, est dirigé par l'application de règles de complétion. Cela évite la génération de structures inutiles et facilite son implémentation. Nous implémentons cet algorithme et fournissons un certain nombre d'expériences de preuve de concept qui démontrent l'importance du raisonnement avec des clés de liage pour la tâche d'interconnexion des données.

Abstract

Knowledge Graphs (KGs) are unceasingly used by different organisation to represent real-world entities in the form of a graph. They may use an ontological layer for describing the classes and properties of the represented entities. RDF knowledge graphs are knowledge graphs that convey to the RDF model. RDF knowledge graph interlinking is the task of identifying different IRIs belonging to different RDF knowledge graphs and referring to the same real-world entity. This facilitates data integration and interoperability by combining different entity descriptions present in different knowledge graphs.

There exist different methods for addressing the task of interlinking RDF knowledge graph. Link keys are among these methods. They are used for interlinking RDF knowledge graphs described using different ontologies. Link keys specify the properties to be compared to decide whether two entities belonging to different classes and present in different knowledge graphs are the same.

Link keys can be expressed as logical axioms, and, thus, it is possible to combine them with ontologies, and ontology alignments to perform logical reasoning. In this thesis, we aim to study the problem of reasoning with link keys. To formally investigate this problem, we model RDF knowledge graphs, ontologies, and ontology alignments using the description logic $\mathcal{ALC}$. We choose the description logic $\mathcal{ALC}$ as a base language for reasoning. $\mathcal{ALC}$ covers many modeling capabilities used for knowledge representation and allows for a more easy extension to more expressive description logics. We extend $\mathcal{ALC}$ with link keys and individual equalities, the resulting description logic is called $\mathcal{ALC}+\mathcal{LK}$. We show that link key entailment can be reduced to link key consistency checking without the need of introducing the negation of link keys.

Then we design an algorithm for deciding the consistency of $\mathcal{ALC}+\mathcal{LK}$ ontology. We have proved that the algorithm is sound, complete, and always terminates. This algorithm runs in 2ExpTime. However, there exist ExpTime algorithms for reasoning in $\mathcal{ALC}$ and the completion rules added for handling link keys and equalities require no more computational power than that of $\mathcal{ALC}$.

In the light of the above, we design a sound, complete, worst-case optimal algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. This algorithm is inspired by the compressed tableau algorithm, which allows obtaining the ExpTime optimal complexity result. However, this algorithm has a non-directed behaviour which obstruct its implementation.

Last but most importantly, we propose a sound, complete, and worst-case optimal tableau algorithm for reasoning in the description logic $\mathcal{ALC}$ with individuals and link keys. This algorithm, in contrast to the non-directed one, is directed by the application of completion rules. This avoids the generation of useless structures and facilitates its implementation. We implement this algorithm and provide a number of proof-of-concept experiments that demonstrates the importance of reasoning with link keys for the data interlinking task.

Chapter 1

Introduction

" . . . when you connect data together, you get power". Tim Berners-Lee

In this chapter, we define RDF graphs interlinking problem and link keys in Section 1.1. We introduce the problem of reasoning with link keys in Section 1.2. We present the contribution of the thesis in Section 1.3. Finally, we give the outline of the thesis in Section 1.4.

1.1 Context

The Semantic Web (SW) is an extension of the World Wide Web (WWW) through standards set by the World Wide Web Consortium (W3C). The Resource Description Framework (RDF) is a W3C standard model for data representation. The RDF Schema (RDFS) and the Ontology Web Language (OWL) are W3C standard languages for representing knowledge about this data. This knowledge adds meaning to the data and allow to easily exchange and combine data belonging to different platforms.

1.1.1 Knowledge representation on the semantic web

A Knowledge Graph (KG) is a representation of structured data in the form of entities and relations between them. This data is represented in a way that machines can read, "understand", and extract facts from. RDF graphs are knowledge graphs that comply with the RDF model. In RDF, data is represented in the form of triples. A triple is formed of a subject, a predicate, and an object. Subjects and predicates are resources, but objects can be either resources or literals. Resources are identified by Internationalised Resources Identifiers (IRIs).

RDF graphs are usually described using ontologies. These ontologies allow for a fluent representation of various types of data descriptions: data schema, taxonomies, and vocabularies. These vocabularies are usually composed of a set of concepts (or classes) that represent the kinds of objects in the domain of knowledge and a set of relations (or properties, roles) that represent the relations between concepts in the domain of knowledge. They are expressed using the standards in the semantic web stack: RDFS and OWL. Moreover, OWL has explicit formal semantics based on Description Logics (DLs).

1.1.2 Interlinking RDF graphs

RDF graphs are often created independently of each other. As a result, they may contain different IRIs that refer to the same real-world entity but are not explicitly related by owl:sameAs property. The owl:sameAs property links two different IRIs that refer to the same real-world entity. Linking these IRIs allows RDF graphs to complement each other. The problem of linking different entities across different graphs is called data interlinking. There have been different methods to address the problem of data interlinking. These methods fall into two main categories: numerical and logical.

Numerical approaches reduce the task of data interlinking into a similarity computation task. While the logical approaches define a set of rules or axioms, which state what makes two entities equal. They can be re-used within a given domain and can be used to profit from logical reasoning.

Numerical approaches calculate the similarity between two different entities that belong respectively to the given pair of RDF graphs. The similarity between two entities is calculated by similarity functions, based on the property values of the given pair of entities. The entities that are similar enough are considered identical and are linked by owl:sameAs property. Some of the numerical approaches like Silk [1] enable user to specifying the conditions that entities must fulfil to be linked. Others, like Limes [2] are fully automatic.

Logical approaches are, in turn, divided into rule and key-based approaches. Rule-based approaches use rules to derive identity links from the input RDF graphs and their ontologies [3, 4, 5].

Key-based approaches aim at extracting a set of keys. Each key consists of a set of properties and a class, the properties allow to uniquely identify an instance that belongs to the specified class. According to this definition, two instances that have the same values for the properties of a key are considered the same. More precisely, a key has the form

$$(\{p_1, ..., p_k\} \ \mathsf{key} \ C)$$

where $p_1, \cdots, p_k$ are properties and C is a class. An example of a key is the following:

$$(\{\mathsf{creator}, \mathsf{title}\} \ \mathsf{key} \ \mathsf{Work}) \tag{1.1}$$

stating that whenever two instances of the class Work share values for role creator and for role title, respectively, then they denote the same entity.

To use key-based approaches to interlink a pair of datasets, candidate keys are first extracted from datasets and then the best candidate keys are selected according to different quality measures [6, 7, 8]. When the RDF graphs are described using the same ontology, keys can be directly used for interlinking these RDF graphs.

But to interlink RDF graphs that are described using different ontologies, keys need to be combined with ontology alignments [9] that relate the properties and classes. Thus, the limitation of using keys for interlinking two RDF graphs is the necessity of having the same ontology describing both graphs, or to have an alignment between their ontologies. Link keys overcome this limitation.

1.1.3 Link keys

Link keys are axioms used to interlink a pair of RDF graphs described using different ontologies [10]. Link keys are reminiscent of EGDs [11]. But EGDs have been proposed in the relational database theory, while link keys are adapted to the specificities of the RDF model and to the task of interlinking RDF graphs described using different ontologies expressed in RDF-S/OWL. Contrary to the relational model, in RDF, properties do not relate objects to literal values only, but also to other objects. Additionally, properties may not be functional.

A link key between two RDF graphs G_1 and G_2 is an expression of the form

$$(\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\} \text{ linkkey } \langle C, D\rangle)$$

where $\langle C, D\rangle$ is a pair of concepts belonging respectively to G_1 and G_2 and $\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle$ is a non-empty sequence of pairs of properties where for each $\langle P_i, Q_i\rangle$ in $\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\}$, P_i belongs to G_1 and Q_i belongs to G_2. It states that if two entities belonging respectively to the concepts C and D share at least one value for each pair of the possibly multivalued properties $\langle P_i, Q_i\rangle$ then they are the same. An example of a link key is:

$$(\{\langle \mathsf{creator}, \mathsf{auteur}\rangle, \langle \mathsf{title}, \mathsf{titre}\rangle\} \text{ linkkey } \langle \mathsf{NonFiction}, \mathsf{Essai}\rangle) \tag{1.2}$$

stating that whenever an instance of the class NonFiction and an instance of the class Essai, share values for roles author and auteur, and for roles title and titre, respectively, they denote the same entity.

This type of link key is called in-link key. There exists another type of link keys called eq-link key that, in contrast in-link key, necessitates that two entities belonging respectively to the concepts C and D must share all the values for each pair of properties included in $\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\}$ to be considered the same [12]. In this thesis, we only consider in-link keys.

Link keys can be constructed by domain experts or automatically extracted from two datasets [10, 13, 14]. Once obtained link keys can be given to a link generating tool such as [3] to generate the set of identity links.

1.2 Problem statement

Ontologies and ontology alignments are logical axioms that constraint the RDF graphs and link keys between these RDF graphs have to be consistent with these constraints as well. Link keys are logical axioms and can be combined with ontologies and ontology alignments to perform logical reasoning. Checking the consistency of link keys can be done by examining wether the satisfaction of a link key violates the logical axioms present in the ontologies or ontology alignments.

Reasoning with link keys is beneficial for the task of data interlinking and can complement link key extraction. Detecting inconsistent link keys reduce the number of identity links established between entities that refer to different real-world objects (false-positive links) and thus they are not relevant for the interlinking task.

Reasoning with link keys allows to check the consistency of simple (Example 1, Link key 1.2) or *complex* link keys (Example 1, Link key 1.8). Complex link keys are link keys composed of complex properties or complex concepts.

The following example illustrates this. Knowledge is modelled in description logics.

Example 1. *Consider two RDF graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ about books. In $\mathcal{G}_1$'s schema the main class is* Work *and contains a subclass* NonFiction. creator *and* title *are a key in this ontology for the* Work *class as described in (1.1). In $\mathcal{G}_2$'s schema , there is a class* Essai *with* auteur, lecteur *and* titre *roles and classes of people such as* Philosophe. *An alignment between $\mathcal{G}_1$'s and $\mathcal{G}_2$'s schemas tells us that an* NonFiction *is more general than a* Essai *which has at least one* Philosophe *as* lecteur *(e.g. reader), that* creator *is equivalent to* auteur *and that* title *is equivalent to* titre. *This can be expressed in description logics as:*

$$\mathsf{NonFiction} \sqsubseteq \mathsf{Work} \tag{1.3}$$

$$\mathsf{title} \equiv \mathsf{titre} \tag{1.4}$$

$$\mathsf{creator} \equiv \mathsf{auteur} \tag{1.5}$$

$$\mathsf{NonFiction} \sqsupseteq \mathsf{Essai} \sqcap \exists \mathsf{lecteur}.\mathsf{Philosophe} \tag{1.6}$$

The key can be expressed as a link key:

$$(\{\langle \mathsf{creator}, \mathsf{creator}\rangle, \langle \mathsf{title}, \mathsf{title}\rangle\} \text{ linkkey } \langle \mathsf{Work}, \mathsf{Work}\rangle) \tag{1.7}$$

The set of statements (1.3–1.7) is sufficient for generating some links. However, for a user, it is not easy to find this out and a program requires a lot of inferences. It is thus useful to find more direct link keys entailed: they will be easier to check by a user and can be directly processed by a link generator. For instance, the link key (1.8) is entailed by (1.3–1.7):

$$\begin{aligned}(\{\langle \mathsf{creator}, \mathsf{auteur}\rangle, \langle \mathsf{title}, \mathsf{titre}\rangle\} \text{ linkkey} \\ \langle \mathsf{NonFiction}, \mathsf{Essai} \sqcap \exists \mathsf{lecteur}.\mathsf{Philosophe}\rangle)\end{aligned} \tag{1.8}$$

though the more simple link key (1.2) is not entailed.

The problem is that there exists no specific reasoning algorithm to do reasoning with link keys.

Reasoning with link keys is challenging. Modeling link keys directly in description logics, the basis of OWL language is not possible. Modeling link keys in description logics requires to impose the equalities on property values which is not possible. Extending description logics with link keys is thus necessary to express link keys. As a result, we cannot use the existing reasoning mechanisms for description logics to do reasoning with link keys. Reasoning with link keys requires to develop a new reasoning mechanism or extend one of the existing algorithms for handling link keys. Moreover, in contrast to checking satisfaction of axioms which requires individual examination, checking the consistency of link keys necessitates to check a set of individuals satisfying the condition of these link keys.

1.3 Contribution

In this thesis, we study the problem of reasoning in the description logic $\mathcal{ALC}$ extended with link keys, which we call $\mathcal{ALC}+\mathcal{LK}$. We denote by an $\mathcal{ALC}+\mathcal{LK}$ ontology, a knowledge base composed of a set of data assertions, vocabularies, and link keys. More precisely, an $\mathcal{ALC}+\mathcal{LK}$ ontology is composed of three components $\langle \mathcal{A}, \mathcal{T}, \mathcal{LK}\rangle$. The ABox $\mathcal{A}$ is composed a set of $\mathcal{ALC}$ concept and role assertions, this ABox represents the triples contained in RDF graphs.

The TBox $\mathcal{T}$ is composed of a set of GCIs. This TBox represents the ontologies of RDF graphs. Finally, the LKBox $\mathcal{LK}$ is composed of a set of $\mathcal{ALC}$ link keys.

We have chosen the description logic $\mathcal{ALC}$ as a base language since it is sufficiently expressive to support many fundamental constructors for web ontologies. It contains many modeling capabilities including concept disjointness, domain and range of roles. It provides a solid basis for extending to more expressive logics allowing to do reasoning with link keys in the future.

We have shown that:

1. Link key entailment can be reduced to consistency checking without the need of introducing the negation of link keys and that reasoning with link keys is decidable. Moreover, we have designed an algorithm for deciding the consistency of $\mathcal{ALC}+\mathcal{LK}$ ontology. This algorithm extends the standard tableau algorithm for reasoning in $\mathcal{ALC}$ with rules dealing with link keys. To ensure its termination, it uses anywhere blocking. It is sound, complete and runs in 2EXPTIME. However, reasoning the description logic $\mathcal{ALC}$ is EXPTIME [15], and the completion rules added for dealing with link keys requires no more expressive reasoning power than that of $\mathcal{ALC}$. These rules do not introduce any new source of complexity besides the one coming from the interaction between the disjunction and existential rules for $\mathcal{ALC}$ (discussed in Section 3.1.1 of Chapter 3). Thus we argue that reasoning in $\mathcal{ALC}+\mathcal{LK}$ can be done in EXPTIME.

2. To achieve the EXPTIME optimal complexity, we adopted the compressed tableau approach. We first designed a *non-directed* algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. The algorithm is called non-directed because it is not oriented by the application of rules. This algorithm is sound, complete, and has an EXPTIME complexity. Since reasoning the description logic $\mathcal{ALC}$ is EXPTIME [15] and this algorithm has an EXPTIME complexity, this algorithm is a worst-case optimal algorithm.

3. After proving that the complexity of reasoning in $\mathcal{ALC}+\mathcal{LK}$ is EXPTIME. We have designed a compressed tableau algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. This algorithm is also sound, complete and has an EXPTIME complexity. However, in contrast to the non-guided algorithm mentioned above, this algorithm applies a set of completion rule. By doing this, it avoids the unnecessary generations done in the non-guided algorithm and only generates what is implied by the completion rules, which makes it practical for implementation.

4. We have implemented the compressed tableau algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. We have carried out a set of experiments which demonstrates the usefulness of reasoning with link keys for the data interlinking task.

Moreover, we have done a preliminary study on the combination of the description logic $\mathcal{ALCI}$ and link keys (Chapter 6). We have defined a finite exponential structure for representing an $\mathcal{ALCI}+\mathcal{LK}$ ontology and proved that we can extract a model from this representation. This representation extends that of an $\mathcal{ALC}+\mathcal{LK}$ ontology used by the compressed tableau algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. This is the first step toward designing a compressed tableau algorithm for $\mathcal{ALCI}+\mathcal{LK}$.

1.4 Organisation

Below we provide an overview of how the thesis is organized, along with the main contributions of each chapter.

- **Chapter 2: Preliminaries**

  We introduce and define the necessary vocabularies and notions required to understand the thesis.

- **Chapter 3: Related Work**

  We present the state-of-the-art algorithms for reasoning with keys and rules in description logics. We also position the work done in this thesis among them.

- **Chapter 4: A tableau algorithm for reasoning in the description logic $\mathcal{ALC}$ with link keys and equalities**

  We introduce the logic $\mathcal{ALC}{+}\mathcal{LK}$, which extends the description logic $\mathcal{ALC}$ with equalities and link keys. We show that link key entailment can be reduced to consistency checking without the need of introducing the negation of link keys. We designed a 2EXPTIME tableau-based algorithm for deciding the consistency of an $\mathcal{ALC}{+}\mathcal{LK}$ ontology. We prove that this algorithm is sound, complete and always terminates.

- **Chapter 5: Reasoning in the description logic $\mathcal{ALC}{+}\mathcal{LK}$ is ExpTime**

  We prove that the consistency checking problem in $\mathcal{ALC}{+}\mathcal{LK}$ has a lower complexity than 2EXPTIME by providing a sound and complete algorithm that runs in EXPTIME. This algorithm is worst-case optimal. It is inspired by the compressed tableau algorithm for the description logic $\mathcal{SHIQ}$ [16]. Similarly to [16], it uses an exponential structure called compressed tableau for representing ontology models, this guarantees that its complexity is EXPTIME.

- **Chapter 6: A worst-case optimal tableau algorithm for the description logic $\mathcal{ALC}$ with individuals and link keys**

  We introduce a worst-case optimal tableau algorithm for reasoning in the description logic $\mathcal{ALC}{+}\mathcal{LK}$. This algorithm reduces the inefficiency of the algorithm described in Chapter 5. To the contrary of the previous algorithm, this algorithm is guided by a set of $\mathcal{ALC}$, link keys and equality completion rules, which reduces dramatically the size of the compressed tableau being constructed.

- **Chapter 7: Implementation and experiments**

  We describe the implementation of StaréLK, the software implementing the optimal tableau algorithm for the description logic $\mathcal{ALC}{+}\mathcal{LK}$ presented in Chapter 6, and explain its architecture. Then we report on a proof-of-concept evaluation that show the usefulness of reasoning with link keys for the task of data interlinking.

- **Chapter 8: Conclusion and future work**

  We provide a summary of the thesis and discuss the different directions for future work.

Chapter 2

Preliminaries

"Syntax is merely a necessary device by which we attach semantics to the representation, and it makes little sense to claim that a representation formalism is semantically more powerful merely because it has more syntactical constructs" [17]

In this chapter, we present the preliminaries required to understand this thesis. We start by giving brief definitions on ontologies and description logics. Then we introduce the description logic $\mathcal{ALC}$. Finally, we explain the standard tableau algorithm for reasoning in the description logic $\mathcal{ALC}$.

2.1 Introduction

Ontologies are used in computer science communities, particularly in Artificial Intelligence (AI), as one of the important knowledge representation formalisms. Ontologies can be defined using different knowledge representation formalisms like glossaries, thesauri, taxonomies, metadata, XML, schemas, and data models. But, more interestingly, ontologies can be represented using a formal language (logic). Description logics are decidable fragments of first-order logic and have been traditionally and extensively used as the formal languages for describing ontologies.

2.2 The description logic $\mathcal{ALC}$ and its extensions

In description logics, the domain of knowledge is described by concept and role descriptions, i.e. expressions that are built from atomic concepts and atomic roles using the constructors provided by the particular description logic. Below we give the *syntax* and *semantics* of the $\mathcal{ALC}$ family of description logics.

2.2.1 Syntax and Semantics of $\mathcal{ALC}$

Every family of description logics is described by the *syntax* and *semantics* of its logical constructors. The syntax is the set of well-formed and meaningful expressions constructed from a set of atomic expressions and a set of rules, while the semantics determines the meanings behind these expressions. The logical constructors in $\mathcal{ALC}$ are concept conjunction, disjunction, negation, and universal and existential quantification.

Definition 1 (Syntax of $\mathcal{ALC}$)**.** *Let* $\mathbf{C}$ *be a set of concept names* (*also called atomic concepts*)*,* $\mathbf{R}$ *be a set of role names* (*also called atomic roles*)*. The set of* $\mathcal{ALC}$ *concepts descriptions over* $\mathbf{C}$ *and* $\mathbf{R}$ *is inductively defined as follows:*

- *every concept name is an* $\mathcal{ALC}$ *description.*
- $\top$ (*the top concept*) *and* $\bot$ (*the bottom concept*) *are* $\mathcal{ALC}$ *descriptions.*
- *Given two concept names* $C, D \in \mathbf{C}$ *and a role name* $R \in \mathbf{R}$*,* $C \sqcap D$ (*conjuction*)*,* $C \sqcup D$ (*disjuction*)*,* $\neg C$ (*negation*)*,* $\forall R.C$ (*universal role quantification*) *and* $\exists R.C$ (*existential role quantification*) *are also* $\mathcal{ALC}$ *descriptions.*

Description Logics separate an ontology into two components, an *assertional* part called the ABox and a *terminological* part called the TBox. The ABox represents knowledge about a concrete situation while the TBox represents knowledge about the structure of the domain. More precisely, an ABox represents knowledge about specific individuals in the domain and the relation between them. An ABox, thus, contains two types of assertions, the first one is called *concept assertions* and the second one is called *role assertions*. The TBox contains a finite set of concept descriptions, called terminological axioms (or simply, axioms). An axiom is an introduction of the names of new concepts and new roles, or an assertion of a subsumption relationship between the concepts.

Definition 2 (ABox)**.** *Let* $\mathbf{I}$ *be a set of individual names. For* $a, b \in \mathbf{I}$*,* C *a possibly complex* $\mathcal{ALC}$ *concept in* $\mathbf{C}$ *and* $R \in \mathbf{R}$ *a role name, an expression of the form* $C(a)$ *is called an* $\mathcal{ALC}$ *concept assertion and* $R(a, b)$ *is called an* $\mathcal{ALC}$ *role assertion. A finite set of* $\mathcal{ALC}$ *concept and role names is called an* $\mathcal{ALC}$ *ABox* (*or simply, an ABox*)*.*

Definition 3 (TBox)**.** *For* C *and* D *possibly complex* $\mathcal{ALC}$ *concepts, an expression of the form* $C \sqsubseteq D$ *is called an* $\mathcal{ALC}$ *general concept inclusion* (*GCI*)*.* $C \equiv D$ *is an abbreviation of* $C \sqsubseteq D$ *and* $D \sqsubseteq C$*. A finite set of GCIs is called an* $\mathcal{ALC}$ *TBox (or simply, a TBox).*

A TBox is called *acyclic* iff no concept name uses itself.

Definition 4 (Knowledge base)**.** *A knowledge base is a pair* $\langle \mathcal{A}, \mathcal{T} \rangle$ *made up of an ABox* $\mathcal{A}$ *and a TBox* $\mathcal{T}$*.*

In this thesis, we use the word ontology to refer to a knowledge base made up of an ABox and a TBox.

To give the semantics of $\mathcal{ALC}$ concepts and roles, *interpretations* are used. An interpretation is formed of an interpretation domain containing a set of individuals. An interpretation fixes for each concept name, its extension, i.e., indicating which of the elements belong to which concepts. It fixes, as well, for each role name its extension, which pair of elements are related to which roles.

Definition 5 (Semantics of $\mathcal{ALC}$)**.** *An interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is composed of a non-empty set* $\Delta^{\mathcal{I}}$*, called the* domain *of* $\mathcal{I}$*, and a* valuation *function* $\cdot^{\mathcal{I}}$ *which maps every concept name in* $\mathbf{C}$ *to a subset of* $\Delta^{\mathcal{I}}$*, every role name in* $\mathbf{R}$ *to a subset of* $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$*.*

This valuation function is extended to concepts in such a way that, for all concepts C, D *and for every role name* R *in* $\mathbf{R}$*, the following is fulfilled:*

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

An interpretation $\mathcal{I}$ satisfies an ABox assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, an Abox assertion $R(a, b)$ if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$. Given an ABox assertion α, $\mathcal{I} \models \alpha$ denotes that $\mathcal{I}$ satisfies α. $\mathcal{I}$ is a model of an ABox $\mathcal{A}$ if it satisfies every ABox assertion in it.

An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$, denoted by $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a *TBox* $\mathcal{T}$ if $\mathcal{I}$ satisfies every GCI in $\mathcal{T}$.

An interpretation $\mathcal{I}$ is a model of an ontology $\mathcal{O}$ composed of an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$ if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{A}$.

2.2.2 Reasoning problems and services

The precise syntax and semantics of description logics, together with their nice computational properties, make them amenable to automated inference. Three basic types of reasoning tasks for $\mathcal{ALC}$ namely, concept satisfiability, ontology satisfiability, and the more general task of entailment. Given an $\mathcal{ALC}$ ontology $\mathcal{O}$, concept satisfiability is the task of checking whether C, a possibly complex concept, is satisfiable w.r.t. $\mathcal{O}$. This means to check if there is a model $\mathcal{I}$ for $\mathcal{O}$ s.t. $C^{\mathcal{I}} \neq \emptyset$. The second one is checking the satisfiability (or consistency) of a given ontology. Given an ontology $\mathcal{O}$, $\mathcal{O}$ is satisfiable if it has a model. The last and most general one is ontology entailments. Entailment of a TBox or ABox statement, α, α is entailed by $\mathcal{O}$, written as $\mathcal{O} \models \alpha$, if each model for $\mathcal{O}$ is also a model for α.

2.2.3 Reduction of ontology entailment to ontology consistency

In $\mathcal{ALC}$, ontology entailment of GCIs, concept assertions, is reducible to ontology consistency checking. Indeed, given an $\mathcal{ALC}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}\rangle$, two concepts C, D and two individuals a, b:

1. $\mathcal{O} \models C \sqsubseteq D$ iff $\langle \mathcal{A} \cup \{(C \sqcap (\neg)D)(x)\}, \mathcal{T}\rangle$ is inconsistent
2. $\mathcal{O} \models C(a)$ iff $\langle \mathcal{A} \cup \{(\neg)C(a)\}, \mathcal{T}\rangle$ is inconsistent

where x is an individual not present in $\mathcal{O}$. Notice that ontology entailment of role assertions ($R(a, b)$) may require considering negation of roles, which go beyond $\mathcal{ALC}$ expressivity.

As a consequence, all the reasoning problems mentioned above can be reduced to knowledge base or ontology consistency, i.e, we can use a knowledge base consistency technique to answer this reasoning questions. The most famous reasoning technique is the tableau-based technique.

2.2.4 Extension of $\mathcal{ALC}$ with inverse roles $\mathcal{I}$

The fact that a family of description logics includes inverse role is normally indicated by the letter $\mathcal{I}$ in its name. Adding inverse roles to the description logic $\mathcal{ALC}$ will enlarge the scope of reasoning applications.

Definition 6 (Inverse roles)**.** *For a role name R, R^- is an inverse role. The set of roles is* $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$.

The semantics of inverse role is $(R^-)^{\mathcal{I}} := \{\langle a, b\rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} | \langle b, a\rangle \in (R)^{\mathcal{I}}\}$.

In the description logic $\mathcal{ALCI}$, inverse roles can occur in existential and universal restrictions, for example, $\exists r^-.(\forall s.(\exists R.A \sqcap \forall s^-.B))$ is an $\mathcal{ALCI}$ concept.

2.3 Tableau algorithms for reasoning in $\mathcal{ALC}$

The idea behind the tableau-based approach is proving the consistency of a knowledge base by demonstrating the existence of a suitable witness of its consistency. Given a knowledge base $\mathcal{K}$ formed of an Abox $\mathcal{A}$ and a TBox $\mathcal{T}$, a witness of consistency is an interpretation $\mathcal{I}$ that represent a model for the input knowledge base ($\mathcal{I} \models \mathcal{K}$). This model can be represented as a graph whose nodes represent the individuals of the model. The labels of these nodes represent the set of concepts satisfied by the individuals. The labels of the edges between two nodes represent the roles between these two individuals.

The algorithm works on a set of ABoxes, initialized with the input ABox $\mathcal{A}$. Each ABox represents a possible model for the input ontology. A set of completion rules (Figure 2.1) that explicate the semantics of $\mathcal{K}$ is applied on ABoxes contained in this set. The new ABoxes are copies of the original one plus some new assertions. There are two types of rules: deterministic and non-deterministic rules. The deterministic rules are $\rightarrow_{\sqcap}$, $\rightarrow_{\forall}$, $\rightarrow_{\exists}$, and $\rightarrow_{\sqsubseteq}$. The non-deterministic rules is $\rightarrow_{\sqcup}$. Upon each application of a completion rule transforms the an ABox is replaced by one or two ABoxes.

The interaction between the $\rightarrow_{\exists}$ and the $\rightarrow_{\sqsubseteq}$ might lead to a non-termination problem. Given, for example, an ontology composed of a TBox $\{A \sqsubseteq \exists r.A\}$, and an ABox $\{A(a)\}$. The algorithm first applies $\rightarrow_{\sqsubseteq}$ rule which will create two ABoxes $\mathcal{A}'$ and $\mathcal{A}''$. The first one $\mathcal{A}'$ contains $A(a)$ and $\sim A(a)$ and thus it is not clash-free. The second one $\mathcal{A}''$ contains $A(a), R(a,t), A(t)$. The individual t satisfy the condition of the $\rightarrow_{\sqsubseteq}$ through $A \sqsubseteq \exists r.A$. The application of $\rightarrow_{\sqsubseteq}$ on t will in turn generate two new ABoxes. Similarly, this will create a new individual which satisfy the condition of the $\rightarrow_{\sqsubseteq}$ and so forth. In order to avoid the non-termination problem, the tableau algorithm uses a technique called blocking. This technique detects repetitions "loops" in partially constructed models. When an individual is blocked the rules generating individual are not longer applicable on it. This technique guarantees the termination of a tableau algorithm and preserves its soundness and completeness.

Rule $\rightarrow_{\sqcap}$
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(s)$, but it does not contain both $C_1(s)$ and $C_2(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(s), C_2(s)\}$
Rule $\rightarrow_{\sqcup}$
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(s)$, but neither $C_1(s)$ nor $C_2(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(s)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{C_2(s)\}$
Rule $\rightarrow_{\forall}$
Condition: $\mathcal{A}$ contains $(\forall R.C)(s)$ and $R(s,t)$, but it does not contain $C(t)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(t)\}$
Rule $\rightarrow_{\exists}$
Condition: $\mathcal{A}$ contains $(\exists R.C)(s)$ but there is no individual name t such that $\mathcal{A}$ contains $R(s,t)$ and $C(t)$, and s is not blocked.
Action: $\mathcal{A}' := \mathcal{A} \cup \{R(s,t), C(t)\}$ where t is an individual not occurring in $\mathcal{A}$.
Rule $\rightarrow_{\sqsubseteq}$
Condition: $\mathcal{T}$ contains $C \sqsubseteq D$ and there is an individual name s such that $\mathcal{A}$ does contain neither $\sim C(s)$ nor $D(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{\sim C(s)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{D(s)\}$

Figure 2.1 – Completion rules for $\mathcal{ALC}$.

An ABox that contains a clash is called closed and otherwise it is called open. A tableau to which no rule can be applied is called complete, the term complete comes from the inability to apply any completion rule on it (detailed explanations of closed, open and complete ABox are given in the next chapter). If the algorithm was able to build a complete and open ABox, then this ABox represent a witness for the consistency of $\mathcal{K}$. In this case, the algorithm returns true. Otherwise, the algorithm concludes that $\mathcal{K}$ is inconsistent and returns false.

2.4 Conclusion

In this chapter, we have explained the syntax and semantics of the $\mathcal{ALC}$ family of description logics, which allows to formally describe the ontologies of a specific domain. We have highlighted its main reasoning problems as well as its extension with inverse roles. Then we have presented the standard tableau algorithm for the description logic $\mathcal{ALC}$. In the next chapter, we will present the state-of-art algorithms for reasoning with keys and rules in description logics.

Chapter 3

Literature Review

" The question of whether a computer can think is no more interesting than the question of whether a submarine can swim." Edsger W. Dijkstra

There have been different methods proposed to do reasoning in description logics. Among these methods, we discuss the approaches that allow to do reasoning with *keys*, as link keys are generalisation of keys. We also study the approaches that allow to do reasoning with *rules*, as link keys can be modelled using rules. Furthermore, since link keys are axioms that hold between a pair of different ontologies, we also study the methods that have been proposed to do reasoning with *correspondences*. Finally, we present an overview of the complexity of reasoning in the description logic $\mathcal{ALC}$ and some of its simple extensions.

3.1 Reasoning in description logics with ontological constraints

Keys are widely used in modeling databases and RDF datasets. More interestingly, the nature of keys made them suitable for addressing the problem of data interlinking. A key is the set of properties that uniquely identify the instances of a given concept. Given an ontology $\mathcal{O}$, a key κ over $\mathcal{O}$ has the following form

$$(\{p_1, ..., p_k\} \text{ key } C)$$

where $p_1, \ldots, p_k$ are roles and C is a concept in $\mathcal{O}$. An interpretation $\mathcal{I}$ satisfies κ if

$$\forall \delta, \eta, x_1, \cdots, x_n \in \Delta^{\mathcal{I}}, \delta, \eta \in C^{\mathcal{I}} \wedge \bigwedge_{1 \leq i \leq k} ((\delta, x_i) \in p_i^{\mathcal{I}} \wedge (\eta, x_i) \in p_i^{\mathcal{I}}) \Rightarrow \delta = \eta.$$

However, when the datasets are described using different ontologies, *alignments* providing correspondences between the roles and concepts of different ontologies, must be used to perform data interlinking. Different approaches integrating keys to description logics have been proposed. Some of these approaches are concerned with proving the decidability of a specified family of description logics extended with keys, or establishing the lower bound complexity of the extended logic. More interestingly, other approaches provide a reasoning mechanism for the extended logic. There exists a variety of reasoning approaches for checking the satisfaction of a description logic family. These approaches includes consequence-based approaches [18], automata-based approaches [19]. However, the most popular one is the tableau based approach (Subsection 3.1.1 and Subsection 2.3 of Chapter 2 for a detailed explanation).

3.1.1 Tableau algorithms for reasoning in description logics

The tableau approach is a technique for deciding the consistency problems in description logics. There exist efficient tableau algorithms implementing the tableau-based approach such as Pellet [20], Hermit [21], and Racer [22]. To check the consistency of an ontology, tableau-based algorithms apply a set of completion rules implied by the semantics of constructs contained in the given description logic family to derive a *complete* and *clash-free* ABox. A complete ABox is an ABox to which no more completion rule is applicable and a clash-free ABox is an ABox that contains no contradictions. This ABox represents a model of the input ontology and can be represented by a completion graph. In the presence of cyclic TBoxes, the termination of these algorithms is ensured by using a blocking technique. Among other completion rules for the description logic $\mathcal{ALC}$, there exists $\exists$-rule and $\sqcup$-rule which respectively handle the existential restrictions and concept disjunction. The existential rule ($\exists$-rule) can generate an exponential number of new individuals contained in the derived ABoxes. The non-deterministic rule ($\sqcup$-rule) can generate an exponential number of ABoxes derived from the input ABox $\mathcal{A}$. The interaction between these rules is usually responsible for the doubly exponential complexity of the standard tableau algorithm for description logic families containing these constructs [23].

There exists, as well, a compressed tableau algorithm [16]. This algorithm, in contrast to the standard tableau algorithm, has an exponential complexity for the description logic $\mathcal{SHOIQ}$. The description logic $\mathcal{SHOIQ}$ is an expressive family of description logics formed of the following constructors which extends $\mathcal{ALC}$ with transitive roles, role hierarchy, nominals, inverse roles, and qualified number restrictions. This method uses a *star-type* for representing a group of similar individuals. Similar individuals are equal individuals or individuals satisfying the same set of concepts and have the same connections to other individuals. The algorithm reduces the exponential complexity raised by handling disjunctions. When dealing with a disjunction, the algorithm instead of duplicating the whole current structure which aims to represent a model, generates just a new star-type to be chosen if needed. This behaviour allows reducing global non-determinism to local non-determinism.

3.1.2 Reasoning with keys

Keys have been integrated to description logics families in two ways. The first adds keys into a separate set of constraints called key box [24, 25, 26, 27]. While others model keys by a new concept constructor called a path-functional dependency (PFD) [28], or special primitive concepts [29].

3.1.2.1 Reasoning with keys in a separate set of constraints

$\mathcal{DLR}$ is an expressive description logic family particularly suited for modeling database schemas. It generalises the description logic $\mathcal{ALCQI}$, which extends $\mathcal{ALC}$ with inverse roles and cardinality restrictions. In [24], the authors show that identification constraints and functional dependencies can be captured by $\mathcal{DLR}$ logic. They prove that unary keys, i.e. keys that contain only a single property, can be modeled through number restrictions. Non-unary keys, on the contrary, cannot be represented in $\mathcal{DLR}$. They proved that the consistency checking in $\mathcal{DLR}$ is EXPTIME-complete. This approach reduces the problem of reasoning in $\mathcal{DLR}$ to the problem of reasoning in the description logic $\mathcal{CIQ}$ [30] and then uses the existing inference methods for $\mathcal{CIQ}$. In [25], the authors extend the $\mathcal{DLR}$ logic with a formalisation of database

keys named $\mathcal{DLR}_{key}$. They proved that reasoning in $\mathcal{DLR}_{key}$ can be reduced to reasoning in $\mathcal{DLR}$ by expressing keys as $\mathcal{DLR}$ assertions. It follows that the worst-case computational complexity of $\mathcal{DLR}_{key}$ is EXPTIME-complete, as it is the case for $\mathcal{DLR}$.

The main reason for this complexity result is the reducing the problem of reasoning in $\mathcal{DLR}_{key}$ into the problem of verifying keys, and testing the satisfiability of $\mathcal{DLR}$ of a possibly exponential number of knowledge bases of polynomial size. Also, the authors do not provide any reasoning algorithm for the proposed logic.

In [26], Calvanese et al. extend $\mathcal{DLR}$ with a formalisation of database keys and present an algorithm for reasoning over it. The extended logic is called $\mathcal{DLR}_{ifd}$. A $\mathcal{DLR}_{ifd}$ knowledge base $\mathcal{K}$ is a set of $\mathcal{T} \cup \mathcal{A} \cup \mathcal{F}$ assertions where $\mathcal{T} \cup \mathcal{A}$ is a $\mathcal{DLR}$ knowledge base and $\mathcal{F}$ a set of identification and functional dependency assertions. A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of the other sets. An identification assertion (or key) on a concept has the form:

$$(id\ C\ [i_1]R_1, ..., [i_h]R_h)$$

where C is a concept, each R_j is a relation, and each i_j denotes one component of R_j. This assertion states that two instances of C cannot agree on the participation to $R_1, ..., R_h$ via components $i_1, ..., i_h$, respectively. They also provide a reasoning procedure for $\mathcal{DLR}_{ifd}$ that takes as input a $\mathcal{DLR}_{ifd}$ knowledge base $\mathcal{K}$, and tries to find a saturated ABox $\mathcal{A}_s$ that represents a model for $\mathcal{K}$. This reasoning procedure has the same complexity result as [25]. In addition, this procedure is not implemented.

Lutz et al. introduce in [27] the $\mathcal{ALCOK}$(D) family of description logics that contain keys composed of features, i.e. functional roles whose values belong to a concrete domain. Adding concrete domains to description logics allows to describe concrete features of instances like name or age. They provide a tableau algorithm for deciding $\mathcal{ALCOK}$(D)-concept satisfiability with respect to boolean restricted key boxes, i.e. key boxes restricted to a boolean combination of concept names. This tableau algorithm starts with an initial data structure induced by the input concept and then repeatedly applies completion rules to it. The completion rules are enriched with **Rch** and **Rp** rules. The **Rch** rule is a non-deterministic rule that guesses if an individual satisfies the concept present in the key. This rule is necessary to ensure the completeness of the algorithm. The **Rp** rule handles equalities between individuals. The complexity of this algorithm is NEXPTIME. The authors have also identified several undecidable cases, which are all related to the presence of a concrete domain.

There exists, as well, a set of algorithms, based on the chase technique, for query answering in the presence of Tuple and Equality Generating Dependencies (TGDs, and EGDs). TGDs are similar to Horn rules with existentially-quantified variables in the head. EGDs are a generalisation of functional dependencies [11]. The chase technique modifies the individuals present in the knowledge base by a set of chase steps until all dependencies are satisfied. However, problems arise from the combination of EGDs and TGDs, because for simple types of EGDs, the implication problem for EGDs and TGDs, and the query answering problem are undecidable [31]. Hence, there exists no algorithm for which implication can be decided or queries can be answered using finite resources.

3.1.2.2 Reasoning with keys as a new concept constructor

In [28], the authors introduce keys as a new constructor called path-functional dependency (PFD). PFD is a more general form of functional dependency constraint for semantic data models. They show that full integration of keys and functional dependencies in the families of description logics that include inverse roles is undecidable. They provide syntactic restrictions on occurrences of PFDs and on the syntax of the PFD constructor itself which guarantees decidability. More precisely, they show that allowing PFDs within the scope of monotone concept constructors on the right-hand side of inclusion dependencies leads to decidable implication problems. Moreover, they show that reasoning with unrestricted PFDs and inverse roles is undecidable.

In [29], the authors present a sound and complete mechanism for modeling keys and functional dependencies. However, the family of description logics they used has very limited expressivity. It lacks general inclusion axioms and inverse roles are not taken into account.

Finally, we would like to point out that it is possible to model keys using owl:hasKey construct contained in OWL 2. However, the semantics of owl:hasKey is restricted to being applied to only named individual and not to other individuals.

3.1.2.3 Link keys cannot be reduced to keys

It is possible to express keys as link keys. Certainly, $(\{P_1, ..., P_n\}\ \mathsf{key}\ C)$ is equivalent to $(\{\langle P_1, P_1\rangle, ..., \langle P_n, P_n\rangle\})\ \mathsf{linkkey}\ \langle C, C\rangle$. However, link keys are not necessarily expressible as keys. Consider, for instance, the link key $\lambda = (\langle P, Q\rangle\ \mathsf{linkkey}\ \langle \top, \top\rangle)$. Among all the possible keys κ_1, κ_2, and κ_3 built from the properties P and Q for $\top$ there is no key that has the same model as λ. Indeed, let $\kappa_1 = (P\ \mathsf{key}\ \top), \kappa_2 = (Q\ \mathsf{key}\ \top)$ and $\kappa_3 = (\langle P, Q\rangle\ \mathsf{key}\ \top)$. Consider now the interpretation $\mathcal{I}$ defined by $P^{\mathcal{I}} = \{(a, u), (b, u)\}$ and $Q^{\mathcal{I}} = \{(a, v), (b, v)\}$ where a, b, u, v are different domain individuals. Then $\mathcal{I} \not\models \kappa_i$ $(i = 1, 2, 3)$ whereas $\mathcal{I} \models \lambda$.

3.1.2.4 Conclusion

The algorithms given in Chapters 4, 5, and 6 allow to do reasoning with link keys formed from multiple property pairs which is not the case for non-unary keys in [24], and unlike [29] these algorithms allow to do reasoning with general inclusion axioms, and unlike [24, 25, 26] these algorithms allow to do reasoning with concept disjunction. Furthermore, in contrast to [27], the algorithm given in Chapter 6 is implemented and has an EXPTIME complexity. Finally and above all, since link keys cannot be expressed as keys, we cannot directly use the approaches described above for reasoning with keys to reason with link keys.

3.1.3 Reasoning with rules

A logical rules is an implication between an antecedent and a consequent:

$$B_1, \cdots, B_n \rightarrow H$$

where H is called consequent or head and $B_1, \cdots, B_n$ is called antecedent or body. It states that whenever the conditions specified in the antecedent hold, the conditions specified in the consequent must hold as well. The Semantic Web Rule Language (SWRL) is a combination of

OWL description logic ($\mathcal{SROIQ}$) and logical rules. Such a combination increases the expressivity of OWL, by allowing to express arbitrary axioms. However, SWRL is undecidable [33]. The reason behind SWRL's undecidability is based on the ability to encode the infinite tiling domino problem [34] in SWRL.

1. Description Logics Rules: SWRL rules that can be expressed with OWL 2, i.e. OWL family of ontologies that are based on $\mathcal{SROIQ}$ DL.

2. DL-safe Rules: SWRL rules that have variable assignment constraints.

3.1.3.1 Reasoning with DL-safe rules

A DL-safe rule is a logical rule of the form

$$B_1, \cdots, B_n \rightarrow H$$

where H and B_i are DL-atoms. A DL-atom is an atom of the form $C(a), R(a, b)$ where C and R are respectively a concept and a role in the input ontology. DL-safe rules are rules applied to only known individuals. For that they necessitate that each variable in the body of the rule is bound to a non-DL atom, i.e. an atom that does not occur in the input ontology. Adding this non-DL atom concept assertion for each explicitly named individual in the input knowledge base ensures that the variables are bound to only individuals present in the input knowledge base, and not to new individuals created due to the presence of existential axioms. The effect of the DL-safety restriction varies according to the nature of the application. In some applications, the scope of reasoning is limited to known individuals like information retrieval. In such applications, the impact of DL-safety restriction is not serious and DL-safe rules can infer most of the possible conclusions. In contrast, in applications like natural language processing, requiring intensional reasoning, i.e. reasoning exploiting the conceptual schema (which creates unnamed individuals), DL-safety is a serious restriction involving since many conclusions involve unnamed individuals.

In [35], the authors present a decidable algorithm for reasoning in OWL-DL with DL-safe rules. This algorithm is based on resolution calculus. Resolution calculus is widely used for theorem proving in first-order logic. To prove the correctness of a statement, the resolution calculus adds its negation to the sets of axioms that are known to be true. It then uses a set of inference rules to show that this leads to a contradiction. This algorithm reduces a knowledge base to a positive disjunctive datalog program [36, 37], leading to the same set of ground facts as the knowledge base. This algorithm runs in EXPTIME-complete.

The variables occurring in the body of a DL-safe rule can only be bound to named individuals. However, this is not a general restriction of link keys, but link keys made-up of $\mathcal{ALC}$ concepts and roles are only satisfied by named individuals (Lemma 4 of Chapter 6). As a result, in $\mathcal{ALC}$+$\mathcal{LK}$ link keys can be modelled as DL-safe rule. Consider the link key $\lambda = (\{\langle P_1, Q_1\rangle, \cdots, \langle P_n, Q_n\rangle\}$ linkkey $\langle C, D\rangle)$, λ can be transformed into the following rule DL-safe rule r:

$$P_1(x, u), Q_1(y, u), \cdots, P_n(x, t), Q_n(y, t), C(x), D(y), O(x), O(y) \rightarrow x \approx y$$

where O is a non-DL atom and $x, y, u, \cdots, t$ are variables bound to named individuals.

The DL-safety restriction does not allow to model link keys containing inverse roles. Since, in the presence of inverse roles, link keys can be satisfied by new individuals not occurring in the initial ABox. The body and the consequent of a link key rule, in this case, contain unnamed objects. Let $\lambda = (\{\langle P^-, Q^- \rangle\}$ linkkey $\langle C, D \rangle)$ be a link key, where P^- and Q^- are the inverse of the named roles P and Q and let r be the DL-safe rule of λ:

$$r : P^-(x,t), Q^-(y,t), C(x), D(y), O(x), O(y) \rightarrow x \approx y$$

where O is a non-DL atom. Let $\mathcal{I}$ be an interpretation defined by $C^{\mathcal{I}} = \{a\}$, $D^{\mathcal{I}} = \{b\}$, $P^{\mathcal{I}} = \{(u,a)\}$ and $Q^{\mathcal{I}} = \{(u,b)\}$ where u is an old (known) individual and a, b are new (unknown) individuals. Since $P^{\mathcal{I}} = \{(u,a)\}$ and $Q^{\mathcal{I}} = \{(u,b)\}$ we get that $(P^-)^{\mathcal{I}} = \{(a,u)\}$ and $(Q^-)^{\mathcal{I}} = \{(b,u)\}$, $\mathcal{I} \models \lambda$. However, a and b cannot be bound to the variables of r. Link keys composed of inverse roles have been shown useful for the data interlinking task [38].

3.1.3.2 Conclusion

Link keys can be modeled as unrestricted SWRL rules (since they can be modeled as DL-safe rules), however, this will lead to the undecidability of reasoning with link keys. The restricted decidable fragments of SWRL rules are description logic Rules and DL-safe rules. Link keys cannot be modeled as description logic Rules, as they cannot be expressed in OWL2. The reason behind this is the inability of OWL2 to connect the values of the properties specified in a link key and say that they are the same. To the contrary, link keys made up of $\mathcal{ALC}$ concepts and roles can be modelled as DL-safe rules, but this not the case for description logic languages containing inverse roles. The compressed tableau algorithm for the description logic $\mathcal{ALC}+\mathcal{LK}$ given in Chapter 6 can be extended. We discuss in Chapter 6, the first steps toward designing a compressed tableau algorithm for the description logic $\mathcal{ALCI}+\mathcal{LK}$.

3.1.4 Reasoning with correspondences

A Distributed System (DS) is composed of a set of ontologies, and these ontologies are interconnected by correspondences. An ontology correspondence determines the relationship between concepts and properties of different ontologies. Link keys are axioms that decide whether two individuals present in possibly two different ontologies are equal or not. The problem of reasoning with link keys can be thus seen as a problem of reasoning in a distributed system.

Reasoning in a distributed system can be done using *centralized* or *decentralized* reasoning approaches. The reasoning algorithms introduced above are centralized, i.e. they assume that the complete knowledge is contained in a single system and all reasoning steps are carried out on this system. In this thesis, as well, we do centralized reasoning. In which we merge a set of $\mathcal{ALC}$ ontologies, and a set of link keys into a single knowledge base and we carry the reasoning procedure on it.

In contrast to the centralised reasoning algorithms, there exist other decentralized reasoning algorithms. The decentralized algorithms perform a set of reasoning tasks, each task is done in a separate knowledge base. The result of each reasoning task is propagated among the distributed system through *bridge rules*. There have been different approaches proposed to do reasoning in a distributed system [39, 40, 41, 42]. The decentralized approaches respect privacy and allow for information hiding by requiring only local reasoning rather than global reasoning that

requires the global reasoner to access each knowledge base of the system. More importantly, it supports languages specificity, since it combines different local reasoning procedures, each supporting the local ontology language. However, these approaches usually suffer from poor performance due to the massive amount of message exchanges between the different local reasoners and checking solving queries which require to access the global ontology.

In [39], the authors introduce the notion of Distributed description logic (DDL). A DDL knowledge base consists of a set of description logic Information Systems (IS) each possibly equipped with a set of TBoxes, and a set of DDL alignments between them. DDL alignments are encoded using bridge rules. These bridge rules describe the correspondences between the different IS in the DDL. They extended the reasoning mechanisms on IS described using one common schema to the case of a DDL. To avoid the propagation of local inconsistencies presented on a single IS or inconsistencies in the bridge rules to the global system, they introduced a set of interpretations which limit the effect of inconsistent local TBoxes in the desired way. They proved that reasoning in DDL can be translated to description logic reasoning. This requires that each ontology contained in DDL is contained in some decidable family of description logic and that the description logic supports qualified existential restriction and general theories, $\mathcal{SHIQ}$ TBoxes. This necessitates the existence of a local reasoner for at least one local ontology of the DDL system that supports reasoning in $\mathcal{SHIQ}$.

In [40], the authors propose a tableau algorithm for reasoning on DDL ontologies with restricted bridge rules. The distributed TBoxes in the DDL must be acyclic and the restricted bridge rules do not allow for individual and complex concept. The basic idea behind this algorithm is the translation the satisfiability problem with respect to a DDL ontology into a set of several local satisfiability problems with respect to local ontologies. This algorithm has been implemented in a software called DRAGO and many approaches [43, 44] exploit successfully the reasoning algorithm implemented in DRAGO to test the correspondences which are created by different matching systems. The main drawback of this algorithm is the need of having a tableau-based reasoning for local ontologies.

There exist other decentralized system called Integrated Distributed description logics (IDDL)-based systems. These systems are very different from those based on DDL. In IDDL-based systems, the reasoner takes into account the correspondences between ontologies for deciding the local consistency of each ontology. These systems have higher computational complexity than the DDL-based systems since they check each model of mappings against all local ontologies.

In [45], the authors design an algorithm for reasoning on an IDDL system composed of $\mathcal{SROIQ}$ ontologies. This algorithm searches for the valid settings from which extended ontologies with a consistent ontology alignment can be built from the input system. If the algorithm cannot find any such configuration then the input IDDL system is inconsistent. This system is decidable when the consistency of the local description logics is decidable. Moreover, in contrast to [39], the expressiveness of local knowledge bases does not need to be known as long as local reasoners can handle at least $\mathcal{ALC}$. The complexity of this algorithm is bounded by a double exponential function in the size of the alignments and the authors themselves declared that their algorithm is intractable.

In [42], an algorithm for reasoning on a network of aligned ontologies possibly including link keys was developed. This reasoner allows reasoning in a decentralized manner under the IDDL semantics [41]. This algorithm uses two sub-algorithms. The first one allows the propagation equalities. These equalities are either individual equalities of individuals belonging to ontologies in the network, or equalities resulting from applying link key correspondences.

The second algorithm allows the propagation of concept unsatisfiabilities from one ontology to the other via concept correspondences. The main algorithm terminates when the ontologies and the alignment are no longer modified. This algorithm is executed by a global reasoner. To represent knowledge propagation through the network, the global reasoner sends different assertions/axioms to local reasoners located on different sites. The local reasoners check the entailment and the consistency of the local ontologies. The global reasoner and each local reasoner use HermiT [46] as OWL reasoners. The communication between the global reasoner and all local reasoners is based on OWLLink [47]. This algorithm runs in polynomial time in the size of the network if each check of entailment or consistency occurring in the algorithms is considered an oracle. The algorithm has been implemented and integrated within a reasoner called Draon [48]. The authors have carried out a set of experiments and reported good run time results. To achieve better performance results, the authors introduce a new semantics of correspondences: given a correspondence $C \rightarrow D$, the unsatisfiability of D implies unsatisfiability of C, rather than concept subsumption as usual, where C and D belong to the concepts of two different ontologies in the network.

3.1.4.1 Conclusion

The nature of link keys makes them adequate for reasoning in a IDDL-based systems. However, the doubly exponential complexity of [45] restricts us from extending to deal with link keys. In addition, the soundness of [42] cannot be established when ontologies that allow for inverse roles or when the correspondences are equipped with the standard semantics. The restriction on inverse roles does not allow for merging individuals which are not in the initial ABox. This is not the case for the extendable EXPTIME compressed algorithms given in Chapters 5 and 6.

3.2 The complexity of reasoning in the description logic $\mathcal{ALC}$ and its simple extensions

The complexity of the standard tableau methods for reasoning in the description logic $\mathcal{ALC}$ with general TBox is 2EXPTIME [21]. The sources of complexity in the tableau algorithms are the creation of possibly an exponential number of new ABoxes upon the application of non-deterministic rules (or-branching), and the exponential expansion of the model upon the application of the existential rule for satisfying the existential quantifiers (and-branching). Below, we report on some works that discuss the complexity of deciding consistency in the description logic $\mathcal{ALC}$ with general TBoxes.

In [15], the authors proved that the lower bound complexity for concept satisfiability in $\mathcal{ALC}$ with general TBoxes is EXPTIME-complete. This lower bound complexity for concept satisfiability in $\mathcal{ALC}$ was proved by reducing the existence of winning strategies for Infinite Boolean Game (IBG). In [49], the authors proposed the first worst-case optimal EXPTIME tableau algorithm for reasoning in an $\mathcal{ALC}$ knowledge base. This algorithm is worst-case optimal because it matches the lower bound was achieved for the description logic $\mathcal{ALC}$ with general TBoxes. This algorithm works on a prefixed formulae $e{:}C$ where e is a string corresponding to a particular world in the underlying model built by the tableau and C is a set of concepts. The reason behind this complexity result is computation reuse for both concept satisfiability and unsatisfiability. The main idea behind computation reuse is learning from local

proofs of unsatisfiability by permanently caching them and by temporary caching of satisfiable witnesses. A set of concepts C in $e{:}C$ is not processed if it has already been shown to be satisfiable in e or in any other model e' where $e' < e$. A set of concepts C in $e{:}C$ is not processed if it is unsatisfiable in any of the underlying models. They have discussed as well the effect of several optimizations like propositional backjumping, simplification, or semantic branching on the complexity. Propositional backjumping skips over the concepts that have already contributed to unsatisfiability. Simplification allows to simplify the structure of concepts at any level of nesting of connectives, this is done by reducing the substructure that is already contained in the ontology. Semantic branching chooses the common disjuncts between the alternative branches of the search tree and applies the disjunction rule on the basis of these disjuncts. This allows to detect recurrence of an unsatisfiable disjunct and thus avoids reconsidering them. They have as well defined their own optimization including modal backjumping, which allows to skip the branching points visited by the algorithm before to the prefix that has been proven unsatisfiable. However, to the best of our knowledge, this algorithm has not been implemented and thus these optimization techniques are not utilized.

The authors of [21], present a refinement of the tableau reasoning algorithms called hypertableau. This algorithm tries to curb the sources of inefficiency in the tableau reasoning algorithms. For the complexity arriving from the application of existential rules, the algorithm uses anywhere pairwise blocking, which allows extending the set of possible blockers that prohibits the application of existential rules. For the second origin of complexity coming from the application of non-deterministic rule, the algorithm relies on hyperresolution calculus. The algorithm behaves as follows, it first preprocesses the input knowledge base into a set of DL-clauses, that are universally quantified implications containing description logic concepts and roles as predicates. The main derivation rule for DL-clauses is hyperresolution which means that an atom from the consequent of a DL-clause is derived only if all atoms from the DL-clause antecedent can be matched to already derived consequences. This algorithm allows to do reasoning in the description logic $\mathcal{SHOIQ}+$. The authors first believed that anywhere blocking will improve the worst-case complexity for $\mathcal{SHOIQ}+$, however, this is not the case ([21], pages 213 to 216). Despite its 2EXPTIME complexity, the algorithm has performance improvements over state-of-the-art reasoners on several famous ontologies such as Gazetteer and GALEN.

We now report on the complexity results of some simple extensions of $\mathcal{ALC}$ [15]. Reasoning in the description logics $\mathcal{ALCI}$ and $\mathcal{ALCQ}$ which respectively extend $\mathcal{ALC}$ with inverse roles and qualifying number restrictions are EXPTIME-complete. The complexity of reasoning in the description logic $\mathcal{ALCIQ}$ with general TBoxes is the same.

However, concept satisfiability and subsumption in the description logic $\mathcal{ALCOI}$ which extend $\mathcal{ALCI}$ with nominals are EXPTIME-hard without TBoxes. Also, concept satisfiability in the description logic $\mathcal{ALCOIQ}$ is NEXPTIME-complete whether or not TBoxes are present. There are even some extensions of $\mathcal{ALC}$ in which satisfiability and subsumption are undecidable. The first one is the extension of $\mathcal{ALC}$ with so-called role value maps (RVMs). The second one is the extension of $\mathcal{ALC}$ with concrete domains, called $\mathcal{ALC}(\mathcal{D})$. The proofs of undecidability are based on the reduction of an undecidable version of the tiling problem and on the reduction from the halting problem of two-register machines respectively.

3.2.1 Conclusion

Anywhere blocking which is used in [21] is used in all our reasoning algorithms. However, it is not the reason behind the reduction of the algorithm complexity from 2EXPTIME to EXPTIME. Indeed, the algorithm given in Chapter 4 has a 2EXPTIME complexity which is not the case for the EXPTIME algorithms given in Chapters 5 and 6. These algorithms are inspired by the EXPTIME compressed tableau algorithm for reasoning in the description logic $\mathcal{SHOIQ}$ [16]. They introduce an efficient way to handle non-determinism, which is one of the main objectives of the techniques used in [49, 21]. Instead of duplicating the whole completion tree being constructed upon the application of non-deterministic rules, as it is the case for the standard tableau algorithms, the algorithms given in Chapters 5 and 6, only duplicate the part of the tree that is concerned with non-determinism.

Moreover, the lower bound complexity for reasoning in the description logic $\mathcal{ALC}$ is EXPTIME-complete. This means that there exists an EXPTIME algorithm for reasoning in $\mathcal{ALC}$ and all EXPTIME problems can be polynomially reduced to $\mathcal{ALC}$. The algorithms proposed in Chapters 5 and 6 have an EXPTIME complexity and they are worst-case optimal. Since otherwise, there must exist a polynomial algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. Since all EXPTIME problems can be polynomially reduced to $\mathcal{ALC}$ then, in this case, PTIME = EXPTIME , which contradicts the result PTIME < EXPTIME .

We would also like to point out that despite the undecidability results of reasoning with concrete domain [15, 27], it is absence does not seem to present a limitation of reasoning with link keys (Chapter 7).

3.3 Summary

Link keys are axioms for deciding the equality between two individuals that possibly belong to different ontologies. Link keys are a generalization of keys. There exist several algorithms to do reasoning with keys in description logics including [26, 27]. However, link keys cannot be reduced to keys and these algorithms cannot be directly used to do reasoning with link keys. Link keys can be also expressed as logical rules. There exist many algorithms that allows to do reasoning with rules in description logics among them [35]. However, it leads to undecidability results. There also exist decentralized algorithms for reasoning with ontological correspondences including [42], however, this algorithm is not sound when the correspondences are equipped with the standard semantics and it cannot be extended to deal with inverse roles. The next chapter will introduce a new family of description logic, where the description logic $\mathcal{ALC}$ is extended with individual equalities, inequalities, and link keys. Moreover, it provides a sound and complete reasoning algorithm for this new logic.

Chapter 4

A 2EXPTIME tableau algorithm for reasoning in the description logic $\mathcal{ALC}$ with link keys and individual equalities

4.1 Introduction

Link keys can be expressed as logical axioms, and, together with other kinds of knowledge such as ontologies and ontology alignments, may entail new link keys. Checking the entailment of new link keys necessitates to perform reasoning. Reasoning with link keys requires representing the knowledge enriched with link keys in a precise and clear manner. For this aim, in this chapter we introduce the description logic $\mathcal{ALC}+\mathcal{LK}$ which extends $\mathcal{ALC}$ with link keys and individual equalities. Individual equalities are a requirement of the presence of link keys.

We show that link key entailment can be reduced to consistency checking without the need of introducing the negation of link keys. This means to decide if a link key λ is entailed by an ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ consisting of a set $\mathcal{A}$ of assertional axioms, a set $\mathcal{T}$ of terminological axioms, and a set $\mathcal{LK}$ of link key axioms, is equivalent to checking if $\mathcal{O}$ with the negation of λ, represented as a set of assertional axioms, is inconsistent, i.e. it does not have a model. Based on this reduction we present, in this chapter, an algorithm to decide the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. This algorithm is a tableau-based algorithm which extends the standard tableau algorithm for reasoning in $\mathcal{ALC}$ [50] by adding completion rules to handle link keys and equalities. Moreover, we investigate the theoretical properties of the algorithm. We show that this algorithm terminates and it is sound and complete. For proving the soundness of the algorithm, we use unravelled interpretations because the canonical $\mathcal{ALC}$ interpretations may not satisfy link keys.

The remainder of this chapter is organized as follows. Section 4.2 provides gives the definitions required to understand the syntax and semantics of $\mathcal{ALC}+\mathcal{LK}$. Section 4.3 reduces link keys entailment to link key consistency checking. Section 4.4 presents the tableau algorithm for the description logic $\mathcal{ALC}+\mathcal{LK}$. Section 4.5 provides examples of its use. Section 4.6 gives and proves the theoretical properties of the algorithm. Finally, we conclude the chapter in Section 4.7.

4.2 The description logic $\mathcal{ALC}$+$\mathcal{LK}$

The description logic $\mathcal{ALC}$+$\mathcal{LK}$ extends $\mathcal{ALC}$ with link keys made up of $\mathcal{ALC}$ concepts and role names and individual equalities. The syntax and semantics of the description logic $\mathcal{ALC}$+$\mathcal{LK}$ extends the syntax and semantics of $\mathcal{ALC}$ as defined below.

Definition 7 (Syntax of $\mathcal{ALC}$+$\mathcal{LK}$)**.** *Let* **C***,* **R** *and* **I** *be non-empty sets of* concept names*,* role names *and* individuals*, respectively. The syntax of $\mathcal{ALC}$+$\mathcal{LK}$ extends the one given for $\mathcal{ALC}$ in Definition 1 of Chapter 2 by equality and inequality assertions and $\mathcal{ALC}$ link key expression. An equality assertion has the form $a \approx b$ and inequality assertions has the form $a \not\approx b$ where a, b are individuals in* **I***. An $\mathcal{ALC}$* link key *(simply called link key) is an expression of the form*

$$(\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\} \text{ linkkey } \langle C, D\rangle)$$

such that $\langle C, D\rangle$ is a pair of $\mathcal{ALC}$ concepts and $\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\}$ is a non-empty sequence of pairs of role names in **R***. An* LKBox *is a finite set of link keys.*

A triple $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$, where $\mathcal{T}$ is a TBox, $\mathcal{A}$ is an ABox extended with equality and inequality assertions and $\mathcal{LK}$ is an LKBox, is called an $\mathcal{ALC}$+$\mathcal{LK}$ ontology.

By abuse of notation, we will write

$$(\{\langle P_i, Q_i\rangle\}_{i=1}^{n} \text{ linkkey } \langle C, D\rangle)$$

instead of

$$(\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\} \text{ linkkey } \langle C, D\rangle).$$

Below we define the semantics of $\mathcal{ALC}$+$\mathcal{LK}$.

Definition 8 (Semantics of $\mathcal{ALC}$+$\mathcal{LK}$)**.** *An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK}\rangle$ if $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T}\rangle$, $\mathcal{I}$ is a model of every equality and inequality assertion in $\mathcal{A}$, and $\mathcal{I}$ is a model of the LKBox $\mathcal{LK}$. $\mathcal{I}$ is a model of $\langle \mathcal{A}, \mathcal{T}\rangle$ if satisfies every role and concept assertion in $\mathcal{A}$, every GCI in $\mathcal{T}$ as defined in Definition 5 of Chapter 2. $\mathcal{I}$ is a model of every equality and inequality assertion in $\mathcal{A}$ if it satisfy as well every equality and inequality assertion in $\mathcal{A}$:*

$$a \approx b \text{ if } a^{\mathcal{I}} = b^{\mathcal{I}}$$
$$a \not\approx b \text{ if } a^{\mathcal{I}} \neq b^{\mathcal{I}}$$

and $\mathcal{I}$ is a model of the LKBox $\mathcal{LK}$ if $\mathcal{I}$ satisfies every link key in $\mathcal{LK}$.

An interpretation $\mathcal{I}$ satisfies a link key $(\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ linkkey $\langle C, D\rangle)$, which will be denoted by $\mathcal{I} \models (\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ linkkey $\langle C, D\rangle)$, if

$$\forall \delta, \eta, x_1, \cdots, x_n \in \Delta^{\mathcal{I}},$$
$$\delta \in C^{\mathcal{I}} \wedge \eta \in D^{\mathcal{I}} \wedge \bigwedge_{1 \leq i \leq n} ((\delta, x_i) \in P_i^{\mathcal{I}} \wedge (\eta, x_i) \in Q_i^{\mathcal{I}}) \Rightarrow \delta = \eta$$

An ontology $\mathcal{O}$ is consistent *if there exists a model of $\mathcal{O}$, i.e. there is a model of $\mathcal{T}$, $\mathcal{A}$ and $\mathcal{LK}$. An ontology $\mathcal{O}$* entails *a GCI, an ABox assertion or a link key α, written $\mathcal{O} \models \alpha$, if every model of $\mathcal{O}$ satisfies α.*

We use $|S|$ to denote the cardinality of a set S. Given an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, we denote by $\mathsf{sub}(\mathcal{O}) = \mathsf{sub}(\mathcal{A}, \mathcal{T}, \mathcal{LK})$ the set of all sub-concepts occurring in $\mathcal{A}, \mathcal{T}$ and $\mathcal{LK}$. The size of an ontology $\mathcal{O}$ is denoted by $||\mathcal{O}|| = ||\mathcal{A}|| + ||\mathcal{T}|| + ||\mathcal{LK}||$ where $||\mathcal{A}||$ is the size of all assertions, $||\mathcal{T}||$ the size of all GCIs and $||\mathcal{LK}||$ the size of all link keys. It holds that $|\mathsf{sub}(\mathcal{O})|$ is polynomially bounded by $||\mathcal{O}||$ since if a concept is represented as string then a sub-concept is a substring.

Finally, given two individuals s, t in $\mathbf{I}$, we define the label of s as $L(s) = \{C \in \mathsf{sub}(\mathcal{O}) \mid C(s) \in \mathcal{A}\}$ and the label of $\langle s, t \rangle$ as $L(s,t) = \{R \in \mathbf{R} \mid R(s,t) \in \mathcal{A}\}$. We assume hereafter, without loss of generality, that the individuals of all ABoxes are labelled in this way.

4.3 Reduction of ontology entailment to ontology consistency

In $\mathcal{ALC}$+$\mathcal{LK}$, ontology entailment of GCIs, concept assertions, equality and inequality statements, and link keys is reducible to ontology consistency checking. Indeed, given an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, two concepts C, D and two individuals a, b:

$$\mathcal{O} \models C \sqsubseteq D \text{ iff } \langle \mathcal{A} \cup \{(C \sqcap \neg D)(x)\}, \mathcal{T}, \mathcal{LK} \rangle \text{ is inconsistent}$$
$$\mathcal{O} \models C(a) \text{ iff } \langle \mathcal{A} \cup \{\neg C(a)\}, \mathcal{T}, \mathcal{LK} \rangle \text{ is inconsistent}$$
$$\mathcal{O} \models a \approx b \text{ iff } \langle \mathcal{A} \cup \{a \not\approx b\}, \mathcal{T}, \mathcal{LK} \rangle \text{ is inconsistent}$$
$$\mathcal{O} \models a \not\approx b \text{ iff } \langle \mathcal{A} \cup \{a \approx b\}, \mathcal{T}, \mathcal{LK} \rangle \text{ is inconsistent}$$

where x is a new individual not present in $\mathcal{O}$. Notice that ontology entailment of role assertions may require considering negation of roles, which go beyond $\mathcal{ALC}$+$\mathcal{LK}$ expressivity.

This result can be extended to link keys. It is not necessary to express link key negation, but sufficient to provide an ABox witnessing this negation. Lemma 1 below proves that link key entailment can be reduced to consistency checking: given a link key λ, $\mathcal{O} \models \lambda$ if and only if $\langle \mathcal{A} \cup \mathcal{A}', \mathcal{T}, \mathcal{LK} \rangle$ is inconsistent, where $\mathcal{A}'$ represents the negation of λ.

Lemma 1 (Reduction of ontology entailment to consistency)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. It holds that*

$$\mathcal{O} \models (\{\langle P_i, Q_i \rangle\}_{i=1}^{n} \ \mathsf{linkkey} \ \langle C, D \rangle) \textit{ iff } \langle \mathcal{A} \cup \mathcal{A}', \mathcal{T}, \mathcal{LK} \rangle \textit{ is inconsistent}$$

with $\mathcal{A}' = \{C(x), D(y), x \not\approx y\} \cup \bigcup_{i=1}^{n} \{P_i(x, z_i), Q_i(y, z_i)\}$ and $x, y, z_1, \cdots, z_n$ are new individuals not present in $\mathcal{O}$.

Proof. Let $\lambda = (\{\langle P_i, Q_i \rangle\}_{i=1}^{n} \ \mathsf{linkkey} \ \langle C, D \rangle)$. Assume first that $\mathcal{O} \models \lambda$. Let us show that $\mathcal{O}' = \langle \mathcal{A} \cup \mathcal{A}', \mathcal{T}, \mathcal{LK} \rangle$ is inconsistent. By contradiction, assume that $\mathcal{O}'$ has a model $\mathcal{I}$. Since $\mathcal{A} \subseteq \mathcal{A} \cup \mathcal{A}'$, then $\mathcal{I}$ must be a model of $\mathcal{O}$ too. Moreover, since $\mathcal{I}$ is a model of $\mathcal{O}'$, $\mathcal{I}$ must be a model of $\mathcal{A}'$, which means that $x^{\mathcal{I}} \in C^{\mathcal{I}}$, $y^{\mathcal{I}} \in D^{\mathcal{I}}$, $\langle x^{\mathcal{I}}, z_i^{\mathcal{I}} \rangle \in P_i^{\mathcal{I}}$, $\langle y^{\mathcal{I}}, z_i^{\mathcal{I}} \rangle \in Q_i^{\mathcal{I}}$ and $x^{\mathcal{I}} \neq y^{\mathcal{I}}$. This implies that $\mathcal{I} \not\models \lambda$. Thus, we have a model $\mathcal{I}$ of $\mathcal{O}$ such that $\mathcal{I} \not\models \lambda$. Therefore, $\mathcal{O} \not\models \lambda$, which contradicts the assumption.

Assume now that $\mathcal{O} \not\models \lambda$. Let us show that $\mathcal{O}' = \langle \mathcal{A} \cup \mathcal{A}', \mathcal{T}, \mathcal{LK} \rangle$ is consistent. Since $\mathcal{O} \not\models \lambda$, then there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models \mathcal{O}$ and $\mathcal{I} \not\models \lambda$ (otherwise λ would be entailed). Since $\mathcal{I} \not\models \lambda$, by the semantics of link keys, there exists $\delta, \delta', \delta_1, \ldots, \delta_n \in \Delta^{\mathcal{I}}$ such that $\delta \in C^{\mathcal{I}}$, $\delta' \in D^{\mathcal{I}}$, $(\delta, \delta_1) \in P_1^{\mathcal{I}}, (\delta', \delta_1) \in Q_1^{\mathcal{I}}, \ldots, (\delta, \delta_n) \in P_n^{\mathcal{I}}, (\delta', \delta_n) \in Q_n^{\mathcal{I}}$ and $\delta \neq \delta'$. Let us extend $\mathcal{I}$ by defining $x^{\mathcal{I}} = \delta$, $y^{\mathcal{I}} = \delta'$, $z_1^{\mathcal{I}} = \delta_1, \ldots, z_n^{\mathcal{I}} = \delta_n$. Then, $\mathcal{I}$ is a model of $\mathcal{A}'$. $\mathcal{I}$ is still a model of $\mathcal{O}$. Therefore, $\mathcal{I}$ is a model of $\mathcal{O}'$ and, thus, $\mathcal{O}'$ is consistent. □

Thanks to Lemma 1, ontology entailment in $\mathcal{ALC}+\mathcal{LK}$ can be reduced to ontology consistency. The following section describes a tableau algorithm for checking the consistency of an ontology in $\mathcal{ALC}+\mathcal{LK}$.

4.4 Tableau algorithm for $\mathcal{ALC}+\mathcal{LK}$

The algorithm to decide if an ontology with link keys $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ is consistent, starts with $\mathcal{A}_0$ and applies the completion rules listed in Figure 4.1, guided by $\mathcal{T}$ and $\mathcal{LK}$. The completion rules generate new ABoxes. If no more rule is applicable to a generated ABox and this ABox does not contain any obvious contradiction (called clash) then there exists a model of $\mathcal{O}_0$ that can be built from the ABox, otherwise no model exist. This algorithm is based on the standard tableau algorithm for reasoning in $\mathcal{ALC}$ [50] to which we have added specific completion rules for dealing with link keys.

More precisely, we use $\mathbf{A}$ to denote a set of ABoxes and $\langle \mathbf{A}, \mathcal{T}, \mathcal{LK} \rangle$ is a *generalised ontology* to be used by the method. At the beginning, $\mathbf{A}$ is initialised with $\mathbf{A}_0 = \{\mathcal{A}_0\}$. $\langle \mathbf{A}, \mathcal{T}, \mathcal{LK} \rangle$ is said to be *consistent* if there exists $\mathcal{A} \in \mathbf{A}$ such that $\langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ is consistent.

The application of a completion rule transforms the set of ABoxes into another set of ABoxes. There are two types of rules: deterministic and non-deterministic rules. Each application of a deterministic rule replaces an ABox $\mathcal{A} \in \mathbf{A}_k$ by a new ABox $\mathcal{A}' \in \mathbf{A}_{k+1}$. However, the application of a non-deterministic rule replaces an ABox $\mathcal{A} \in \mathbf{A}_k$ by several new ABoxes $\mathcal{A}'_1 \ldots \mathcal{A}'_n \in \mathbf{A}_{k+1}$.

The algorithm then generates a sequence of sets of ABoxes:

$$\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \ldots \tag{4.1}$$

such that $\mathbf{A}_{k+1}$ is obtained from $\mathbf{A}_k$ by applying a completion rule. An ontology $\langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ with $\mathcal{A} \in \mathbf{A}_k$ is called a *derived ontology* from $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ and $\mathcal{A}$ a *derived ABox*. Such a derived ABox $\mathcal{A}$, and the corresponding ontology, is called *complete* if no completion rule is applicable.

4.4.1 Preprocessing

As usual, to ease the description of the completion rules, we start with a preprocessing step. All concepts occurring in the initial ontology are expressed into negation normal form (NNF), *i.e.* negation only occurs in front of concept names. Any $\mathcal{ALC}$-concept can be transformed to an equivalent one in NNF by using De Morgan's laws and the duality between existential and universal restrictions. In addition, all concepts occurring in all link keys are in NNF as well. Note that the NNF of a concept C can be computed in polynomial time in the size of C [51]. For a concept C, $\sim C$ will denote the negation normal form of $\neg C$.

4.4.2 Blocking

As for $\mathcal{ALC}$ with GCIs, blocking (cycle detection) is necessary to ensure the termination of the algorithm. We use anywhere blocking, the main idea behind this kind of is to extend the set of potential blockers for an individual than its predecessors (as it is the case for ancestor blocking). This kind of blocking allows to reduce the size of the constructed models.

Before giving the definition of blocking, we make a distinction between old and new individuals. Let $\mathcal{O}_k = \langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with set of individuals $I \neq \emptyset$. Assume that $\mathcal{O}_k$ is derived from an initial ontology $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ with a set of individuals I_0. We have $I_0 \subseteq I$. An individual $a \in I$ is called *old* if $a \in I_0$, and *new* otherwise. New individuals result from applying specific rules (in Figure 4.1, $\rightarrow_\exists$ is the only such rule). We will write $I = I_{old} \uplus I_{new}$ where $I_{old} = I_0$ and $I_{new} = I \setminus I_0$. In particular, $\mathcal{O}_0$ has old individuals only, and no new individuals.

We assume that there is a total order over $I_{old} = \{s_1, \ldots, s_n\}$ with $s_i < s_j$ for all $1 \leq i < j \leq n$. If a rule adds a new individual s to an ABox, then $<$ is extended by setting $s_i < s$ for all $1 \leq i \leq n$ and $t < s$ if t was added to the ontology prior to s. By construction, $<$ is a total order over $I = I_{old} \uplus I_{new}$.

For the sake of simplicity, we assume that, if an equality assertion $x \approx y$ or an inequality assertion $x \not\approx y$ belongs to $\mathcal{A}_0$ then $x < y$. This has no impact on consistency checking because $\approx$ and $\not\approx$ are symmetric.

Definition 9 (Order and equivalence among individuals)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals I and an order relation $<$ over I. For each individual $s \in I$, we use s^+ to denote the transitive closure of s with respect to the relation $\approx$ (appearing in assertions), i.e. s^+ is the smallest set such that $s \in s^+$, and if $c \approx b \in \mathcal{A}$ or $b \approx c \in \mathcal{A}$ with some $c \in s^+$ then $b \in s^+$. The function $\mathsf{e}(s)$ associates to each individual s the smallest element of s^+ with respect to the order relation $<$.*

Below we give the definition of a blocked individual.

Definition 10 (Blocking)**.** *Let $\mathcal{O}_k = \langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ be a derived $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals $I = I_{old} \uplus I_{new}$. An individual $s \in I_{new}$ is* blocked *by an individual $t \in I_{new}$ if $t < s$ and $L(s) \subseteq L(t)$. We denote by $\mathsf{b}(s)$ the least individual (with respect to the total order $<$) that blocks s.*

Notice that only new individuals may be blocked. Also, given a blocked element $s \in I_{new}$, the existence and uniqueness of $\mathsf{b}(s)$ is guaranteed by the fact that $<$ is a finite strict total order (and, thereby, a well-order), so the set of blocking elements of s, which is not empty, has a least element in $<$ which is unique. The following lemma proves that $\mathsf{b}(s)$ is always non blocked.

Lemma 2. *Let $\mathcal{O}_k = \langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ be a derived $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals I. If $s \in I$ is a blocked individual then $\mathsf{b}(s)$ is not blocked.*

Proof. By contradiction, assume that $\mathsf{b}(s)$ is blocked by an individual $t \in I$. Then, $t < \mathsf{b}(s)$ and $L(\mathsf{b}(s)) \subseteq L(t)$. Since s is blocked by $\mathsf{b}(s)$, we have $\mathsf{b}(s) < s$ and $L(s) \subseteq L(\mathsf{b}(s))$. Hence, $t < \mathsf{b}(s) < s$ and $L(s) \subseteq L(t)$, which contradicts the definition of $\mathsf{b}(s)$. □

4.4.3 Clashes

Clashes are atomic contradictions. Given an ontology with link keys $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, we will say that $\mathcal{A}$ contains a clash if one of the two following situations occurs:

- **$\neg$-clash**: $\{A(s), \neg A(s)\} \subseteq \mathcal{A}$ for some individual name s and a concept name A, or
- **$\not\approx$-clash**: $\{x \not\approx y\} \subseteq \mathcal{A}$ with $x \in y^+$ for some individuals x, y.

If $\mathcal{A}$ contains no clash, we say that $\mathcal{A}$, and $\mathcal{O}$, is *clash-free*.

The case when $\{\bot(s)\} \subseteq \mathcal{A}$, for some individual s, will be considered a $\neg$-clash too (implicitly, $\{\bot \sqsubseteq \neg\top\} \subseteq \mathcal{T}$ and $\top(t) \in \mathcal{A}$ for all t). We will write $\mathcal{A} \rightarrow \neg$-clash and $\mathcal{A} \rightarrow \not\approx$-clash if $\mathcal{A}$ contains, respectively, a $\neg$-clash or a $\not\approx$-clash.

4.4.4 Completion rules

Completion rules transform the ABox of a generalised ontology. They leave the TBox and LKBox unchanged. This transformation is monotonic, *i.e.* it only adds new assertions and never removes anything from the ontology.

Figure 4.1 shows the list of completion rules of the algorithm. They are standard completion rules for reasoning in $\mathcal{ALC}$ together with three more rules to deal with link keys ($\rightarrow_{\text{chooseLK1}}$, $\rightarrow_{\text{chooseLK2}}$ and $\rightarrow_{\text{LK}}$) and a rule to handle equality ($\rightarrow_{\approx}$). The $\rightarrow_{\text{LK}}$ rule translates the semantics of link keys. The $\rightarrow_{\text{chooseLK1}}$ and $\rightarrow_{\text{chooseLK2}}$ rules make it explicit whether two individuals a and b that satisfy the condition of a link key should be set as equal or not. Certainly, given an interpretation $\mathcal{I}$, the absence of an assertion $C(a)$ (resp. $D(b)$) from an ontology does not necessarily imply that $a^{\mathcal{I}} \notin C^{\mathcal{I}}$ (resp. $b^{\mathcal{I}} \notin D^{\mathcal{I}}$). For this purpose, we need to add $\sim C(a)$ (resp. $\sim D(b)$) explicitly.

Contrary to [52], the $\rightarrow_{\approx}$ rule does not remove any assertion from ABoxes. It just makes $L(x) = L(y)$, $L(x, z) = L(y, z)$, $L(z, x) = L(z, y)$ for some individual z if $x \approx y$ belongs to the ABox $\mathcal{A}$.

A derived ABox is *closed* if it is either complete or contains a clash. A generalised ontology $\langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ is called *closed* if each $\mathcal{A} \in \mathbf{A}_k$ is closed. A closed generalised ontology $\langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ is called *successful* if there exists $\mathcal{A} \in \mathbf{A}_k$ which is complete and clash-free.

At this point, we have all the necessary elements to present the algorithm for checking ontology consistency. Algorithm 1 below returns YES if it builds a successful generalised ontology $\langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ from a generalised ontology $\langle \{\mathcal{A}_0\}, \mathcal{T}, \mathcal{LK} \rangle$, and NO otherwise.

Algorithm 1: Tableau algorithm for $\mathcal{ALC}$+$\mathcal{LK}$ ontology consistency checking

Input : An $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$
Output: Consistency of $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$

```
Initialize a set of ABoxes A = {A_0} ;
while there is a completion rule r in Figure 4.1 which is applicable to an individual s
  in some A ∈ A do
    Apply r to s;
if there is a clash-free ABox A ∈ A then
    return YES;
else
    return NO;
```

and completeness of the algorithm. Soundness guarantees that if the algorithm derives a successful generalised ontology $\mathcal{O}_k$ from $\mathcal{O}_0$, then $\mathcal{O}_0$ is consistent (Proposition 5).

Before proving termination, soundness and completeness of the algorithm, we illustrate it with examples.

Rule $\rightarrow_{\sqcap}$
Condition: $\mathcal{A}$ contains $(C_1 \sqcap C_2)(s)$, but it does not contain both $C_1(s)$ and $C_2(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(s), C_2(s)\}$
Rule $\rightarrow_{\sqcup}$
Condition: $\mathcal{A}$ contains $(C_1 \sqcup C_2)(s)$, but neither $C_1(s)$ nor $C_2(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C_1(s)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{C_2(s)\}$
Rule $\rightarrow_{\forall}$
Condition: $\mathcal{A}$ contains $(\forall R.C)(s)$ and $R(s,t)$, but it does not contain $C(t)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{C(t)\}$
Rule $\rightarrow_{\exists}$
Condition: $\mathcal{A}$ contains $(\exists R.C)(s)$ but there is no individual name t such that $\mathcal{A}$ contains $R(s,t)$ and $C(t)$, and s is not blocked.
Action: $\mathcal{A}' := \mathcal{A} \cup \{R(s,t), C(t)\}$ where t is an individual not occurring in $\mathcal{A}$. Set $x < t$ for all individuals x in $\mathcal{A}$.
Rule $\rightarrow_{\textbf{choose}}$
Condition: $\mathcal{T}$ contains $C \sqsubseteq D$ and there is an individual name s such that $\mathcal{A}$ does contain neither $\sim C(s)$ nor $D(s)$.
Action: $\mathcal{A}' := \mathcal{A} \cup \{\sim C(s)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{D(s)\}$
Rule $\rightarrow_{\textbf{chooseLK1}}$
Condition: $\mathcal{LK}$ contains $(\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ $\mathsf{linkkey}$ $\langle C, D\rangle)$, and there exist individual names $x, y, z_1, \ldots, z_n$ such that $P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}$ for $1 \leq i \leq n$ and $\{C(x), \sim C(x)\} \cap \mathcal{A} = \emptyset$ *Action*: $\mathcal{A}' := \mathcal{A} \cup \{C(x)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{\sim C(x)\}$
Rule $\rightarrow_{\textbf{chooseLK2}}$
Condition: $\mathcal{LK}$ contains $(\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ $\mathsf{linkkey}$ $\langle C, D\rangle)$, and there exist individual names $x, y, z_1, \ldots, z_n$ such that $P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}$ for $1 \leq i \leq n$ and $\{D(y), \sim D(y)\} \cap \mathcal{A} = \emptyset$
Action: $\mathcal{A}' := \mathcal{A} \cup \{D(y)\}$, $\mathcal{A}'' := \mathcal{A} \cup \{\sim D(y)\}$
Rule $\rightarrow_{\textbf{LK}}$
Condition: $\mathcal{LK}$ contains $(\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ $\mathsf{linkkey}$ $\langle C, D\rangle)$, and there exist individual names $x, y, z_1, \ldots, z_n$ such that $C(x)$, $D(y)$, $P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}$ for $1 \leq i \leq n$, and $\mathcal{A} \cap \{x \approx y, y \approx x\} = \emptyset$
Action: $\mathcal{A}' := \mathcal{A} \cup \{x \approx y\}$ if $x < y$, and $\mathcal{A}' := \mathcal{A} \cup \{y \approx x\}$ otherwise.
Rule $\rightarrow_{\approx}$
Condition: $\mathcal{A}$ contains $y \approx x$ (with $y \neq x$), and $\Sigma \cap \mathcal{A} \neq \emptyset, \Sigma \setminus \mathcal{A} \neq \emptyset$ where Σ is one of the following sets of assertions: $\{C(x), C(y)\}$, $\{R(x,z), R(y,z)\}$, $\{R(z,x), R(z,y)\}$, for some concept C, or some individual z and some role R
Action: $\mathcal{A}' := \mathcal{A} \cup \Sigma$.

Figure 4.1 – Completion rules for $\mathcal{ALC}$+$\mathcal{LK}$.

4.5 Examples

This section provides a few examples of the use of the tableau-based algorithm described in Section 4.4. Example 2, derived from [52], illustrates a link inference. Example 3 shows the effect of the new $\rightarrow_{\mathbf{chooseLK}}$ rule. Examples 4 and 5 show how the validity and non validity of the link keys of Example 1 may be obtained. Though the language used in Example 1 is slightly more expressive, as it covers role name equivalence, but it can be rewritten as an $\mathcal{ALC}$ ontology to take these into account. Finally, Example 6 shows the effect of blocking and will be further used to illustrate the proofs of properties in Section 4.6.

Each example displays the initial entailment to check (when applicable), the initial knowledge base corresponding to the reduction of the problem to a unsatisfiability test and the application of the rules of the algorithm. Each line corresponds to the application of a rule to an ABox. It identifies the rule applied, the resulting ABox and the clashes ($\neg$, $\not\approx$) or completion ($\Box$) of the ABox.

Example 2 (Chained link generation)**.**

Entailment: $(\langle P, R\rangle \text{ linkkey } \langle C, D\rangle), (\langle Q, S\rangle \text{ linkkey } \langle E, F\rangle),$
$C(a), P(a, c), E(c), Q(c, v), D(b), R(b, d), F(d), S(d, v) \models a \approx b$

Initial knowledge base:

$$\begin{aligned}\mathcal{A}_0 &= \{C(a), P(a, c), E(c), Q(c, v), D(b), R(b, d), F(d), S(d, v), a \not\approx b\}\\ \mathcal{T} &= \emptyset\\ \mathcal{LK} &= \{(\langle P, R\rangle \text{ linkkey } \langle C, D\rangle), (\langle Q, S\rangle \text{ linkkey } \langle E, F\rangle)\}\end{aligned}$$

Algorithm:

$$\begin{array}{lll}\mathcal{A}_0 \rightarrow_{\mathbf{LK}} & \mathcal{A}_1 := \mathcal{A}_0 \cup \{c \approx d\} & \\ \mathcal{A}_1 \rightarrow_{\approx} & \mathcal{A}_2 := \mathcal{A}_1 \cup \{P(a, d), E(d), Q(d, v), R(b, c), F(c), S(c, v)\} & \\ \mathcal{A}_2 \rightarrow_{\mathbf{LK}} & \mathcal{A}_3 := \mathcal{A}_2 \cup \{a \approx b\} & \not\approx\end{array}$$

The unique closed ABox contains a clash. Hence, the entailment is valid.

Example 3 (ChooseLK in action)**.**

Entailment: $(\langle P, Q\rangle \text{ linkkey } \langle C, D\rangle), (\langle P, R\rangle \text{ linkkey } \langle C, \neg D\rangle)$
$C(a), P(a, v), P(a, w), Q(b, v), R(b, w) \models a \approx b$

Initial knowledge base:

$$\begin{aligned}\mathcal{A}_0 &= \{C(a), P(a, v), P(a, w), Q(b, v), R(b, w), a \not\approx b\}\\ \mathcal{T} &= \emptyset\\ \mathcal{LK} &= \{(\langle P, Q\rangle \text{ linkkey } \langle C, D\rangle), (\langle P, R\rangle \text{ linkkey } \langle C, \neg D\rangle)\}\end{aligned}$$

Algorithm:

$$
\begin{array}{lll}
\mathcal{A}_0 \rightarrow_{\textbf{chooseLK2}} & \mathcal{A}_{01} := \mathcal{A}_0 \cup \{D(b)\} & \\
& \mathcal{A}_{02} := \mathcal{A}_0 \cup \{\neg D(b)\} & \\
\mathcal{A}_{01} \rightarrow_{\textbf{LK}} & \mathcal{A}_{03} := \mathcal{A}_{01} \cup \{a \approx b\} & \not\approx \\
\mathcal{A}_{02} \rightarrow_{\textbf{LK}} & \mathcal{A}_{04} := \mathcal{A}_{02} \cup \{a \approx b\} & \not\approx
\end{array}
$$

All closed ABoxes contain a clash. Hence, the entailment is valid.

Example 4 (Link key inference)**.** *This is the inference of the link key (1.8) given in Example 1 (the concepts and role names correspond to the initials of those of the example). The example is not expressed in $\mathcal{ALC}$ because it contains role equivalence statements. However, an equivalent $\mathcal{ALC}$+$\mathcal{LK}$ ontology may be obtained through rewriting the ontology. Here, we encode it by duplicating the ABox statements containing the equivalent properties.*

Entailment: $(\langle C, C\rangle, \langle T, T\rangle \text{ linkkey } \langle W, W\rangle), N \sqsubseteq W, T \equiv T', C \equiv A, N \sqsupseteq E \sqcap \exists L.P \models (\langle C, A\rangle, \langle T, T'\rangle \text{ linkkey } \langle N, E \sqcap \exists L.P\rangle)$

Initial knowledge base:

$$
\begin{aligned}
\mathcal{A}_0 = \{&N(d), C(d,v), T(d,w), A(d,v), T'(d,w), (E \sqcap \exists L.P)(b), A(b,v),\\
&T'(b,w), C(b,v), T(b,w), d \not\approx b\}\\
\mathcal{T} = \{&N \sqsubseteq W, N \sqsupseteq E \sqcap \exists L.P\}\\
\mathcal{LK} = \{&(\langle C, C\rangle, \langle T, T\rangle \text{ linkkey } \langle W, W\rangle)\}
\end{aligned}
$$

Algorithm:

$$
\begin{array}{lll}
\mathcal{A}_0 \rightarrow_{\sqcap} & \mathcal{A}_1 := \mathcal{A}_0 \cup \{E(b), (\exists L.P)(b)\} & \\
\mathcal{A}_1 \rightarrow_{\exists} & \mathcal{A}_2 := \mathcal{A}_1 \cup \{L(b, v'), P(v')\} & \\
\mathcal{A}_2 \rightarrow_{\textbf{choose}} & \mathcal{A}_{21} := \mathcal{A}_2 \cup \{(\neg E \sqcup \forall L.\neg P)(b)\} & \\
& \mathcal{A}_{22} := \mathcal{A}_2 \cup \{N(b)\} & \\
\mathcal{A}_{21} \rightarrow_{\sqcup} & \mathcal{A}_{211} := \mathcal{A}_{21} \cup \{(\neg E)(b)\} & \neg \\
& \mathcal{A}_{212} := \mathcal{A}_{21} \cup \{(\forall L.\neg P)(b)\} & \\
\mathcal{A}_{212} \rightarrow_{\forall} & \mathcal{A}_{213} := \mathcal{A}_{212} \cup \{(\neg P)(v')\} & \neg \\
\mathcal{A}_{22} \rightarrow_{\textbf{choose}} & \mathcal{A}_{221} := \mathcal{A}_{22} \cup \{\neg N(d)\} & \neg \\
& \mathcal{A}_{222} := \mathcal{A}_{22} \cup \{W(d)\} & \\
\mathcal{A}_{222} \rightarrow_{\textbf{choose}} & \mathcal{A}_{2221} := \mathcal{A}_{222} \cup \{\neg N(b)\} & \neg \\
& \mathcal{A}_{2222} := \mathcal{A}_{222} \cup \{W(b)\} & \\
\mathcal{A}_{2222} \rightarrow_{\textbf{LK}} & \mathcal{A}_{2223} := \mathcal{A}_{2222} \cup \{d \approx b\} & \not\approx
\end{array}
$$

All closed ABoxes contain a clash. Hence, the entailment is valid.

Example 5 (Link key non inference)**.** *This is the non-inference of the link key (1.2) in Example 1. The same comments as in Example 4 apply. For readability, we adopted an ABox numbering scheme different from that of other examples in which only the path leading to a complete and clash-free ABox is numbered.*

Entailment: $(\langle C, C\rangle, \langle T, T\rangle \text{ linkkey } \langle W, W\rangle), N \sqsubseteq W, T \equiv T', C \equiv A, N \sqsupseteq E \sqcap \exists L.P \models (\langle C, A\rangle, \langle T, T'\rangle \text{ linkkey } \langle N, E\rangle)$

Initial knowledge base:

$$\begin{aligned}\mathcal{A}_0 &= \{N(d), C(d,v), T(d,w), A(d,v), T'(d,w), E(b), A(b,v), T'(b,w), C(b,v),\\ &\qquad T(b,w), d \not\approx b\}\\ \mathcal{T} &= \{N \sqsubseteq W, N \sqsupseteq E \sqcap \exists L.P\}\\ \mathcal{LK} &= \{(\langle C, C\rangle, \langle T, T\rangle \text{ linkkey } \langle W, W\rangle)\}\end{aligned}$$

Algorithm:

$\mathcal{A}_0 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_* := \mathcal{A}_0 \cup \{\neg N(d)\}$	$\neg$
	$\mathcal{A}_1 := \mathcal{A}_0 \cup \{W(d)\}$	
$\mathcal{A}_1 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_* := \mathcal{A}_1 \cup \{N(b)\}$	
	$\mathcal{A}_2 := \mathcal{A}_1 \cup \{(\neg E \sqcup \forall L.\neg P)(b)\}$	
$\mathcal{A}_2 \rightarrow_{\sqcup}$	$\mathcal{A}_* := \mathcal{A}_2 \cup \{\neg E(b)\}$	$\neg$
	$\mathcal{A}_3 := \mathcal{A}_2 \cup \{(\forall L.\neg P)(b)\}$	
$\mathcal{A}_3 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_4 := \mathcal{A}_3 \cup \{\neg N(b)\}$	
	$\mathcal{A}_* := \mathcal{A}_3 \cup \{W(b)\}$	
$\mathcal{A}_4 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_* := \mathcal{A}_4 \cup \{\neg N(v)\}$	
	$\mathcal{A}_5 := \mathcal{A}_4 \cup \{W(v)\}$	
$\mathcal{A}_5 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_6 := \mathcal{A}_5 \cup \{N(v)\}$	
	$\mathcal{A}_* := \mathcal{A}_5 \cup \{(\neg E \sqcup \forall L.\neg P)(v)\}$	
$\mathcal{A}_6 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_* := \mathcal{A}_6 \cup \{\neg N(w)\}$	
	$\mathcal{A}_7 := \mathcal{A}_6 \cup \{W(w)\}$	
$\mathcal{A}_7 \rightarrow_{\textbf{choose}}$	$\mathcal{A}_8 := \mathcal{A}_7 \cup \{N(w)\}$	
	$\mathcal{A}_* := \mathcal{A}_7 \cup \{(\neg E \sqcup \forall L.\neg P)(w)\}$	
$\mathcal{A}_8 \rightarrow_{\textbf{chooseLK2}}$	$\mathcal{A}_* := \mathcal{A}_8 \cup \{W(b)\}$	
	$\mathcal{A}_9 := \mathcal{A}_8 \cup \{\neg W(b)\}$	$\square$

$\mathcal{A}_9$ is a complete and clash-free derived ABox. Hence, the entailment is invalid.

Example 6 (Knowledge base consistency)**.** *This example is particular, since it is only concerned with the consistency of a knowledge base. It is used in the remainder for illustrating the proofs.*

Initial knowledge base:

$$\begin{aligned}\mathcal{A}_0 &= \{(\exists W.(\exists R.\top \sqcap \exists P.\exists R.\top \sqcap \exists Q.\exists R.\top))(a), P(s,a), Q(a,s)\}\\ \mathcal{T} &= \emptyset\\ \mathcal{LK} &= \{(\langle R, R\rangle \text{ linkkey } \langle \top, \top\rangle)\}\end{aligned}$$

Algorithm:

$$\begin{array}{lll}
\mathcal{A}_0 \rightarrow_{\exists} & \mathcal{A}_1 := \mathcal{A}_0 \cup \{W(a,b), (\exists R.\top \sqcap \exists P.\exists R.\top \sqcap \exists Q.\exists R.\top)(b)\} & \\
\mathcal{A}_1 \rightarrow_{\sqcap} & \mathcal{A}_2 := \mathcal{A}_1 \cup \{(\exists R.\top)(b), (\exists P.\exists R.\top \sqcap \exists Q.\exists R.\top)(b)\} & \\
\mathcal{A}_2 \rightarrow_{\sqcap} & \mathcal{A}_3 := \mathcal{A}_2 \cup \{(\exists P.\exists R.\top)(b), (\exists Q.\exists R.\top)(b)\} & \\
\mathcal{A}_3 \rightarrow_{\exists} & \mathcal{A}_4 := \mathcal{A}_3 \cup \{R(b,c), \top(c)\} & \\
\mathcal{A}_4 \rightarrow_{\exists} & \mathcal{A}_5 := \mathcal{A}_4 \cup \{P(b,d), (\exists R.\top)(d)\} & b \text{ blocks } d \\
\mathcal{A}_5 \rightarrow_{\exists} & \mathcal{A}_6 := \mathcal{A}_5 \cup \{Q(b,e), (\exists R.\top)(e)\} & b \text{ blocks } e \ \square
\end{array}$$

The unique closed ABox is complete and clash-free. Hence, the initial knowledge base is consistent.

Figure 4.2 (p. 44) displays the derived ABox corresponding to $\mathcal{A}_6$.

4.6 Properties of the method

We establish the termination (Section 4.6.2), soundness (Section 4.6.3), completeness (Section 4.6.4) and complexity (Section 4.6.5) of the proposed method (Algorithm 1). But first, we have to introduce properties which are necessary for the proof of soundness and completeness (Section 4.6.1).

4.6.1 Some properties of derived ontologies

The following lemma shows a property of an ABox which is derived by completion rules.

Lemma 3. *Let $\mathcal{O}_k = \langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK}\rangle$ be a derived $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals $I = I_{old} \uplus I_{new}$. It holds that*

1. *If $R(a,c), S(b,c) \in \mathcal{A}_k$ and $a \neq b$ then $a, b, c \in I_{old}$.*
2. *If $(a \approx b) \in \mathcal{A}_k$ then $a, b \in I_{old}$.*
3. *If $(a \not\approx b) \in \mathcal{A}_k$ then $a, b \in I_{old}$.*

Proof. Assume first that the $\rightarrow_{\approx}$ rule was not used in the derivation of $\mathcal{A}_k$. In this case, by the behaviour of the $\rightarrow_{\exists}$ rule, the only kinds of role assertions that may be included in $\mathcal{A}_k$ are: $R(u,v)$ with $u, v \in I_{old}$, $R(u,s)$ with $u \in I_{old}$ and $s \in I_{new}$, and $R(s,t)$ with $s, t \in I_{new}$, where R is a role name. Therefore, $(*)$ if $R(u,v) \in \mathcal{A}_k$ and $v \in I_{old}$ then $u \in I_{old}$. Also, since the $\rightarrow_{\exists}$

rule always adds new individual names, we have $(**)$ if $R(u,s), S(v,s) \in \mathcal{A}_k$ and $u \neq v$ then $s \in I_{old}$. Item 1 of the lemma follows from $(*)$ and $(**)$.

Now, assume that $(x \approx y) \in \mathcal{A}_k$. If $(x \approx y) \in \mathcal{A}_0$ then $x, y \in I_{old}$. Assume that $(x \approx y) \notin \mathcal{A}_0$. Then $x \approx y$ was added to $\mathcal{A}$ by applying the $\rightarrow_{\mathrm{LK}}$ rule. This means that there are $C, D, P_1, Q_1, z_1, \ldots, P_n, Q_n, z_n$ such that $(\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ $\mathsf{linkkey}$ $\langle C, D\rangle), P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}_k$. The same argument as used above allows to conclude that $x, y \in I_{old}$.

Assume now that the $\rightarrow_{\approx}$ rule was used in the derivation of $\mathcal{A}_k$. Imagine that this derivation was

$$\mathcal{A}_0 \rightarrow \mathcal{A}_1 \rightarrow \ldots \rightarrow \mathcal{A}_k$$

and that $\mathcal{A}_n \rightarrow_{\approx} \mathcal{A}_{n+1}$ (for $0 \leq n \leq k-1$) was the first application of the $\rightarrow_{\approx}$ rule. As before, the only role assertions that $\mathcal{A}_n$ may include are: $R(u,v)$ with $u, v \in I_{old}$, $R(u,s)$ with $u \in I_{old}$ and $s \in I_{new}$, and $R(s,t)$ with $s, t \in I_{new}$ where R is a role name. Also, if $(x \approx y) \in \mathcal{A}_n$ then $x, y \in I_{old}$. By the behaviour of the $\rightarrow_{\approx}$ rule, the same holds in $\mathcal{A}_{n+1}$. Then, the same holds in $\mathcal{A}_k$ too, and the same argument used before proves Item 2 of the lemma.

Finally, since no completion rule adds an inequality assertion to a derived ontology, Item 3 holds too. □

Lemma 4 is a consequence of Lemma 3 simply stating that the $\rightarrow_{\mathrm{LK}}$ rule can only be applied to individuals of $\mathcal{A}_0$.

Lemma 4. *Let $\mathcal{O}_k = \langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK}\rangle$ be a derived $\mathcal{ALC}$+$\mathcal{LK}$ ontology with set of individuals $I = I_{old} \uplus I_{new}$. If there are distinct individuals $x, y, z_1, \ldots, z_n \in I$ with $C(x), D(y), P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}_k$ for $1 \leq i \leq n$, then $x, y, z_1, \ldots, z_n \in I_{old}$.*

Proof. This is a direct consequence of Lemma 3, more precisely of the proof of Item 2. □

This lemma may seem surprising. It owes to the fact that, contrary to constraints such as role-value-maps [53], link keys work backwards: they take advantage of role value equality to identify role bearers. Role-value-maps take advantage of role bearer equality to identify role values. Hence, as soon as there cannot be role equality among individuals generated by the tableau method, these individuals (in I_{new}) cannot be identified.

This does not render link keys useless: on the contrary, their role is to identify individuals among the ABox, not those generated by the method.

4.6.2 Termination

To prove termination of Algorithm 1, we need to prove that it returns YES or NO after performing a finite number of ontology transformations, *i.e.* the loop between Lines 2-3 in Algorithm 1 is finite.

Proposition 1 (Termination)**.** *Let $\mathcal{O}_0$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Algorithm 1 terminates on $\mathcal{O}_0$.*

Proof. There are three factors that can affect the termination of Algorithm 1: the generation of new ABoxes by the non deterministic rules ($\rightarrow_{\sqcup}$, $\rightarrow_{\mathrm{chooseLK1}}$, $\rightarrow_{\mathrm{chooseLK2}}$ and $\rightarrow_{\mathrm{choose}}$), the generation of new assertions by all rules and especially of new individuals by the $\rightarrow_{\exists}$ rule, and the possible non monotonically increasing behaviour of these rule application. We address the three issues.

First, Algorithm 1 adds an assertion to an ABox when a completion rule is applicable, and never removes anything from them. This behavior of Algorithm 1 is a consequence of the completion rules. Hence the ABoxes can only grow. Similarly, the number of generated ABoxes can only increase. Now let us proove that these are bounded.

Let $\mathbf{A}_k$ be a set of ABoxes built by Algorithm 1 from an $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$. It holds that each ABox $\mathcal{A} \in \mathbf{A}_k$ contains (i) the initial assertions coming from $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$, (ii) individuals $I = I_{old} \uplus I_{new}$, (iii) concept assertions $C(x)$ associated to each individual x, and (iv) role assertions $R(x, y)$ associated to two individuals x, y. Let $\ell = ||\mathcal{O}_0||$, we have $|\mathsf{sub}(\mathcal{O}_0)| \leq O(\ell)$. By the blocking condition, we have $L(d) \neq L(d')$ for all new individuals $d, d' \in I_{new}$ with $d \neq d'$. Since $L(d) \subseteq \mathsf{sub}(\mathcal{O}_0)$, we obtain $|I| \leq O(2^\ell)$. Hence, $|\mathcal{A}| \leq O(2^\ell)$ for all $\mathcal{A} \in \mathbf{A}_k$.

Finally, the number of generated ABoxes is bounded. From each ABox $\mathcal{A}$, for each individual d and each concept $C \in L(d)$ or an axiom $C \sqsubseteq D$ there is at most one new ABox that is created and added by the $\rightarrow_\sqcup$, $\rightarrow_{\text{chooseLK1}}$, $\rightarrow_{\text{chooseLK2}}$ and $\rightarrow_{\text{choose}}$ rules. Moreover, when an application of a nondeterministic rule to an individual d in $\mathcal{A}$ due to a concept $C \in L(d)$ leads to add a new ABox $\mathcal{A}'$, C no longer triggers another application by the same nondeterministic rule to the individual d in $\mathcal{A}'$ copied from $\mathcal{A}$. Therefore, the number of generated ABoxes is bounded by $|I|^{\ell \times |I|} \leq O(2^{2^\ell})$.

Hence, Algorithm 1 can only generate a generalised ontology comprising a finite number of bounded-size ABoxes and it only adds assertions and never removes anything from the generalised ontology. Therefore, Algorithm 1 terminates. □

4.6.3 Soundness

Since Algorithm 1 is a decision procedure, it is sound if it is ensured that when it returns YES the input ontology is consistent. Thus, for soundness, we have to prove that, if Algorithm 1 is able to derive a successful generalised ontology $\mathcal{O}_k = \langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK} \rangle$, then $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ has a model. For this, we will use $\mathcal{A} \in \mathbf{A}_k$ to define an interpretation $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$, and show that $\mathcal{I}$ is a model of $\mathcal{O}_0$.

As usual, derived ABoxes containing a clash do not represent models. One may define $\mathcal{I}$ by interpreting all individuals in $\mathcal{A}$ as themselves, and then, for every concept name A, $s \in A^\mathcal{I}$ iff $A(s) \in \mathcal{A}$, and, for every role name R, $\langle s, t \rangle \in R^\mathcal{I}$ iff s is not blocked and $R(s, t) \in \mathcal{A}$, or s is blocked and $R(\mathsf{b}(s), t) \in \mathcal{A}$. This simple interpretation, called the canonical interpretation [54], is used in the case of $\mathcal{ALC}$. It also builds a model of $\mathcal{O}_0$ in case of complete clash-free non blocked derived ABoxes.

It turns out that this does not work for complete clash-free blocked derived ABoxes. Indeed, it may lead to a situation where $\mathcal{I}$ does not satisfy a link key even though $\mathcal{A}$ is complete. This is illustrated by Example 7.

Example 7 (Inadequacy of the canonical interpretation)**.** *Figure 4.2 depicts the single derived ABox $\mathcal{A}_6$ at the end of Example 6. d and e are labelled with $\exists R.\top$, but they do not have R-offspring since they are blocked by b which is also labelled by $\exists R.\top$. The canonical interpretation $\mathcal{I}$ associated with such a situation would simply be defined such that $\langle d^\mathcal{I}, c^\mathcal{I} \rangle \in R^\mathcal{I}$, $\langle e^\mathcal{I}, c^\mathcal{I} \rangle \in R^\mathcal{I}$ (and of course $\langle b^\mathcal{I}, c^\mathcal{I} \rangle \in R^\mathcal{I}$). It is depicted in the left-hand side of Figure 4.3. The problem is that $\mathcal{I}$ does not satisfy $\mathcal{LK}$ because d, e and b are different, through they all share a value for role R.*

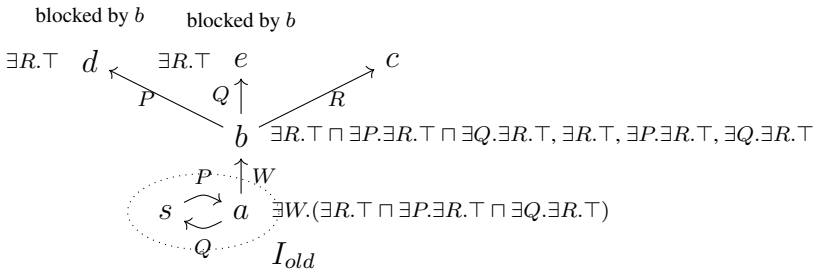


Figure 4.2 – Derived ABox corresponding to $\mathcal{A}_6$ in Example 6.

It is complete, clash-free and blocked. In order to overcome the problem of Example 7, we will consider a different interpretation $\mathcal{I}$, that we call the *unravelled interpretation*.

It is inspired from the unravelling technique used in [55] to devise a (possibly infinite) tree-like model from a derived ABox for the expressive description logic $\mathcal{SHIQ}$.

We will show that the unravelled interpretation is a model of $\mathcal{O}_0$ independently from whether the derived ABox is blocked or not.

The unravelled interpretation associates paths to individuals in $\mathcal{A}_k$. These paths are sequences of names of individuals in the derived ABox. For instance, $\mathsf{p} = \langle a, b, c\rangle$ is such a path. Its last element (c) is called its tail and we write $\mathsf{tail}(\mathsf{p}) = c$; its first element ($a$) is called its root. A path containing only one element is called a root path.

Below we give the formal definition of the unravelled interpretation.

Definition 11 (Unravelled interpretation)**.** *Let $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK}\rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology; let $\mathcal{A}_k$ be a complete and clash-free ABox derived from $\mathcal{O}_0$ with set of individuals $I = I_{old} \uplus I_{new}$. The interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ of $\mathcal{O}_0$ unravelled from $\mathcal{A}_k$ (or* unravelled *interpretation from $\mathcal{A}_k$) is defined as follows:*

1. *$\Delta^{\mathcal{I}}$ is the smallest set of* paths *built as follows:*
   (a) *$\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}_a = \langle \mathsf{e}(a)\rangle$ for each $a \in I_{old}$. In this case, $a^{\mathcal{I}} = \mathsf{p}_a$.*
   (b) *For each $\mathsf{p} \in \Delta^{\mathcal{I}}$ such that $R(\mathsf{tail}(\mathsf{p}), a) \in \mathcal{A}_k$, $a \in I_{new}$ and a is not blocked, $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, a\rangle$.*
   (c) *For each $\mathsf{p} \in \Delta^{\mathcal{I}}$ such that $R(\mathsf{tail}(\mathsf{p}), a) \in \mathcal{A}_k$, $a \in I_{new}$ and a is blocked, $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(a)\rangle$.*
2. *For each concept name A, $A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} \mid A(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k\}$*
3. *For each role name R,*

$$R^{\mathcal{I}} = \{\langle \mathsf{p}_a, \mathsf{p}_b\rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid R(a, b) \in \mathcal{A}_k\} \cup$$
$$\{\langle \mathsf{p}, \mathsf{p}'\rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, a\rangle, R(\mathsf{tail}(\mathsf{p}), a) \in \mathcal{A}_k, a \text{ is not blocked}\} \cup$$
$$\{\langle \mathsf{p}, \mathsf{p}'\rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(a)\rangle, R(\mathsf{tail}(\mathsf{p}), a) \in \mathcal{A}_k, a \text{ is blocked}\}$$

From Definition 23, individuals of I_{old} are assigned a root path and equivalent individuals are interpreted as the same root path. Each individual of I_{new} generates paths in the unravelled interpretation obtained by concatenating the path associated to their ancestor in the application

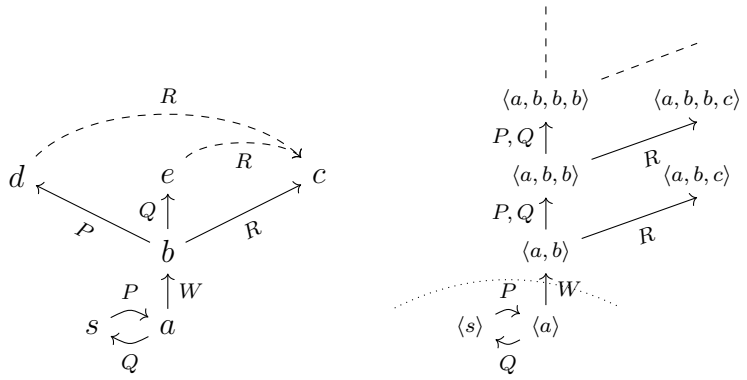


Figure 4.3 – Left: Canonical interpretation corresponding to the derived ABox of Figure 4.2.

of the $\rightarrow_\exists$ rule to its name if it is not blocked and the name of its blocking node otherwise. Hence, each path in the domain $\Delta^\mathcal{I}$ of the unravelled interpretation is rooted at a path corresponding to an old individual of the ontology, *i.e.* an individual of $\mathcal{A}_0$.

It is possible to display the unravelled interpretation as an edge-labelled directed graph such that each element of $\Delta^\mathcal{I}$ is a node and there is an edge between two nodes if the pair of nodes belongs to the interpretation of a relation. Edges are labelled by the set of roles in which the corresponding pair appears. Figure 4.3 (right) displays such a graph.

The domain of the unravelled interpretation $\Delta^\mathcal{I}$ may be infinite because, as illustrated in Example 7, paths for blocked nodes may end with one of their ancestor in the derived ABox.

Example 7 (Unravelled interpretation)**.** *If $\mathcal{I}$ is the unravelled interpretation from $\mathcal{A}_6$ of Example 6, $\mathcal{I}$ interprets d and e as the same path individual $\langle a, b, b\rangle$. In this way, the link key is satisfied by $\mathcal{I}$. The unravelled interpretation $\mathcal{I}$ is depicted at the right-hand side of Figure 4.3 as a graph. It is a tree, albeit infinite, rooted in a root node ($\langle a\rangle$). Actually, the graph of unravelled interpretations corresponds to the image of a forest made of trees whose branches extend to the sky, while underground their roots can be connected and interleaved in an arbitrary way. Notice that $\langle a, b, b\rangle, \langle a, b, b, b\rangle, \langle a, b, b, b, b\rangle \ldots$ belong to $\Delta^\mathcal{I}$, i.e. $\Delta^\mathcal{I}$ is infinite. Also note that, there is no pair of elements of $\Delta^\mathcal{I}$ have the same value for R because, at each stage of the tree, the R-value is different: $\langle a, b, c\rangle, \langle a, b, b, c\rangle, \langle a, b, b, b, c\rangle \ldots$ Hence, the link key cannot apply.*

It does not satisfy $\mathcal{LK}$ since d, e and b have a common R-value and are different. Right: Unravelled interpretation from the same derived ABox. It is infinite but satisfies $\mathcal{LK}$ since all R-values are different due to different prefixes.

The unravelled interpretation from $\mathcal{A}_k$ is not an interpretation of $\mathcal{O}_k$, as it does not interpret individuals in I_{new}, but it is an interpretation of $\mathcal{O}_0$. Proposition 5 shows that it is a model of $\mathcal{O}_0$.

The argument of the proof goes as follows: The interpretation unravelled from $\langle \mathcal{A}_k, \mathcal{T}, \mathcal{LK}\rangle$ satisfies the $\mathcal{ALC}$ ontology $\langle \mathcal{A}_0, \mathcal{T}\rangle$. By Lemma 4, link keys only apply to statements involving individuals of $\mathcal{A}_0$. Since the derived ABox $\mathcal{A}_k$ is complete, it contains the result of the application of all link keys (individuals from $\mathcal{A}_0$ cannot be blocked). Unravelling does two things: (a) merging all $\mathcal{A}_0$ individuals related by $\approx$, and (b) expanding blocked individuals into (possibly infinite) trees. Hence, in both cases, all link keys are satisfied because no different individuals satisfy the link key conditions.

Proposition 2 (Soundness)**.** *If Algorithm 1 derives a successful generalised ontology from an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O}_0$, then $\mathcal{O}_0$ is consistent.*

Proof. Let $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology and $\mathcal{O}_k = \langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK} \rangle$ be the successful generalised ontology derived from $\mathcal{O}_0$. This means that there exists a complete and clash-free ABox $\mathcal{A}_k \in \mathbf{A}_k$.

To prove the lemma, we show that the unravelled interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ from $\mathcal{A}_k$ according to Definition 23 is a model of $\mathcal{O}_0$. Recall that $I = I_{old} \uplus I_{new}$ where I_{old} is the set of the individuals in $\mathcal{O}_0$ and I is the set of all individuals in $\mathcal{O}_k$.

Apart from equalities $x \approx y \in \mathcal{A}_0$, an application of the $\rightarrow_{\mathrm{LK}}$ rule can add a new equality while no rule can remove any equality. Therefore, each transitive closure a^+ for some individual a changes monotonically, *i.e.* $a^+(\mathcal{A}_{k-1}) \subseteq a^+(\mathcal{A}_k)$ for every individual a where $a^+(\mathcal{X})$ denotes the transitive closure a^+ defined over an ABox $\mathcal{X}$ (*cf.* Definition 9). In the sequel, we write a^+ for $a^+(\mathcal{A}_k)$. We rely on the following claims:

$$a \in I_{old} \Longrightarrow a^{\mathcal{I}} = \mathsf{e}(a)^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(a)} \tag{4.2}$$

$$a \in I_{old} \Longrightarrow a^+ \subseteq I_{old} \Longrightarrow \mathsf{e}(a) \in I_{old} \tag{4.3}$$

$$a \in I_{new} \Longrightarrow a^+ = \{a\} \tag{4.4}$$

$$\mathcal{A}_k \text{ is complete and clash-free} \Longrightarrow \begin{cases} L(x) = L(\mathsf{e}(x)), \text{and} \\ L(x,y) = L(\mathsf{e}(x), \mathsf{e}(y)) \end{cases} \tag{4.5}$$

The claim (4.2) is due to Definition 23 while the claims (4.3) and (4.4) are direct consequences of Lemma 3(2). The claim (4.5) is a consequence of the non-applicability of the $\rightarrow_{\approx}$ rule.

To prove that $\mathcal{I}$ is a model of $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$, we have to prove that $\mathcal{I}$ satisfies all assertions in $\mathcal{A}_0$, all GCIs in $\mathcal{T}$ and all link keys in $\mathcal{LK}$.

Assume $a \approx b \in \mathcal{A}_0$. This implies that $a, b \in I_{old}$ and $a, b \in a^+$. By the claim (4.2), $a^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(a)}$ and $b^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(b)}$. Since, $a, b \in a^+$ then $\mathsf{e}(a) = \mathsf{e}(b)$, and we have $a^{\mathcal{I}} = b^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(a)}$. Thus $a^{\mathcal{I}} = b^{\mathcal{I}}$.

Assume that $R(a, b) \in \mathcal{A}_0$. This implies that $a, b \in I_{old}$. We have $a^{\mathcal{I}} = e(a)^{\mathcal{I}} = \mathsf{p}_{e(a)}$ and $b^{\mathcal{I}} = e(b)^{\mathcal{I}} = \mathsf{p}_{e(b)}$ due to the claim (4.2), and $R(\mathsf{e}(a), \mathsf{e}(b)) \in \mathcal{A}_k$ due to the claim (4.5). By Definition 23, $\langle \mathsf{p}_{e(a)}, \mathsf{p}_{e(b)} \rangle \in R^{\mathcal{I}}$, and thus $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$.

Assume $a \not\approx b \in \mathcal{A}_0$ with $a < b$. We have $a, b \in I_{old}$ due to Lemma 3. By the claim (4.2), we have $a^{\mathcal{I}} = \mathsf{e}(a)^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(a)}$ and $b^{\mathcal{I}} = \mathsf{e}(b)^{\mathcal{I}} = \mathsf{p}_{\mathsf{e}(b)}$. By contradiction, assume that $\mathsf{p}_{\mathsf{e}(a)} = \mathsf{p}_{\mathsf{e}(b)}$. This implies that $\mathsf{e}(a) = \mathsf{e}(b)$ and thus $b \in a^+$, which is a $\not\approx$-clash. This contradicts clash-freeness of $\mathcal{A}_k$. Therefore, $\mathsf{p}_{\mathsf{e}(a)} \neq \mathsf{p}_{\mathsf{e}(b)}$ and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$.

Assume $E(w) \in \mathcal{A}_0$. To show $w^{\mathcal{I}} \in E^{\mathcal{I}}$, we need to show a stronger claim:

$$\text{For all } \mathsf{p} \in \Delta^{\mathcal{I}}, \text{ if } C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k \text{ then } \mathsf{p} \in C^{\mathcal{I}} \tag{4.6}$$

Indeed, $E(w) \in \mathcal{A}_0$ and the claim (4.5) imply $E(\mathsf{e}(w)) \in \mathcal{A}_k$. In addition, $E(w) \in \mathcal{A}_0$ and the claim (4.3) imply that $w, \mathsf{e}(w) \in I_{old}$. By the definition of $\mathcal{I}$, there is some $\mathsf{p} \in \Delta^{\mathcal{I}}$ such that $\mathsf{p} = w^{\mathcal{I}} = \mathsf{e}(w)^{\mathcal{I}}$ and $\mathsf{tail}(\mathsf{p}) = \mathsf{e}(w)$. From the claim (5.1), we obtain $w^{\mathcal{I}} \in E^{\mathcal{I}}$. We now show the claim (5.1). Let us proceed by induction on the length of the concept C.

1. Assume that $C = A$ with a concept name A and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. We have $C^{\mathcal{I}} = A^{\mathcal{I}} = \{\mathsf{p}' \in \Delta^{\mathcal{I}} \mid A(\mathsf{tail}(\mathsf{p}')) \in \mathcal{A}_k\}$ by the definition of $\mathcal{I}$. Hence, $A(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$ implies $\mathsf{p} \in A^{\mathcal{I}}$.

2. Assume that $C = C_1 \sqcap C_2$ and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. Since $\mathcal{A}_k$ is complete then the $\rightarrow_{\sqcap}$ rule is not applicable, hence $C_1(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$, $C_2(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. By induction hypothesis, $\mathsf{p} \in C_1^{\mathcal{I}}$ and $\mathsf{p} \in C_2^{\mathcal{I}}$. Then, $\mathsf{p} \in C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = (C_1 \sqcap C_2)^{\mathcal{I}}$.

3. Assume that $C = C_1 \sqcup C_2$ and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. Since $\mathcal{A}_k$ is complete then the $\rightarrow_{\sqcap}$ rule is not applicable, hence $C_1(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$ or $C_2(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. By induction hypothesis, $\mathsf{p} \in C_1^{\mathcal{I}}$ or $\mathsf{p} \in C_2^{\mathcal{I}}$. Then, $\mathsf{p} \in C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} = (C_1 \sqcup C_2)^{\mathcal{I}}$.

4. Assume now that $C = \forall R.D$ and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$.

   Let $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. From the definition of $\mathcal{I}$, we consider the following two cases:

   - $R(\mathsf{tail}(\mathsf{p}), t) \in \mathcal{A}_k$ and t is not blocked with $t = \mathsf{tail}(\mathsf{p}')$. Since $\mathcal{A}_k$ is complete, the $\rightarrow_{\forall}$ rule is not applicable, thus $D(t) \in \mathcal{A}_k$. By induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. Hence, $\mathsf{p} \in C^{\mathcal{I}}$.
   - $R(\mathsf{tail}(\mathsf{p}), t) \in \mathcal{A}_k$ and t is blocked with $\mathsf{b}(t) = \mathsf{tail}(\mathsf{p}')$. Since $\mathcal{A}_k$ is complete, the $\rightarrow_{\forall}$ rule is not applicable, thus $D(t) \in \mathcal{A}_k$. Since $\mathsf{tail}(\mathsf{p}')$ blocks t, we have $L(t) \subseteq L(\mathsf{tail}(\mathsf{p}'))$, and thus $D(\mathsf{tail}(\mathsf{p}')) \in \mathcal{A}_k$. By induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. Hence, $\mathsf{p} \in C^{\mathcal{I}}$.

5. Assume that $C = \exists R.D$ and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. Since $\mathsf{tail}(\mathsf{p})$ is never blocked and $\mathcal{A}_k$ is complete, the $\rightarrow_{\exists}$ rule is not applicable, and thus there exists $t \in I$ such that $R(\mathsf{tail}(\mathsf{p}), t) \in \mathcal{A}_k$, $D(t) \in \mathcal{A}_k$. By claim (4.5), we have $R(\mathsf{e}(\mathsf{tail}(\mathsf{p})), \mathsf{e}(t)), D(\mathsf{e}(t)) \in \mathcal{A}_k$. By the claims (4.2), (4.4) and Definition 23, $\mathsf{e}(\mathsf{tail}(\mathsf{p})) = \mathsf{tail}(\mathsf{p})$, and thus, $R(\mathsf{tail}(\mathsf{p}), \mathsf{e}(t)), D(\mathsf{e}(t)) \in \mathcal{A}_k$. We distinguish the following two cases:

   - Assume that $\mathsf{e}(t)$ is not blocked. By the definition of $\mathcal{I}$ and $R(\mathsf{tail}(\mathsf{p}), \mathsf{e}(t)) \in \mathcal{A}_k$, there is some $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $\mathsf{tail}(\mathsf{p}') = \mathsf{e}(t)$ and $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. Moreover, since $D(\mathsf{e}(t)) \in \mathcal{A}_k$ and $\mathsf{tail}(\mathsf{p}') = \mathsf{e}(t)$, by induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. Hence, $\mathsf{p} \in C^{\mathcal{I}}$.
   - Assume that $\mathsf{e}(t)$ is blocked. According to the definition of $\mathcal{I}$ and $R(\mathsf{tail}(\mathsf{p}), \mathsf{e}(t)) \in \mathcal{A}_k$, there is some $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $\mathsf{tail}(\mathsf{p}') = \mathsf{b}(\mathsf{e}(t))$ and $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. We have $D(\mathsf{e}(t)) \in \mathcal{A}_k$ implies $D(\mathsf{b}(\mathsf{e}(t))) \in \mathcal{A}_k$. By induction hypothesis, we have $\mathsf{p}' \in D^{\mathcal{I}}$. Hence, $\mathsf{p} \in C^{\mathcal{I}}$.

6. Assume that $C = \sim D$ and $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. We have to show that $\mathsf{p} \notin D^{\mathcal{I}}$. We proceed by induction on the length of D. If D is a concept name then $D(\mathsf{tail}(\mathsf{p})) \notin \mathcal{A}_k$ since $\mathcal{A}_k$ is clash-free. By the definition of $D^{\mathcal{I}}$, $\mathsf{p} \notin D^{\mathcal{I}}$. Assume that $D = C_1 \sqcap C_2$. This implies that $\sim D = \sim C_1 \sqcup \sim C_2$. Due to completeness, $\mathcal{A}_k$ must contain either $\sim C_1(\mathsf{tail}(\mathsf{p}))$ or $\sim C_2(\mathsf{tail}(\mathsf{p}))$. By induction hypothesis, we have $\mathsf{p} \notin C_1^{\mathcal{I}}$ or $\mathsf{p} \notin C_2^{\mathcal{I}}$. Hence, $\mathsf{p} \notin C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$, and thus $\mathsf{p} \notin D^{\mathcal{I}}$. Analogously, we can prove for the case of $D = C_1 \sqcup C_2$.

   Now assume that $D = \exists R.E$. This implies that $\sim D = \forall R.\sim E$. Let $\mathsf{p}' \in \Delta^{\mathcal{I}}$ with $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. By Item 4, we have showed that $\mathsf{p} \in \sim D^{\mathcal{I}}$, and thus $\mathsf{p} \notin D^{\mathcal{I}}$. Analogously, we can prove for the case of $D = \forall R.E$.

We now show that $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$. Let $C \sqsubseteq D \in \mathcal{T}$ and $\mathsf{p} \in C^{\mathcal{I}}$. We have to show $\mathsf{p} \in D^{\mathcal{I}}$. Due to the completeness of $\mathcal{A}_k$, *i.e.* the $\rightarrow_{\text{choose}}$ rule is not applicable, we have

either $\sim C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$ or $D(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$. If $\sim C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$, then $\mathsf{p} \notin C^{\mathcal{I}}$ due to Item 6, which contradicts $\mathsf{p} \in C^{\mathcal{I}}$. Hence, $D(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$ and thus, $\mathsf{p} \in D^{\mathcal{I}}$.

We now show that $\mathcal{I}$ satisfies link keys in $\mathcal{LK}$. Assume that $\lambda = (\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ linkkey $\langle C, D\rangle) \in \mathcal{LK}$. Let us prove that $\mathcal{I}$ satisfies λ. Let $\mathsf{p}, \mathsf{q}, \mathsf{p}_1, \ldots, \mathsf{p}_n \in \Delta^{\mathcal{I}}$ such that $\mathsf{p} \in C^{\mathcal{I}}$, $\mathsf{q} \in D^{\mathcal{I}}$, and $(\mathsf{p}, \mathsf{p}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{p}_i) \in Q_i^{\mathcal{I}}$ for $1 \leq i \leq n$. We have to prove that $\mathsf{p} = \mathsf{q}$. Since $\mathcal{A}_k$ is complete, then neither the $\rightarrow_{\text{chooseLK1}}$ rule nor the $\rightarrow_{\text{chooseLK2}}$ rule may be applied, which means that $\mathcal{A}_k$ contains either $C(\mathsf{tail}(\mathsf{p}))$ or $\sim C(\mathsf{tail}(\mathsf{p}))$, and either $D(\mathsf{tail}(\mathsf{q}))$ or $\sim D(\mathsf{tail}(\mathsf{q}))$. If $\sim C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$ or $\sim D(\mathsf{tail}(\mathsf{q})) \in \mathcal{A}_k$ then $\mathsf{p} \notin C^{\mathcal{I}}$ or $\mathsf{q} \notin D^{\mathcal{I}}$ by the claim (5.1), which contradicts $\mathsf{p} \in C^{\mathcal{I}}$ or $\mathsf{q} \in D^{\mathcal{I}}$. Therefore, $C(\mathsf{tail}(\mathsf{p})) \in \mathcal{A}_k$, $D(\mathsf{tail}(\mathsf{q})) \in \mathcal{A}_k$. We consider the following cases:

Assume that $\mathsf{tail}(\mathsf{p}_i) \in I_{old}$ for all $1 \leq i \leq n$. We obtain $\mathsf{tail}(\mathsf{p}), \mathsf{tail}(\mathsf{q}) \in I_{old}$, $P_i(\mathsf{tail}(\mathsf{p}), \mathsf{tail}(\mathsf{p}_i))$, $Q_i(\mathsf{tail}(\mathsf{q}), \mathsf{tail}(\mathsf{p}_i)) \in \mathcal{A}_k$ from the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{p}_i) \in Q_i^{\mathcal{I}}$. Since $\mathcal{A}_k$ is complete, the satisfaction of the link key implies $\mathsf{tail}(\mathsf{p}) = \mathsf{tail}(\mathsf{q})$. Hence, $\mathsf{p}_{\mathsf{tail}(\mathsf{p})} = \mathsf{p}_{\mathsf{tail}(\mathsf{q})}$. From $\mathsf{p} = \mathsf{p}_{\mathsf{tail}(\mathsf{p})}$ and $\mathsf{q} = \mathsf{p}_{\mathsf{tail}(\mathsf{q})}$, we obtain $\mathsf{p} = \mathsf{q}$.

Assume that $\mathsf{tail}(\mathsf{p}_i) \in I_{new}$ for some $1 \leq i \leq n$. From the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{p}_i) \in Q_i^{\mathcal{I}}$, we obtain $\mathsf{p}_i = \langle \mathsf{p}, \mathsf{tail}(\mathsf{p}_i)\rangle = \langle \mathsf{q}, \mathsf{tail}(\mathsf{p}_i)\rangle$. Thus, $\mathsf{p} = \mathsf{q}$. □

4.6.4 Completeness

Since Algorithm 1 is a decision procedure, Algorithm 1 is complete if it is ensured that when the initial ontology is consistent, the algorithm returns YES. Thus, for completeness, we have to prove that if the initial ontology $\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK}\rangle$ is consistent then Algorithm 1 is able to build a successful generalised ontology $\langle \mathbf{A}, \mathcal{T}, \mathcal{LK}\rangle$.

Proposition 3 (Completeness)**.** *If an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O}_0$ is consistent, then Algorithm 1 derives a successful generalised ontology from $\mathcal{O}_0$.*

Proof. Assume that $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK}\rangle$ and that $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ is a model of $\mathcal{O}_0$. We show that Algorithm 1 can build a generalised ontology $\langle \mathbf{A}_k, \mathcal{T}, \mathcal{LK}\rangle$ with a complete and clash-free ABox $\mathcal{A}_k \in \mathbf{A}_k$.

We maintain a function π which associates each individual s of an ABox $\mathcal{A}_k \in \mathbf{A}_k$ to an individual in $\Delta^{\mathcal{I}}$, *i.e.* $\pi(s) \in \Delta^{\mathcal{I}}$.

After applying a completion rule, we must update π in such a way that π satisfies the following conditions:

$$C(s) \in \mathcal{A}_k \text{ implies } \pi(s) \in C^{\mathcal{I}} \text{ or } \pi(\mathsf{b}(s)) \in C^{\mathcal{I}} \tag{4.7}$$

$$R(s,t) \in \mathcal{A}_k \text{ implies } \langle \pi(s), \pi(t)\rangle \in R^{\mathcal{I}} \text{ or } \langle \pi(s), \pi(\mathsf{b}(t))\rangle \in R^{\mathcal{I}} \tag{4.8}$$

$$s \not\approx t \in \mathcal{A}_k \text{ implies } \pi(s) \neq \pi(t) \tag{4.9}$$

$$s \approx t \in \mathcal{A}_k \text{ implies } \pi(s) = \pi(t) \tag{4.10}$$

According to Proposition 1, Algorithm 1 always terminates at some $\mathbf{A}_n$. Thanks to the function π with Conditions (4.7-4.10) which helps to choose a "good" ABox $\mathcal{A}_k$ among several ABoxes $\mathbf{A}_k$ at each step $k \leq n$, we will show that there is an ABox $\mathcal{A}_n \in \mathbf{A}_n$ which is mapped to $\Delta^{\mathcal{I}}$ by π such that $\langle \mathcal{A}_n, \mathcal{T}, \mathcal{LK}\rangle$ is clash-free.

Assume that there exists such a function π. We show that $\mathcal{A}_n$ is complete and clash-free. When Algorithm 1 terminates, $\mathcal{A}_n$ must be complete. Assume that $A(s), \neg A(s) \in \mathcal{A}_n$. By Condition (4.7), we have $\pi(s) \in A^{\mathcal{I}}$ and $\pi(s) \in (\neg A)^{\mathcal{I}}$. It is not possible since $\mathcal{I}$ is a model. If

$x \not\approx x \in \mathcal{A}_n$ then $\pi(x) \neq \pi(x)$ due to Condition (4.9), which is a contradiction. Assume that $x \not\approx y \in \mathcal{A}_n$ with $x \in y^+$. This implies that $\pi(x) \neq \pi(y)$ and there are $x \approx x_1, \cdots, x_n \approx y \in \mathcal{A}_n$. From Condition (4.10), we obtain $\pi(x) = \pi(y)$ which is a contradiction. Therefore, $\mathcal{A}_n$ is clash-free.

Now, let us define π. For each $s \in I_{old}$, there is some $s^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ since $\mathcal{I}$ is a model of $\mathcal{O}_0$. We define $\pi(s) = s^{\mathcal{I}}$, and $\pi(s) = \pi(s')$ if $s \approx s' \in \mathcal{A}_0$. Let $R(s,t) \in \mathcal{A}_0$. We have $s, t \in I_{old}$, and thus $\pi(s), \pi(t)$ are defined. This implies that $\langle \pi(s), \pi(t) \rangle \in R^{\mathcal{I}}$ since $\mathcal{I}$ is a model of $\mathcal{O}_0$. For individual assertions, it holds that $s \not\approx t \in \mathcal{A}_0$ implies $\pi(s) \neq \pi(t)$ since $\mathcal{I}$ is a model of $\mathcal{O}_0$, and $s \approx t \in \mathcal{A}_0$ implies $\pi(s) = \pi(t)$ by the definition of π. Let $C(s) \in \mathcal{A}_0$. We have $s \in I_{old}$, and thus $\pi(s)$ is defined. This implies that $\pi(s) \in C^{\mathcal{I}}$ since $\mathcal{I}$ is a model of C. Therefore, Conditions (4.7-4.9) are verified for $\mathcal{A}_0$.

In the sequel, we consider each possible transformation performed by a completion rule on $\mathbf{A}_k$. Assume that there is an ABox $\mathcal{A}_k \in \mathbf{A}_k$ such that $\pi(s) \in \Delta^{\mathcal{I}}$ for each individual s occurring in $\mathcal{A}_k$, and π satisfies Conditions (4.7-4.10).

- The $\rightarrow_{\sqcap}$ rule is applied to $(C_1 \sqcap C_2)(s) \in \mathcal{A}_k$. Thus, $C_1(s), C_2(s) \in \mathcal{A}_{k+1}$. By Condition (4.7) and $\pi(s) \in \Delta^{\mathcal{I}}$, we have $\pi(s) \in (C_1 \sqcap C_2)^{\mathcal{I}}$. We obtain $\pi(s) \in C_1^{\mathcal{I}}$ and $\pi(s) \in C_2^{\mathcal{I}}$ since $\pi(s) \in (C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$. Therefore, Condition (4.7) is preserved.

- The $\rightarrow_{\exists}$ rule is applied to $\exists R.C(s) \in \mathcal{A}_k$ where s is not blocked. Thus, the rule adds an individual t and $C(t), R(s,t)$ to $\mathcal{A}_k$. Thus, $C(t), R(s,t) \in \mathcal{A}_{k+1}$. By Condition (4.7) and $\pi(s) \in \Delta^{\mathcal{I}}$, we have $\pi(s) \in (\exists R.C)^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $(\exists R.C)$, there is some $t' \in \Delta^{\mathcal{I}}$ such that $\langle \pi(s), t' \rangle \in R^{\mathcal{I}}$ and $t' \in C^{\mathcal{I}}$. If t is not blocked, we define $\pi(t) = t'$. Thus, Condition (4.7) and (4.8) are preserved. If t is blocked by $\mathsf{b}(t)$, we define $\pi(t) = \pi(\mathsf{b}(t))$. From Condition (4.8), we obtain $\langle \pi(s), \pi(t) \rangle \in R^{\mathcal{I}}$. Moreover, $L(t) \subseteq L(\mathsf{b}(t))$ implies $C(\mathsf{b}(t)) \in \mathcal{A}_{k+1}$. From Condition (4.7), it follows $\pi(\mathsf{b}(t)) = \pi(t) \in C^{\mathcal{I}}$. Hence, Conditions (4.7) and (4.8) are preserved.

- The $\rightarrow_{\forall}$ rule is applied to $\forall R.C(s) \in \mathcal{A}_k$. If s is blocked then $\forall R.C(\mathsf{b}(s)) \in \mathcal{A}_k$ and $\pi(s) = \pi(\mathsf{b}(s))$. Hence, it suffices to consider s that is not blocked. By Condition (4.7) and $\pi(s) \in \Delta^{\mathcal{I}}$, we have $\pi(s) \in (\forall R.C)^{\mathcal{I}}$. Assume that there is an individual t in $\mathcal{A}_k$ such that $R(s,t) \in \mathcal{A}_k$. In this case, the rule adds $C(t)$ to $\mathcal{A}_k$. Thus, $C(t) \in \mathcal{A}_{k+1}$. Assume that t is not blocked, by Condition (4.8) and $\pi(s) \in \Delta^{\mathcal{I}}$, we have $\langle \pi(s), \pi(t) \rangle \in R^{\mathcal{I}}$. Since $\mathcal{I}$ is a model of $\forall R.C$, we obtain $\pi(t) \in C^{\mathcal{I}}$. Thus, Condition (4.7) is preserved. Assume that t is blocked by $\mathsf{b}(t)$. We define $\pi(t) = \pi(\mathsf{b}(t))$. We have $L(t) \subseteq L(\mathsf{b}(t))$, and thus, $C(\mathsf{b}(t)) \in \mathcal{A}_{k+1}$. From Condition (4.7), it follows $\pi(\mathsf{b}(t)) = \pi(t) \in C^{\mathcal{I}}$. Hence, Condition (4.7) is preserved.

- The $\rightarrow_{\mathrm{LK}}$ rule is applied to individuals x, y, z_i with $C(x), D(y) \in \mathcal{A}_k$, $P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}_k$ for $1 \leq i \leq m$. According to Lemma 4, we have $x, y, z_i \in I_{old}$ for $1 \leq i \leq n$. Thus they are not blocked. By Condition (4.7) and (4.8), we have $\pi(x) \in C^{\mathcal{I}}$, $\pi(y) \in D^{\mathcal{I}}$, $\langle \pi(x), \pi(z_i) \rangle \in P_i^{\mathcal{I}}$ and $\langle \pi(y), \pi(z_i) \rangle \in Q_i^{\mathcal{I}}$ for $1 \leq i \leq n$.

  The $\rightarrow_{\mathrm{LK}}$ rule adds $x \approx y$ to $\mathcal{A}_k$. We obtain $(x \approx y) \in \mathcal{A}_{k+1}$. Since $\mathcal{I}$ is a model of $\mathcal{O}_0$, $\mathcal{I}$ must satisfy the link key. Hence, $x^{\mathcal{I}} = y^{\mathcal{I}}$, and thus $\pi(x) = \pi(y)$. Therefore, Condition (4.10) is preserved.

- The $\rightarrow_{\approx}$ rule is applied when $(x \approx y) \in \mathcal{A}_k$. It makes $L(x) = L(y)$ and $L(x, y) = L(\mathsf{e}(x), \mathsf{e}(y))$. This rule does not change individuals, Condition (4.9) and Condition

(4.10) are preserved. If it adds $C(x)$ to $\mathcal{A}_k$ when $C(y) \in \mathcal{A}_k$ (or vice versa) then Condition (4.7) is preserved since $\pi(x) = \pi(y)$ and $\pi(y) \in C^{\mathcal{I}}$ imply $\pi(x) \in C^{\mathcal{I}}$. If it adds $R(x, z)$ (resp. $R(z, x)$) to $\mathcal{A}_k$ when $R(y, z) \in \mathcal{A}_k$ (resp. $R(z, y)$) then Condition (4.8) is preserved since $\pi(x) = \pi(y)$ and $\langle \pi(y), \pi(z) \rangle \in R^{\mathcal{I}}$ (resp. $\langle \pi(z), \pi(y) \rangle \in R^{\mathcal{I}}$) imply $\langle \pi(x), \pi(z) \rangle \in R^{\mathcal{I}}$ (resp. $\langle \pi(z), \pi(x) \rangle \in R^{\mathcal{I}}$).

- The $\rightarrow_{\sqcup}$ rule is applied to $(C_1 \sqcup C_2)(s) \in \mathcal{A}_k$. It transforms $\mathcal{A}_k$ to $\mathcal{A}_{k+1}$ with $C_1(s) \in \mathcal{A}_{k+1}$, and adds a new ABox $\mathcal{A}'_{k+1}$ with $C_2(s) \in \mathcal{A}'_{k+1}$. By Condition (4.7) and $\pi(s) \in \Delta^{\mathcal{I}}$, we have $\pi(s) \in (C_1 \sqcup C_2)^{\mathcal{I}}$, and thus, either $\pi(s) \in C_1^{\mathcal{I}}$ or $\pi(s) \in C_2^{\mathcal{I}}$. Assume that $\pi(s) \in C_1^{\mathcal{I}}$. In this case, we choose $\mathcal{A}_{k+1}$ including s with $\pi(s) \in C_1^{\mathcal{I}}$. This implies that Condition (4.7) is preserved in $\mathcal{A}_{k+1}$. Assume that $\pi(s) \in C_2^{\mathcal{I}}$. In this case, we choose $\mathcal{A}'_{k+1}$ including $C_2(s)$.This implies that Condition (4.7) is preserved in $\mathcal{A}'_{k+1}$.

- The $\rightarrow_{\text{choose}}$ rule is applied to $(\sim C \sqcup D)(s) \in \mathcal{A}_k$ with $C \sqsubseteq D \in \mathcal{T}$. In the same way, we can choose an ABox among $\mathcal{A}_{k+1}$ and $\mathcal{A}'_{k+1}$ such that Condition (4.7) is preserved.

- the $\rightarrow_{\text{chooseLK1}}$ rule is applied to individuals x, y, z_i with $y < x$, $C(x) \in \mathcal{A}_k$, $D(y) \in \mathcal{A}_k$, $P_i(x, z_i), Q_i(y, z_i) \in \mathcal{A}_k$ for $1 \leq i \leq m$. This rule transforms $\mathcal{A}_k$ to $\mathcal{A}_{k+1}$ with $C(x) \in \mathcal{A}_{k+1}$, and adds a new ABox $\mathcal{A}'_{k+1}$ with $\sim C(x) \in \mathcal{A}'_{k+1}$.Since $\mathcal{I}$ is a model, we have either $\pi(x) \in C^{\mathcal{I}}$ or $\pi(x) \in \sim C^{\mathcal{I}}$. Assume that $\pi(x) \in C^{\mathcal{I}}$. In this case, we choose $\mathcal{A}_{k+1}$ including x with $\pi(x) \in C^{\mathcal{I}}$. This implies that Condition (4.7) is preserved in $\mathcal{A}_{k+1}$. Assume that $\pi(x) \in (\sim C)^{\mathcal{I}}$. In this case, we choose $\mathcal{A}'_{k+1}$ including $\sim C(x)$.This implies that Condition (4.7) is preserved in $\mathcal{A}'_{k+1}$.

- the $\rightarrow_{\text{chooseLK2}}$ rule. Analogously.

This completes the proof of preservation of Conditions 4.7-4.10 for each application of a completion rule. □

4.6.5 Complexity

Proposition 4 (Complexity)**.** *Let $\mathcal{O}_0 = \langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Algorithm 1 runs in doubly exponential time in the size of $\mathcal{O}_0$.*

Proof. According to the proof of Proposition 1, Algorithm 1 generates a collection $\mathbf{A}_k$ of ABoxes such that $|\mathbf{A}_k| \leq O(2^{2^\ell})$ and $|\mathcal{A}| \leq O(2^\ell)$ for all $\mathcal{A} \in \mathbf{A}_k$ where $\ell = ||\langle \mathcal{A}_0, \mathcal{T}, \mathcal{LK} \rangle||$. Since Algorithm 1 never removes anything from an intermediate ABox, the complexity is bounded by $O(2^{2^\ell})$. Therefore, it runs in deterministic doubly exponential time in the worst case (2EXPTIME). □

It is known that $\mathcal{ALC}$ with general concept axioms is EXPTIME-complete [56]. This result provides a lower bound of the reasoning problem in $\mathcal{ALC}$+$\mathcal{LK}$. The doubly exponential complexity of Algorithm 1 is caused by the interaction between the nondeterministic behavior, *i.e.* a new ABox is duplicated by non deterministic rules such as the $\rightarrow_{\sqcup}$ rule, and exponential generation of new individuals by the $\rightarrow_{\exists}$ rule. Moreover, we know from Lemma 4 that the completion rules related to the application of link keys are applied only to old individuals I_{old} whose cardinality is polynomial in the size of the ontology. This means that link keys are not responsible of the doubly exponential complexity resulting from Algorithm 1. An open question is whether EXPTIME is the tight lower bound of consistency checking in $\mathcal{ALC}$+$\mathcal{LK}$.

The following theorem is a consequence of all propositions established until now.

Theorem 5. *$\mathcal{ALC}$+$\mathcal{LK}$ consistency can be decided in doubly exponential time in the size of ontologies.*

4.7 Conclusion

In this chapter, we proposed a tableau-based algorithm for reasoning in the description logic $\mathcal{ALC}$+$\mathcal{LK}$, an extension of $\mathcal{ALC}$ with link keys and individual equalities. This algorithm uses anywhere blocking to guarantee its termination. We have provided the proofs of its soundness, completeness, and termination. Reasoning in $\mathcal{ALC}$+$\mathcal{LK}$ is more challenging than $\mathcal{ALC}$. It requires the introduction of new completion rules: the $\rightarrow_{\textbf{chooseLK1}}$, $\rightarrow_{\textbf{chooseLK2}}$, and $\rightarrow_{\textbf{LK}}$ to deal with link keys, and the $\rightarrow_{\approx}$ rule to handle equality. Also, the canonical interpretation used for proving the soundness of the standard $\mathcal{ALC}$ algorithm cannot be directly used for $\mathcal{ALC}$+$\mathcal{LK}$. We have introduced the unravelled interpretation to prove it.

This algorithm has 2EXPTIME complexity. However, reasoning in the description logic $\mathcal{ALC}$ with general TBoxes is EXPTIME-complete and there exist EXPTIME tableau algorithms for reasoning in $\mathcal{ALC}$ [49]. The completion rules added for dealing with link keys $\rightarrow_{\textbf{chooseLK1}}$, $\rightarrow_{\textbf{chooseLK2}}$, $\rightarrow_{\textbf{LK}}$, and $\rightarrow_{\approx}$ do not require any additional computation complexity than those of $\mathcal{ALC}$. In the light of this, we prove in the next chapter that the complexity of reasoning in $\mathcal{ALC}$+$\mathcal{LK}$ is EXPTIME. This goal is achieved by designing a sound and complete algorithm for reasoning in the description logic $\mathcal{ALC}$. To achieve the EXPTIME complexity, this algorithm uses an exponential structure for representing ontology models, this structure is inspired from compressed tableau algorithm for the description logic $\mathcal{SHOIQ}$ [16].

Chapter 5

A worst-case optimal EXPTIME algorithm for reasoning in the description logic $\mathcal{ALC}$ with link keys and individual equalities

5.1 Introduction

In the previous chapter, we have presented a tableau algorithm that decides the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. This algorithm has a doubly exponential complexity. The two sources leading to the doubly exponential complexity, as discussed in the previous chapter, are the generation of possibly an exponential number of new ABoxes upon the application of the non-deterministic rule and the expansion of the ABox due to the possible addition of an exponential number of new individuals upon the application of the existential rule. However, it is known that the consistency problem in $\mathcal{ALC}$ with general TBoxes is in EXPTIME-complete. In addition, the link key completion rules do not require any additional computation power than that of $\mathcal{ALC}$. In this chapter, we prove that consistency checking in $\mathcal{ALC}+\mathcal{LK}$ is EXPTIME, by providing a sound, complete, and EXPTIME algorithm for reasoning in $\mathcal{ALC}+\mathcal{LK}$. This algorithm is inspired from the compressed tableau algorithm for the description logic $\mathcal{SHOIQ}$ [16]. The compressed tableau algorithm eliminates the source of complexity resulting from the application of the non-deterministic rule. This is achieved by reducing the ABox generation into the generation of a smaller and compact structure called *star-type*. This allows to reduce the doubly exponential complexity of the standard tableau into a single exponential complexity. We call this algorithm a non-directed algorithm because it is not guided by the application of completion or saturation rules.

The remainder of this chapter is organized as follows. Section 5.2 provides the definitions required to understand the compressed tableau structure. Section 5.3 presents the non-directed compressed tableau algorithm for the description logic $\mathcal{ALC}+\mathcal{LK}$. In Section 5.4, we provide several examples and execute the algorithm on them. Section 5.5 gives and proves the theoretical properties of the algorithm. We provide our concluding remarks in Section 5.6.

5.2 A compressed tableau for the description logic $\mathcal{ALC}+\mathcal{LK}$

This section introduces an exponential structure, called compressed-tableau, for representing a model of an $\mathcal{ALC}+\mathcal{LK}$ ontology. Before digging into the definition of the compressed tableau and its components we first give the definition of ontology subconcepts. This definition is used by the algorithm and in the proofs.

Given an $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O}$, the set $\mathsf{sub}(\mathcal{O})$ includes all sub-concepts occurring in the axioms or assertions of ontology $\mathcal{O}$. If an axiom or assertion is viewed as a string then each element in $\mathsf{sub}(\mathcal{O})$ can be viewed as a substring. This implies that the cardinality of $\mathsf{sub}(\mathcal{O})$ is bounded by a polynomial function in the size of $\mathcal{O}$.

Definition 12 (Subconcepts of $\mathcal{O}$)**.** *Given an $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ with a set of individuals* $\mathbf{I}$*, the set of* subconcepts *of $\mathcal{O}$ is denoted by* $\mathsf{sub}(\mathcal{O})$ *and defined as the smallest set including all concept names such that:*

1. *if $C \sqsubseteq D \in \mathcal{T}$ then $\sim C \sqcup D \in \mathsf{sub}(\mathcal{O})$;*
2. *if $(\{\langle P_1, Q_1\rangle, \ldots, \langle P_n, Q_n\rangle\}$ $\mathsf{linkkey}$ $\langle C, D\rangle) \in \mathcal{LK}$ then $C, D \in \mathsf{sub}(\mathcal{O})$;*
3. *if $C(a) \in \mathcal{A}$ then $C \in \mathsf{sub}(\mathcal{O})$;*
4. *if $C \in \mathsf{sub}(\mathcal{O})$ then $\sim C \in \mathsf{sub}(\mathcal{O})$;*
5. *if $C_1 \sqcup C_2 \in \mathsf{sub}(\mathcal{O})$ or $C_1 \sqcap C_2 \in \mathsf{sub}(\mathcal{O})$ then $C_1, C_2 \in \mathsf{sub}(\mathcal{O})$;*
6. *if $\exists R.C \in \mathsf{sub}(\mathcal{O})$ or $\forall R.C \in \mathsf{sub}(\mathcal{O})$ then $C \in \mathsf{sub}(\mathcal{O})$;*
7. *if $a \in \mathbf{I}$ then $\{a\} \in \mathsf{sub}(\mathcal{O})$*

A compressed tableau consists of a set of *layers* each of which is composed of *star-types*. A star-type represents a set of semantically *similar* individuals and their neighbours. The intuition behind a star-type σ is to group *similar* individuals and their neighbours together in one structure. Similar individuals are either named equal individuals (ex. $a \approx b$) or anonymous individuals satisfying the same set of concepts. Moreover, a star-type needs to store connections to other star-types which represent others sets of individuals. A star-type is composed of a set of *triples*. A triple has three components: a *head*, a *tie*, and a *tail*. Both the head and the tail are pairs of sets of individuals and concepts.The tie is a set of roles. A triple represent the relation ship between a pair of sets of individuals.

Definition 13 (**Triple**)**.** *Let $\mathcal{O}$ be an $\mathcal{ALC}+\mathcal{LK}$ ontology. Let* $\mathbf{I}$ *and* $\mathbf{R}$ *be the sets of individuals and roles in $\mathcal{O}$. An triple $\rho = \langle\langle X, U\rangle, P, \langle Y, V\rangle\rangle$ where $\langle X, U\rangle$ and $\langle Y, V\rangle$ are pairs such that $X, Y \subseteq \mathbf{I}$, $U, V \subseteq \mathsf{sub}(\mathcal{O})$ and $P \subseteq \mathbf{R}$. We will use* $\mathbf{TR}$ *to denote the set of all triples. Given a triple $\rho = \langle\langle X, U\rangle, P, \langle Y, V\rangle\rangle$, the sets $\langle X, U\rangle$, P and $\langle Y, V\rangle$ are referred to as the* head*, the* tie *and the* tail *of ρ, and they will be denoted by* $\mathsf{head}(\rho)$*,* $\mathsf{tie}(\rho)$ *and* $\mathsf{tail}(\rho)$*. We also denote* $\mathsf{head}_\mathsf{I}(\rho) = X$*,* $\mathsf{head}_\mathsf{C}(\rho) = U$*,* $\mathsf{tail}_\mathsf{I}(\rho) = Y$ *and* $\mathsf{tail}_\mathsf{C}(\rho) = V$*. Finally, a* cyclic *triple is a triple ρ such that* $\mathsf{head}(\rho) = \mathsf{tail}(\rho)$.

The following example gives an ontology $\mathcal{O}$ and a triple τ built from it.

Example 8 (Triple)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}+\mathcal{LK}$ ontology where:*

1. $\mathcal{A} = \{A(a), B(b), R(a,b), S(a,b)\}$,
2. $\mathcal{T} = \emptyset$,
3. $\mathcal{LK} = \emptyset$.

Figure 5.1 depicts the triple $\tau = \Big\langle \langle \{a\}, \{A\} \rangle, \{R, S\}, \langle \{b\}, \{B\} \rangle \Big\rangle$ *built from* $\mathcal{O}$*, where* $\mathsf{head_I}(\tau) = \{a\}$, $\mathsf{head_C}(\tau) = \{A\}$, $\mathsf{tie}(\tau) = \{R, S\}$, $\mathsf{tail_I}(\tau) = \{b\}$ *and* $\mathsf{tail_C}(\tau) = \emptyset$.

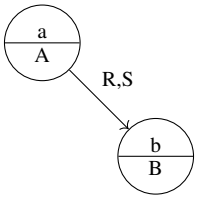


Figure 5.1 – Graphical representation of the triple τ.

A triple τ is called *dummy* if $\mathsf{tie}(\tau) = \emptyset$, $\mathsf{tail}(\tau) = \emptyset$, and non dummy otherwise.
A star-type is a set of triples having the same head.

Definition 14 (Star-type). *Let* $\mathcal{O}$ *be an* $\mathcal{ALC}+\mathcal{LK}$ *ontology. A subset* $\sigma \subseteq \mathbf{TR}$ *is said to be a* star-type *if for all* $\rho, \rho' \in \sigma$, $\mathsf{head}(\rho) = \mathsf{head}(\rho')$. *We will denote the set of all star-types by* $\Sigma(\mathcal{O})$. *Given* $\sigma \in \Sigma(\mathcal{O})$*, the* core *of* σ*, denoted by* $\mathsf{core}(\sigma)$*, is the head of any triple in* σ. *If* $\mathsf{core}(\sigma) = \langle X, U \rangle$*, we will denote* $\mathsf{core_I}(\sigma) = X$ *and* $\mathsf{core_C}(\sigma) = U$. *A star-type* $\sigma \in \Sigma(\mathcal{O})$ *is said to be* nominal *if* $\mathsf{core_I}(\sigma) \neq \emptyset$.

Example 9. *Figure 5.2 depicts a star-type* σ *made up of 3 triples:*

1. $\rho_1 = \langle \langle \{a\}, \{\exists S.A, \forall S.D\} \rangle, \{R\}, \langle \{b\}, \{C, E\} \rangle \rangle$,
2. $\rho_2 = \langle \langle \{a\}, \{\exists S.A, \forall S.D\} \rangle, \{R\}, \langle \{c\}, \{C, F\} \rangle \rangle$, *and*
3. $\rho_3 = \langle \langle \{a\}, \{\exists S.A, \forall S.D\} \rangle, \{S\}, \langle \{t\}, \{D, A\} \rangle \rangle$.

$\mathsf{core}(\sigma) = \langle \{a\}, \{\exists S.A, \forall S.D\} \rangle$ *and* σ *is nominal because* $\mathsf{core_I}(\sigma) = \{a\} \neq \emptyset$.

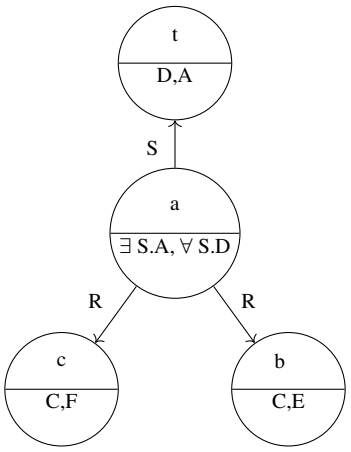
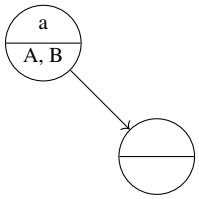


Figure 5.2 – A star-type made up of 3 triples.

Notice that we would need a very specific star-type that has just one dummy triple. Such a star-type is also called *dummy* (see Figure 5.3a).



(a) A dummy star-type made-up of one dummy triple.

(b) A simplified representation of the dummy star-type in Figure 5.3a.

For the sake of simplicity we will represent a dummy star-type by a single node as shown in Figure 5.3b.

To ensure termination of the algorithm in the presence of GCIs we need to define blocking (Definition 16). To define blocking, we need to have an order between star-types. We define the order between star-types in a similar way that defines the order between individuals since star-types represent a set of equal individuals and their neighbours. For that, we introduce the notion of a *layer* (Definition 15). Layers will provide the order between star-types in a compressed tableau.

Definition 15 (**Star-type Layer**). *Let $\mathcal{O}$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^{n}$ be a finite sequence of sets of star-types, i.e. $\Lambda_k \subseteq \Sigma(\mathcal{O})$ for $0 \leq k \leq n$. Then Λ_k will be referred to as a* star-type layer *or simply a* layer.

Definition 16 (**Blocked star-type**). *Let $\mathcal{O}$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^{n}$ be a sequence of layers. Let $\sigma' \in \Lambda_i$ for some $1 < i \leq n$. We say that σ' is* blocked *by a star-type $\sigma \in \Lambda_k$ if $0 < k < i$ and $\mathsf{core}_\mathsf{C}(\sigma') \subseteq \mathsf{core}_\mathsf{C}(\sigma)$.*

The definition of a *valid* star-type is given in Definition 17. Conditions 1–5 capture the semantics of $\mathcal{ALC}$ constructors. Condition 6 is needed because $\mathcal{ALC}$+$\mathcal{LK}$ allows to express equality assertions. Conditions 7 and 8 ensure that valid star-types do not contain any contradiction. Condition 9 is required to satisfy the role assertions present in $\mathcal{A}$. Notice that no condition translates the semantics of link keys. This is because a link key does not apply to one, but several star-types.

Definition 17 (**Valid star-type**). *Let $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let σ be a star-type over $\mathcal{O}$. The star-type σ is* valid *if the following properties hold:*

1. *If $E \sqsubseteq F \in \mathcal{T}$ then $\mathsf{nnf}(\neg E \sqcup F) \in \mathsf{core}_\mathsf{C}(\sigma)$, and $\mathsf{nnf}(\neg E \sqcup F) \in \mathsf{tail}_\mathsf{C}(\rho)$ for all $\rho \in \sigma$.*
2. *$C_1 \sqcap C_2 \in \mathsf{core}_\mathsf{C}(\sigma)$ implies $\{C_1, C_2\} \subseteq \mathsf{core}_\mathsf{C}(\sigma)$, and $C_1 \sqcap C_2 \in \mathsf{tail}_\mathsf{C}(\rho)$ implies $\{C_1, C_2\} \subseteq \mathsf{tail}_\mathsf{C}(\rho)$ for all $\rho \in \sigma$.*
3. *$C_1 \sqcup C_2 \in \mathsf{core}_\mathsf{C}(\sigma)$ implies $\{C_1, C_2\} \cap \mathsf{core}_\mathsf{C}(\sigma) \neq \emptyset$, and $C_1 \sqcup C_2 \in \mathsf{tail}_\mathsf{C}(\rho)$ implies $\{C_1, C_2\} \cap \mathsf{tail}_\mathsf{C}(\rho) \neq \emptyset$ for all $\rho \in \sigma$.*
4. *If $\exists R.D \in \mathsf{core}_\mathsf{C}(\sigma)$ then there is a triple $\rho \in \sigma$ such that $\{D, C_{\exists R.D}\} \subseteq \mathsf{tail}_\mathsf{C}(\rho)$ and $R \in \mathsf{tie}(\rho)$.*

5. *If* $\forall R.C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ *then for every triple* $\rho \in \sigma$*, if* $R \in \mathsf{tie}(\rho)$ *then* $C \in \mathsf{tail}_{\mathsf{C}}(\rho)$.

6. *If* $a \approx b \in \mathcal{A}$ *then* $\{a, b\} \cap \mathsf{core}_{\mathsf{I}}(\sigma) \neq \emptyset$ *implies* $\{a, b\} \subseteq \mathsf{core}_{\mathsf{I}}(\sigma)$*, and* $\{a, b\} \cap \mathsf{tail}_{\mathsf{I}}(\rho) \neq \emptyset$ *implies* $\{a, b\} \subseteq \mathsf{tail}_{\mathsf{I}}(\rho)$ *for all* $\rho \in \sigma$

7. *If* $a \not\approx b \in \mathcal{A}$ *then* $\{a, b\} \not\subseteq \mathsf{core}_{\mathsf{I}}(\sigma)$*, and* $\{a, b\} \not\subseteq \mathsf{tail}_{\mathsf{I}}(\rho)$ *for all* $\rho \in \sigma$.

8. *For each concept name* A*, it holds that* $\{A, \sim A\} \not\subseteq \mathsf{core}_{\mathsf{C}}(\sigma)$*, and* $\{A, \sim A\} \not\subseteq \mathsf{tail}_{\mathsf{C}}(\rho)$ *for all* $\rho \in \sigma$.

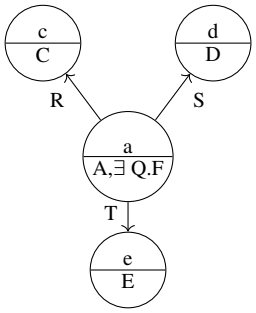
9. *For each* $R(a, b) \in \mathcal{A}$ *if* $a \in \mathsf{core}_{\mathsf{C}}(\sigma)$ *then there is exactly one triple* $\rho \in \sigma$ *such that* $b \in \mathsf{tail}_{\mathsf{I}}(\rho)$ *and* $R \in \mathsf{tie}(\rho)$.

10. *The number of triples of* σ *is bounded by* $|\mathsf{core}(\sigma)| + |\mathcal{A}|$.

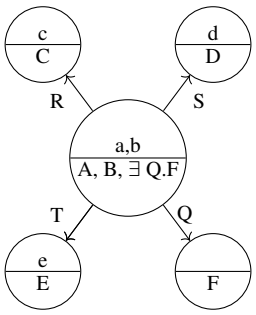
Notice that Property 10 restricts the number of triples of a star-type such that it should be polynomial in the size of $\mathsf{sub}(\mathcal{O})$ and $\mathcal{O}$. The star-types created by the algorithm must satisfy this property to ensure that the EXPTIME complexity of the algorithm. To define a compressed tableau for an $\mathcal{ALC}$ ontology, we need to define first the transitive closure of an individual in $\mathbf{I}$. For each individual $s \in \mathbf{I}$, we use s^+ to denote the transitive closure of s with respect to the relation $\approx$ (appearing in assertions), *i.e.* s^+ is the smallest set such that $s \in s^+$, and if $c \approx b \in \mathcal{A}$ or $b \approx c \in \mathcal{A}$ with some $c \in s^+$ then $b \in s^+$.

Example 10. *Consider the input ontology* $\mathcal{O}$ *where:*

- $\mathcal{A} = \{A(a), B(b), a{\approx}b, C(c), R(a, c), D(d), S(b, d), E(e), T(a, e)\}$,
- $\mathcal{T} = \{A \sqsubseteq B\}$,
- $\mathcal{LK} = \emptyset$.



(a) The star-type σ is an invalid star-type built from $\mathcal{O}$.



(b) A valid star-type σ satisfying the GCI $A \sqsubseteq B \in \mathcal{T}$, the concept $\exists Q.F \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and the equality assertion $a \approx b \in \mathcal{A}$.

Figure 5.4 – An invalid and a valid star-type built from $\mathcal{O}$.

In the standard tableau each individual has a unique label. In opposite, in the compressed tableau a star-type, the same set of individuals might occur in different star-types with different concepts of different neighbours. As a result, the relationship between a pair of individuals in

the standard tableau is one to one, however, in the compressed tableau an individual in one star-type can be related to a set of other star-type containing the same individual. The relationship between star-types is hence a one to many relationship, and has to be stored using a specific function, which we call *matching* function.

Definition 18 (Matching function over a sequence of layers)**.** *Let $\mathcal{O}$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^{n}$ be a finite sequence of layers. A* matching function *over $\mathbf{\Lambda}$ is a function Ω that associates a non-empty set of star-types $\Omega(\sigma, \tau) \subseteq \mathbf{\Lambda}$ to each pair (σ, τ) of a star-type $\sigma \in \mathbf{\Lambda}$ and a non dummy triple $\tau \in \sigma$, and satisfies:*

1. *For every $\sigma \in \mathbf{\Lambda}$ and $\tau \in \sigma$, if $\sigma' \in \Omega(\sigma, \tau)$ then $\mathsf{core}(\sigma') = \mathsf{tail}(\tau)$;*
2. *For every $\sigma \in \mathbf{\Lambda}$ and $\tau \in \sigma$,*
   (a) if $\sigma \in \Lambda_0$ then $\Omega(\sigma, \tau) \subseteq \Lambda_0 \cup \Lambda_1$;
   (b) if $\sigma \in \Lambda_k$ and $0 < k < n$ then $\Omega(\sigma, \tau) \subseteq \Lambda_{k+1}$;

Definition 19 (Neighbour)**.** *A star-type $\sigma \in \Lambda_i$ is called a* ρ-successor *(or successor) of ω via a triple $\rho \in \omega$ if $\sigma \in \Omega(\omega, \rho)$, and ω is called a* ρ-predecessor *(or predecessor) of σ. In this case, if $R \in \mathsf{tie}(\rho)$ then σ is called an R-successor of ω, and ω an R-predecessor of σ. A star-type is called a* neighbor *of another one if one is a successor of the other.*

We are now ready to give the structure of the compressed tableau for the description logic $\mathcal{ALC}$.

Definition 20 (Compressed tableau for an $\mathcal{ALC}$ ontology)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T} \rangle$ be an $\mathcal{ALC}$ ontology $\mathcal{O}$. Let $\mathbf{I}$ be the set of individuals in $\mathcal{O}$. Let $\mathbf{\Lambda}$ be a set of layers and Ω a matching function over $\mathbf{\Lambda}$. The pair $\langle \mathbf{\Lambda}, \Omega \rangle$ is said to be a* compressed tableau *of $\langle \mathcal{A}, \mathcal{T} \rangle$ if the following conditions are satisfied:*

1. *Every star-type $\sigma \in \Lambda_i, 0 \leq i \leq n$, σ is valid.*
2. *For every $\sigma \in \bigcup_{i=0}^{n} \Lambda_i$, σ is nominal iff $\sigma \in \Lambda_0$. Moreover, each star-type in the last layer Λ_n either is blocked, or dummy.*
3. *For every $a \in \mathbf{I}$, there exists a unique valid nominal star-type $\sigma \in \Lambda_0$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. Furthermore, $C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ for each $C(b) \in \mathcal{A}$ with $b \in a^{+}$.*
4. *For each $R(a, b) \in \mathcal{A}$ and $\sigma, \sigma' \in \Lambda_0$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma')$, there is a triple $\rho \in \sigma$ such that $R \in \mathsf{tie}(\rho)$ and $\sigma' \in \Omega(\sigma, \rho)$.*

After defining the compressed tableau for the description logic $\mathcal{ALC}$, and before defining the compressed tableau for the description logic $\mathcal{ALC}$+$\mathcal{LK}$, we must give the conditions that guarantees the satisfaction of link keys.

Definition 21 (Star-types satisfying a link key condition)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC} + \mathcal{LK}$ ontology and $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ be a compressed-tableau of $\mathcal{O}$. Let $\lambda = (\{\langle P_i, Q_i \rangle\}_{i=1}^{n}$ linkkey $\langle C, D \rangle)$ be a link key in $\mathcal{LK}$.*

1. *We say that two star-types* $\sigma, \sigma' \in \Lambda_0$ weakly satisfy the condition *of* λ *if there exist triples* $\rho_1, \ldots, \rho_n \in \sigma$, $\rho'_1, \ldots, \rho'_n \in \sigma'$ *such that* $P_i \in \mathsf{tie}(\rho_i)$, $Q_i \in \mathsf{tie}(\rho'_i)$ *and* $\emptyset \neq \Omega(\sigma, \rho_i) \cap \Omega(\sigma', \rho'_i) \subseteq \Lambda_0$

2. *We say that two star-types* $\sigma, \sigma' \in \Lambda_0$ satisfy the condition *of* λ *if they weakly satisfy* λ*, and* $C \in \mathsf{core}_\mathsf{C}(\sigma)$, $D \in \mathsf{core}_\mathsf{C}(\sigma')$. *Furthermore, we say that* λ *is satisfied over* Λ_0 *if there does not exist any pair of star-types* $\sigma, \sigma' \in \Lambda_0$ *such that* σ, σ' *satisfy* λ *and* $\sigma \neq \sigma'$.

Example 11. *Let* $\mathcal{O}$ *be an* $\mathcal{ALC}$+$\mathcal{LK}$ *ontology where* $\lambda = (\{\langle P, Q\rangle\}$ linkkey $\langle A, B\rangle)$ *is a link key in* $\mathcal{O}$ *and let* $\sigma_a, \sigma_b, \sigma_d$ *and* σ_e *be a set of star-types built from* $\mathcal{O}$. *The star-types* σ_a *and* σ_b *are matched to the same set of star-types* σ_d *and* σ_e *respectively through their pair of triples of properties P and Q (Figure 5.5). These star-types, as a result, satisfy the condition of* $(\{\langle P, Q\rangle\}$ linkkey $\langle A, B\rangle)$. *In the figures that follows, we use the rectangles to represent the equalities between the tail of triples of some star-types and the core of others. This equality besides the other conditions affirms the matching between the involved star-types.*

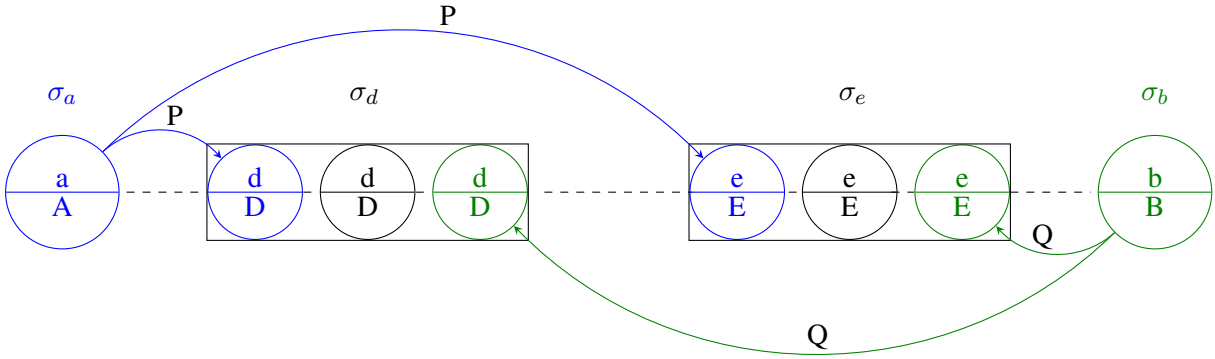


Figure 5.5 – The star-types σ_a and σ_b **satisfy the condition** of λ through the star-types σ_d and σ_e.

A compressed tableau for an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$ is a compressed tableau for $\mathcal{ALC}$ such that every link key in $\mathcal{LK}$ is satisfied. We also add a condition that is necessary to ensure completeness of the algorithm (Item 3 of Definition 22).

Definition 22 (Compressed tableau for the description logic $\mathcal{ALC}$+$\mathcal{LK}$)**.** *Let* $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK}\rangle$ *be an* $\mathcal{ALC}$ *ontology. A* compressed tableau *for* $\mathcal{O}$ *is a pair* $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega\rangle$ *with* $\mathbf{\Lambda} = \langle \Lambda_0, \cdots, \Lambda_n\rangle$ *such that:*

1. $\mathcal{CT}$ *is a compressed tableau of* $\langle \mathcal{A}, \mathcal{T}\rangle$*;*

2. *Each link key* $\lambda = (\{\langle P_i, Q_i\rangle\}_{i=1}^n$ linkkey $\langle C, D\rangle) \in \mathcal{LK}$ *is satisfied over* Λ_0*;*

3. *If there are star-types* σ *and* σ' *that weakly satisfy the condition of a link key* $\lambda = (\{\langle P_i, Q_i\rangle\}_{i=1}^n$ linkkey $\langle C, D\rangle) \in \mathcal{LK}$*, then* $\{C, {\sim}C\} \cap \mathsf{core}_\mathsf{C}(\sigma) \neq \emptyset$ *and* $\{D, {\sim}D\} \cap \mathsf{core}_\mathsf{C}(\sigma') \neq \emptyset$.

5.3 A non-directed algorithm for the description logic $\mathcal{ALC}+\mathcal{LK}$

This section introduces a non-directed algorithm for checking the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. The algorithm operates on a compressed tableau of exponential size. There are two main features in our algorithm which allow us to obtain an exponential complexity. The first one consists in compressing similar individuals, i.e. those individuals which satisfy the same set of concepts and have the same connections to other individuals, into a star-type. The second one is related to handling non-determinism caused by disjunctions. When dealing with a disjunction of the form $C_1 \sqcup C_2$, instead of duplicating the whole current structure which aims to represent a model, our algorithm generates just a new star-type to be chosen if needed. This behaviour allows to reduce global non-determinism to local non-determinism. If the input ontology is consistent, the algorithm builds a compressed tableau that represents a model for the input ontology.

Figures 5.6 and 5.7 show the difference between handling non-determinism in the standard tableau algorithms and the compressed tableau ones. Given an individual c of label $\exists R.(C_1 \sqcup C_2)$ and a successor g of label $C_1 \sqcup C_2$. A standard tableau algorithm, upon the application of $\rightarrow_{\sqcup}$-rule to the individual g, creates two duplicates of the whole completion tree under construction (Figure 5.6). The label of g in the completion tree in the left-hand completion tree is $C_1 \sqcup C_2, C_1$ while in the right-hand side $C_1 \sqcup C_2, C_2$.

In Figure 5.6, for the sake of simplifying the figure, we omit the labels of the individuals not concerned with the non-determinism. In Figure 5.7, we do not represent the individuals $a, \ldots, g$ since they are new individuals thus they correspond to non-nominal star-types.

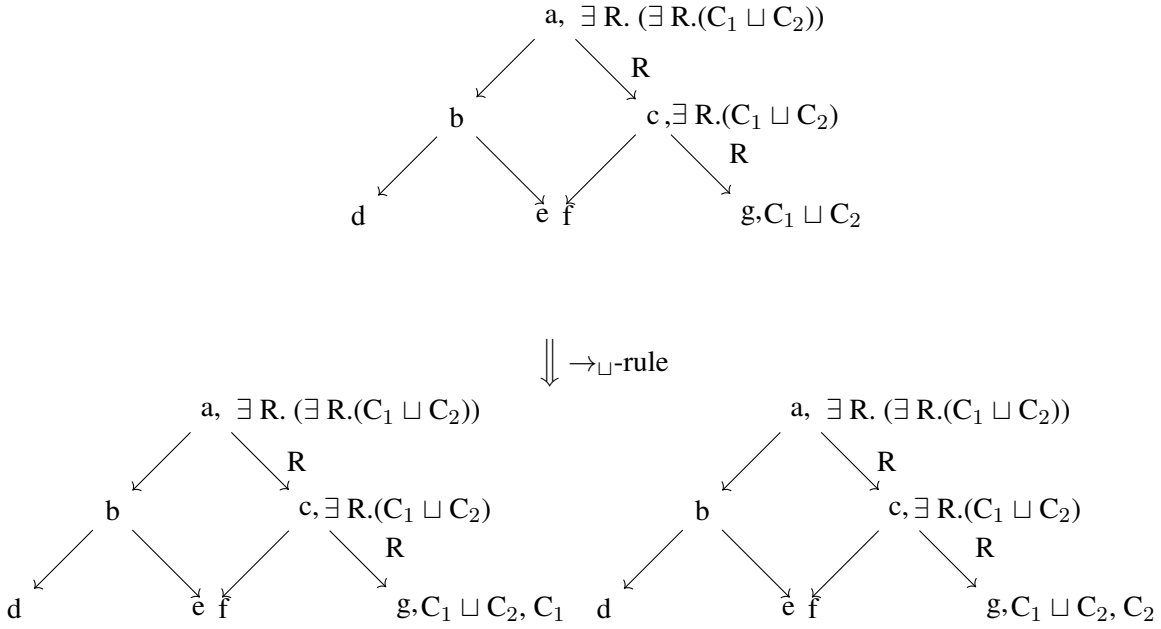


Figure 5.6 – A pair of ABoxes generated upon the application of non-deterministic rule in the standard tableau algorithms.

While a compressed tableau algorithm duplicates only the part of the tableau concerned with the non-determinism (Figure 5.7). In Figure 5.7, the star-type σ_a is matched through its triple of tie R to σ_{c1} and σ_{c2}.

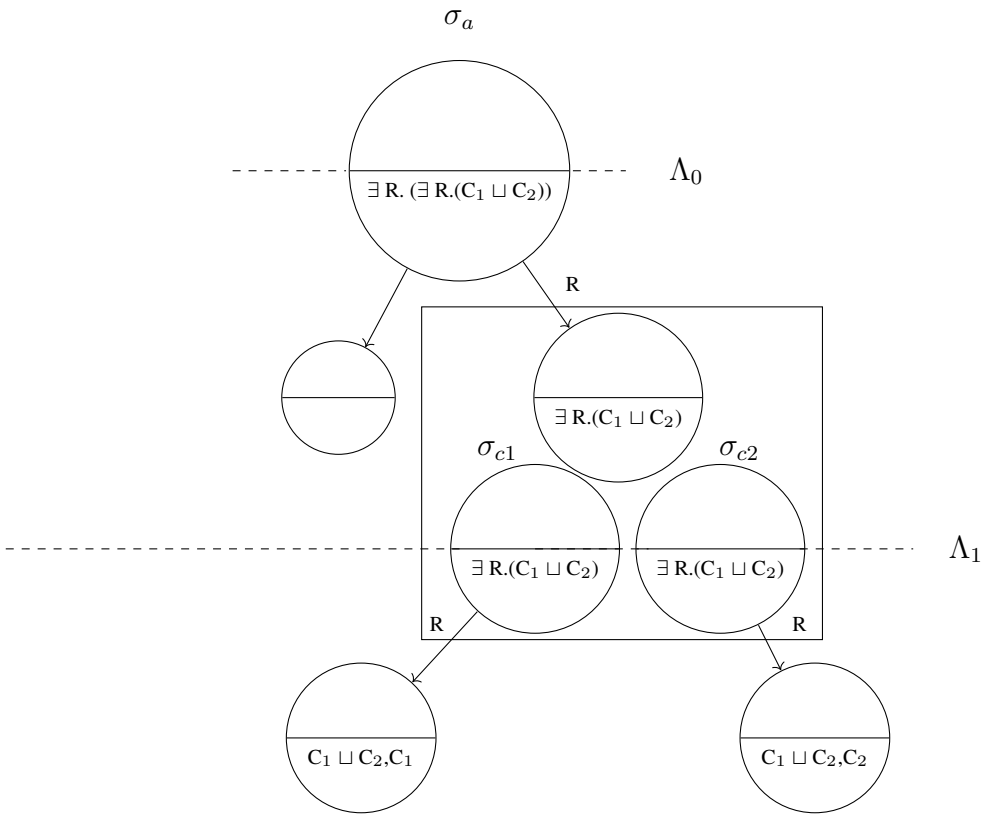


Figure 5.7 – A pair of star-types generated upon the application of non-deterministic rule in the compressed tableau algorithm.

We now present an algorithm for checking consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. Given an $\mathcal{ALC}+\mathcal{LK}$ input ontology $\mathcal{O}$, Algorithm 3 attempts to build a compressed tableau for $\mathcal{O}$.

First, it builds the sets $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$, the sets of all valid nominal and non nominal star-types from $\mathcal{O}$, respectively. Then it chooses a set of nominal valid star-types as a candidate for Λ_0. Then it checks that for each individual there is a unique star-type in Λ_0 which contains that individual, it connects star-types in Λ_0 by Ω, and it checks the satisfaction of each link key in $\mathcal{LK}$.

The algorithm continues to build a layer $\Lambda_i, 1 \leq i \leq n$ by adding all star-types from $\Sigma_C(\mathcal{O})$ that are matched to Λ_{i-1} via Ω. Finally, Algorithm 3 calls Algorithm 2 to remove all the non blocked star-types which have no connection via one of its triples.

Algorithm 2 ensures that for each individual there is still a unique star-type in Λ_0 which contains that individual. If a valid star-type σ has a triple ρ but $\Omega(\sigma, \rho)$ is empty, then σ cannot belong to a compressed tableau, and it should be removed by Algorithm 2. This process can be performed in cascade since the removal of such a star-type σ may make $\Omega(\sigma', \rho')$ empty with $\sigma \in \Omega(\sigma', \rho')$.

Algorithm 2: Algorithm for pruning compressed tableau

Input : $\mathcal{CT} = \langle \Lambda_0, \cdots, \Lambda_n \rangle$ with Ω over $\mathcal{CT}$ and a set of individuals $\mathbf{I}$
Output: true/false

while *there is a non-blocked star-type* $\sigma \in \Lambda_i$ *and* $\rho \in \sigma$ *with* $1 \leq i \leq n$ *such that* $\Omega(\sigma, \rho) = \emptyset$ **do**
  if *there is a* $\sigma' \in \Lambda_{i-1}$ *and* $\rho' \in \sigma'$ *with* $i > 0$ *such that* $\sigma \in \Omega(\sigma', \rho')$ **then**
    Remove σ from $\Omega(\sigma', \rho')$ and Λ_i
  end
end
if *for each* $a \in \mathbf{I}$ *there is some* $\sigma \in \Lambda_0$ *such that* $a \in \mathsf{core_I}(\sigma)$ **then**
  return true;
end
return false;

Algorithm 3: Compressed algorithm for $\mathcal{ALC}$+$\mathcal{LK}$ ontology consistency checking

Input : $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals $\mathbf{I}$
Output: YES/NO

Build a set $\Sigma_N(\mathcal{O})$ of all valid nominal star-types;
Build a set $\Sigma_C(\mathcal{O})$ of all valid non nominal star-types;
foreach $\Lambda_0 \subseteq \Sigma_N(\mathcal{O})$ *such that for each* $a \in \mathbf{I}$ *there is exactly one star-type* $\sigma \in \Lambda_0$ *with* $a \in \mathsf{core}_I(\sigma)$ **do**
  if *for each* $\sigma \in \Lambda_0$ *and* $\rho \in \sigma$ *with* $a \in \mathsf{tail_I}(\rho)$*, there is some* $\sigma' \in \Lambda_0$ *such that* $\mathsf{core}(\sigma') = \mathsf{tail}(\rho)$ **then**
    Add σ' to $\Omega(\sigma, \rho)$;
    if *each link key* λ *in* $\mathcal{LK}$ *is satisfied over* Λ_0*, and for each* $\sigma \in \Lambda_0$ *it holds* $\{C, \sim C\} \cap \mathsf{core_C}(\sigma) \neq \emptyset$ *if* C *occurs in* λ **then**
      $\mathcal{CT} \leftarrow \emptyset$, add Λ_0 to $\mathcal{CT}$, and $i \leftarrow 0$;
      while Λ_i *contains a non blocked star-type which has at least one triple* **do**
        foreach *non blocked star-type* $\sigma \in \mathbf{\Lambda}'$ *and triple* $\rho \in \sigma$ **do**
          Add to $\Omega(\sigma, \rho)$ and Λ_{i+1} all star-type $\sigma' \in \Sigma_C(\mathcal{O})$ such that $\mathsf{core}(\sigma') = \mathsf{tail}(\rho)$;
        end
        Add Λ_{i+1} to $\mathcal{CT}$, and $i \leftarrow i + 1$;
      end
      if pruning$(\mathcal{CT}, \mathbf{I})$ **then**
        return YES;
      end
    end
  end
end
return NO;

5.4 Examples

This section provides an example of an $\mathcal{ALC}+\mathcal{LK}$ ontology shows the execution of Algorithm 3 on it.

Example 12. *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}+\mathcal{LK}$ ontology where:*

1. $\mathcal{A} = \{C(a), D(b), B(g), M(b,g), P(a,v), Q(b,v), (\exists W.(\exists P.\top \sqcap \exists S.\exists P.\top))(v)\}$
2. $\mathcal{T} = \{B \sqsubseteq (\exists P.\top \sqcap \exists S.\exists P.\top) \sqcup \exists R.(E \sqcap D), D \sqsubseteq \neg E\}$
3. $\mathcal{LK} = (\{\langle P, Q\rangle\} \text{ linkkey } \langle C, D\rangle)$

We now show the execution of Algorithm 3 on $\mathcal{O}$:

1. *First, the algorithm builds the set of all valid star-types from the subconcepts of $\mathcal{O}$ ($\mathsf{sub}(\mathcal{O})$). $\mathsf{Sub}(\mathcal{O})$ includes, according to Definition 12, the following concepts* $\{\{a\}, \{g\}, \{b\}, \{v\}, C, D, B, E, \neg E, (\exists P.\top \sqcap \exists S.\exists P.\top) \sqcup \exists R.(E \sqcap D), E \sqcap D, \exists R.(E \sqcap D), \exists P.\top \sqcap \exists S.\exists P.\top, \exists P.\top, \exists S.\exists P.\top, \exists W.(\exists P.\top \sqcap \exists S.\exists P.\top), C_{\exists R.(E \sqcap D)}, C_{\exists P.\top}, C_{\exists S.\exists P.\top}, C_{\exists W.(\exists P.\top \sqcap \exists S.\exists P.\top)}\}$.

2. *Second, it builds the set of all triples* $\mathbf{TR}$ *by first taking any pair of subsets $L_1, L_2 \subseteq \mathsf{sub}(\mathcal{O})$, where each label L_1 and L_2 contains a set of individuals and a set of concepts.*

   We refer by L_x and L'_x the alternative core labels of the star-type containing the set of individuals x, by L_{xi} and L'_{xi} the alternative labels of the i^{th} successor of the star-type of individuals x.

   $L_a = \langle \{a\}, \{C\} \rangle$, $L_b = \langle \{b\}, \{D, \neg E\} \rangle$,
   $L_g = \langle \{g\}, \{B, (\exists P.\top \sqcap \exists S.\exists P.\top) \sqcup \exists R.(E \sqcap D), \exists P.\top \sqcap \exists S.\exists P.\top, \exists P.\top, \exists S.\exists P.\top\} \rangle$,
   $L'_g = \langle \{g\}, \{B, (\exists P.\top \sqcap \exists S.\exists P.\top) \sqcup \exists R.(E \sqcap D), \exists R.(E \sqcap D)\} \rangle$,
   $L_v = \langle \{v\}, \{\exists W.(\exists P.\top \sqcap \exists S.\exists P.\top)\} \rangle$, $L_{ab} = \langle \{a, b\}, \{C, D, \neg E\} \rangle$,
   $L_{g1} = \langle \emptyset, \{C_{\exists P.\top}\} \rangle$, $L'_{g1} = \langle \emptyset, \{E, D, \neg E\} \rangle$,
   $L_{g2} = \langle \emptyset, \{\exists P.\top, C_{\exists S.\exists P.\top}\} \rangle$,
   $L_{v1} = \langle \emptyset, \{\exists P.\top \sqcap \exists S.\exists P.\top, C_{\exists W.(\exists P.\top \sqcap \exists S.\exists P.\top)}\} \rangle$, ...
   Then it takes a subset $r \subseteq \mathbf{R}$ to form a triple ρ with $\mathsf{head}(\rho) = L_1$, $\mathsf{tie}(\rho) = r$, and $\mathsf{tail}(\rho) = L_2$, and adds ρ to $\mathbf{TR}$*. For example, it can build:*

   - $\rho_1 = \langle L_a, \{P\}, L_v \rangle$
   - $\rho_2 = \langle L_b, \{Q\}, L_v \rangle$, $\rho_3 = \langle L_b, \{M\}, L_g \rangle$
   - $\rho_4 = \langle L_{ab}, \{P, Q\}, L_v \rangle$, $\rho_5 = \langle L_{ab}, \{M\}, L_g \rangle$
   - $\rho_6 = \langle L_g, \{P\}, L_{g1} \rangle$, $\rho_7 = \langle L_g, \{S\}, L_{g2} \rangle$
   - $\rho_8 = \langle L'_g, \{R\}, L'_{g1} \rangle$
   - $\rho_9 = \langle L_v, \{W\}, L_{v1} \rangle$
   - $\rho_{10} = \langle L_{g2}, \{P\}, L_{g1} \rangle$,
   - $\rho_{11} = \langle L_{v1}, \{S\}, L_{g2} \rangle$, $\rho_{12} = \langle L_{v1}, \{P\}, L_{g1} \rangle$,
   - $\rho_{13} = \langle L_{g1}, \emptyset, \emptyset \rangle$,
   - $\rho_{14} = \langle L_{g2}, \emptyset, \emptyset \rangle$

3. *From these triples, the algorithm can build the following valid star-types:* $\sigma_1 = \{\rho_1\}$, $\sigma_2 = \{\rho_2, \rho_3\}$, $\sigma_3 = \{\rho_6, \rho_7\}$, $\sigma_4 = \{\rho_9\}$, $\sigma_5 = \{\rho_{11}, \rho_{12}\}$, $\sigma_6 = \{\rho_{10}\}$, $\sigma_7 = \{\rho_{13}\}$, $\sigma_8 = \{\rho_{14}\}$, $\sigma_9 = \{\rho_4, \rho_5\}$. *Notice that the disjunction $(\exists P.\top \sqcap \exists S.\exists P.\top) \sqcup \exists R.(E \sqcap D)$ leads to building a non-valid star-type $\sigma = \{\rho_8\}$ whose tail contains the clash $\{E, \neg E\}$.*

4. *Then, the algorithm chooses* $\Lambda_0 = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$ *as shown in Figure 5.8. Then it connects the star-types according to definition of the matching function* Ω *(Definition 28). For that we have* $\Omega(\sigma_2, \rho_3) = \sigma_3$ *as* $\mathsf{core}(\sigma_3) = \mathsf{tail}(\rho_3)$*, this match is presented by a double edged node. In addition,* $\Omega(\sigma_1, \rho_1) \cap \Omega(\sigma_2, \rho_2) = \{\sigma_4\}$ *as* $\mathsf{tail}(\rho_1) = \mathsf{tail}(\rho_2) = \mathsf{core}(\sigma_4)$*, this match is presented by a triple edged node.*

5. *But this* Λ_0 *does not satisfy the link key* $(\{\langle P, Q\rangle\}$ linkkey $\langle C, D\rangle)$ *in* $\mathcal{LK}$ *because* σ_1 *and* $\sigma_2 \in \Lambda_0$ *they satisfy the link key condition as* $C \in \mathsf{core}_\mathsf{C}(\sigma_1)$ *and* $D \in \mathsf{core}_\mathsf{C}(\sigma_2)$*,* $P \in \mathsf{tie}(\rho_1)$*,* $Q \in \mathsf{tie}(\rho_2)$*, and* $\Omega(\sigma_1, \rho_1) \cap \Omega(\sigma_2, \rho_2) = \{\sigma_4\} \subseteq \Lambda_0$ *however,* $\sigma_1 \neq \sigma_2$*. Satisfying the link key* $(\{\langle P, Q\rangle\}$ linkkey $\langle C, D\rangle)$ *in* Λ_0 *requires to merge* σ_1 *into* σ_2 *due to the equality assertion* $a \approx b$ *implied by the link key. For that such as choice of* Λ_0 *is discarded.*

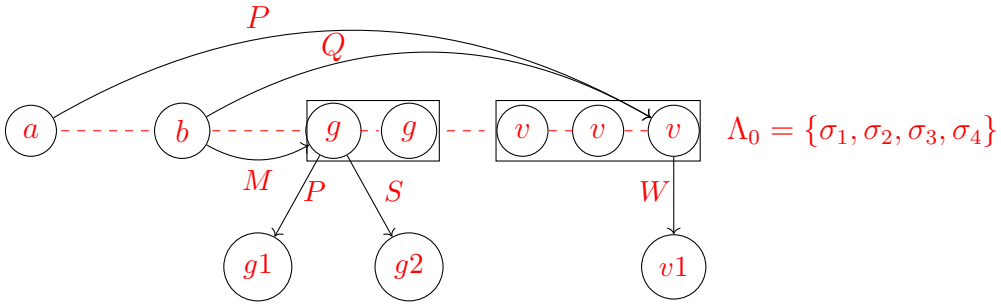


Figure 5.8 – Valid star-types that do not satisfy $(\{\langle P, Q\rangle\}$ linkkey $\langle C, D\rangle)$.

6. *The algorithm now chooses another* $\Lambda_0 = \{\sigma_9, \sigma_3, \sigma_4\}$ *as shown in layer* Λ_0 *in Figure 5.9.*

7. *Then it adds non nominal star-types* σ_7*,* σ_6 *and* σ_5 *to* Λ_1 *since* $\Omega(\sigma_3, \rho_6) = \{\sigma_7\}$*,* $\Omega(\sigma_3, \rho_7) = \{\sigma_6\}$ *and* $\Omega(\sigma_4, \rho_9) = \{\sigma_5\}$ *as shown in Figure 5.9. The algorithm continues by adding non nominal star-types* σ_7 *and* σ_8 *to* Λ_2 *since* $\Omega(\sigma_5, \rho_{12}) = \{\sigma_7\}$*,* $\Omega(\sigma_6, \rho_{10}) = \{\sigma_7\}$ *and* $\Omega(\sigma_5, \rho_{11}) = \{\sigma_8\}$*. Note that* σ_7 *is a neighbour for both* σ_5 *and* σ_6*. This phenomenon illustrates individual compression in a compressed tableau. Moreover, we can observe that* σ_7 *located in* Λ_2 *is blocked by* σ_7 *in* Λ_1*, and* σ_8 *in* Λ_2 *is blocked by* σ_6 *in* Λ_1 *since* $\mathsf{core}(\sigma_8) = \mathsf{core}(\sigma_6)$*.*

8. *Finally, the algorithm calls* pruning *which returns* true *as there exists no non blocked star-type that has no successors and thus for every individual there is still a unique star-type in* Λ_0 *(we have* $a, b \in \mathsf{core}_\mathsf{I}(\sigma_9)$*,* $g \in \mathsf{core}_\mathsf{I}(\sigma_3)$ *and* $v \in \mathsf{core}_\mathsf{I}(\sigma_4)$*). The algorithm terminates and returns* YES *which means that the input ontology is consistent. (see Figure 5.9)*

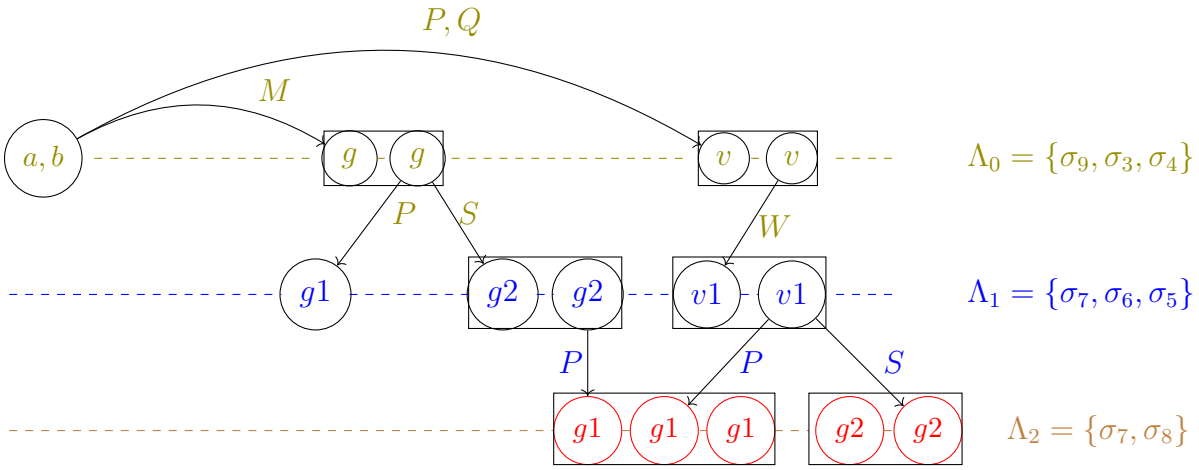


Figure 5.9 – A compressed tableau $\mathcal{CT} = \langle\langle \Lambda_0, \Lambda_1, \Lambda_2\rangle, \Omega\rangle$ for $\mathcal{O}$.

5.5 Properties of the algorithm

The goal of this section is to establish this main result of this chapter. We begin by showing that Algorithm 3 is sound and complete. Then we analyze complexity of Algorithm 3. This results in termination of the algorithm. This shows that the algorithm is a decision procedure for the consistency problem in $\mathcal{ALC}$+$\mathcal{LK}$.

5.5.1 Soundness

The goal of this subsection is to show that every compressed tableau represents a model of an ontology in $\mathcal{ALC}$+$\mathcal{LK}$.

Lemma 5 (Soundness)**.** *If Algorithm 3 returns* YES *from an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O}$ with a set of individuals* **I***, then $\mathcal{O}$ is consistent.*

Proof sketch.

Assume that Algorithm 3 returns YES. Since Algorithm 3 returns YES then due to Line 14, Algorithm 3 builds successfully a structure $\mathcal{CT}=\langle \mathbf{\Lambda}, \Omega\rangle$ with $\mathbf{\Lambda} = \langle \Lambda_0, \ldots, \Lambda_n\rangle$. To show that $\mathcal{O}$ is consistent we first show that $\mathcal{CT}$ is a compressed tableau. Then we define an interpretation $\mathcal{I}$ from $\mathcal{CT}$ and show that $\mathcal{I}$ is a model of the input ontology $\mathcal{O}$.

To show that $\mathcal{CT}$ is a compressed tableau we have to show that it satisfies Items 1, 2 and 3 of Definition 22. Indeed, each star-type in $\mathcal{CT}$ is valid due to Line 1 of Algorithm 3. In addition, it holds that (i) for each individual $a \in \mathbf{I}$ there exists a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ due to Line 3 in Algorithm 3, and Line 6 in Algorithm 2, (ii) Ω is well defined over Λ_0 due to Line 4, and the layers are correctly connected by Ω such that Λ_n contains only blocked star-types. Therefore, $\mathcal{CT}$ satisfies Item 1 in Definition 22. Moreover, due to Line 6 in Algorithm 3 each link key λ in $\mathcal{LK}$ is satisfied over Λ_0, and if C occurs in λ then $\{C, \sim C\} \cap \mathsf{core}_{\mathsf{C}}(\sigma_a) \neq \emptyset$ for all star-type $\sigma_a \in \Lambda_0$, and thus $\mathcal{CT}$ satisfies Items 2 and 3 in

Definition 22. To show that $\mathcal{O}$ is consistent, we define from $\mathcal{CT}$ an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ by unravelling $\mathcal{CT}$ and show that $\mathcal{I}$ is a model of $\mathcal{O}$. Such an interpretation associates "paths" to star-types contained in the layers $\langle \Lambda_0, ..., \Lambda_n \rangle$. Such a path is a sequence of star-types in $\mathcal{CT}$. For instance, $\mathsf{p} = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ is such a path. Its last element (σ_3) is called its last and we write $\mathsf{last}(\mathsf{p}) = \sigma_3$; its first element ($\sigma_1$) is called its root. A path containing only one element is called a root path. More precisely,

1. $\Delta^{\mathcal{I}}$ is the smallest set of *paths* built as follows:
   (a) $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}_\sigma = \langle \sigma \rangle$ for each $\sigma \in \Lambda_0$. In this case, we define $a^{\mathcal{I}} = \mathsf{p}_\sigma$ for each $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$
   (b) For each $\mathsf{p} \in \Delta^{\mathcal{I}}$ with $\mathsf{tail}(\mathsf{p}) \in \Lambda_i$ and each star-type $\sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho)$ with $\sigma' \in \Lambda_{i+1}$ and $\rho \in \mathsf{tail}(\mathsf{p})$, if σ' is not blocked then $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, \sigma' \rangle$, otherwise, $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(\sigma') \rangle$,
2. For each concept name A, $A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} \mid A \in \mathsf{core}_{\mathsf{C}}(\mathsf{tail}(\mathsf{p})\}$. For each role name R, $R^{\mathcal{I}} = \mathbf{R}_1 \cup \mathbf{R}_2 \cup \mathbf{R}_3$ where
   (a) $\mathbf{R}_1 = \{\langle \mathsf{p}_\sigma, \mathsf{p}_{\sigma'} \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}_\sigma), \rho), R \in \mathsf{tie}(\rho)\}$
   (b) $\mathbf{R}_2 = \{\langle \mathsf{p}, \mathsf{p}' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, \sigma' \rangle, \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho), R \in \mathsf{tie}(\rho),$ σ' *is not blocked*$\}$
   (c) $\mathbf{R}_3 = \{\langle \mathsf{p}, \mathsf{p}' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(\sigma') \rangle, \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho),$ $R \in \mathsf{tie}(\rho), \sigma'$ *is blocked*$\}$

The following properties are direct consequences of the construction of $\mathcal{I}$: (i) a path p may be infinite since when it reaches a blocked star-type it can be extended to its blocking star-type located at some layer Λ_i, and then extended to a star-type located at Λ_{i+1}, and so on; (ii) root paths can be arbitrarily connected via Ω while non root paths whose tails are located in the same layer are never connected via Ω; and (iii) a path never has two different prefixes. To show that $\mathcal{I}$ is a model of $\mathcal{O}$, we have to show that $\mathcal{I}$ satisfies all assertions, axioms and link keys in $\mathcal{O}$. Satisfaction of the individual and role assertions can be straightforwardly obtained from Definitions 20 and 22 and validity of the star-types used to define $\mathcal{I}$. Satisfaction of the concept assertions and GCIs can be established by proving an intermediate result which says that : for all $\mathsf{p} \in \Delta^{\mathcal{I}}$ if $C \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$ then $\mathsf{p} \in C^{\mathcal{I}}$. Satisfaction of link keys results from the three properties of $\mathcal{I}$ mentioned above. □

Proof. Assume that Algorithm 3 returns YES. Due to Line 14, Algorithm 3 builds successfully a structure $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_0, ..., \Lambda_n \rangle$. First, we show that $\mathcal{CT}$ is a compressed tableau, i.e. it satisfies Items 1, 2 and 3 in Definition 20. Indeed, each star-type in $\mathcal{CT}$ is valid due to Lines 1-2 in Algorithm 3. In addition, it holds that (i) for each individual $a \in \mathbf{I}$ there is a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ due to Line 3 in Algorithm 3, and Line 6 in Algorithm 2, (ii) Ω is well defined over Λ_0 due to Line 4, and the layers are correctly connected by Ω such that Λ_n contains only blocked star-types. Therefore, $\mathcal{CT}$ satisfies Item 1 in Definition 20. Moreover, each link key in $\mathcal{LK}$ is satisfied over Λ_0 due to Line 6 in Algorithm 3, and thus $\mathcal{CT}$ satisfies Item 2 in Definition 20. Finally, since $\mathsf{sub}(\mathcal{O})$ contains C and $\neg C$ if C occurs in a link key in $\mathcal{LK}$, we have $C \in \mathsf{core}_{\mathsf{C}}(\sigma_a)$ or $\neg C \in \mathsf{core}_{\mathsf{C}}(\sigma_a)$ for all star-type $\sigma_a \in \Lambda_0$. Hence, $\mathcal{CT}$ satisfies Item 3 in Definition 20.

To show that $\mathcal{O}$ is consistent, we define from $\mathcal{CT}$ an unravelled interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and prove that $\mathcal{I}$ is a model for $\mathcal{O}$. Such an unravelled interpretation associates paths to star-types contained in the layers $\mathbf{\Lambda} = \langle \Lambda_0, ..., \Lambda_n \rangle$ of $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$. These paths are sequences of star-types in the compressed tableau. For instance, $\mathsf{p} = \langle \sigma_1, \sigma_2, \sigma_3 \rangle$ is such a path. Its last element (σ_3) is called its tail and we write $\mathsf{tail}(\mathsf{p}) = \sigma_3$; its first element ($\sigma_1$) is called its root. A path containing only one element is called a root path.

Definition 23 (Unravelled interpretation)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ be a compressed tableau for $\mathcal{O}$ with $\mathbf{\Lambda} = \langle \Lambda_0, \cdots, \Lambda_n \rangle$.*

More precisely, $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\mathcal{O}$ unravelled from $\mathcal{CT}$ is defined as follows:

1. *$\Delta^{\mathcal{I}}$ is the smallest set of* paths *built as follows:*
   (a) *$\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}_\sigma = \langle \sigma \rangle$ for each $\sigma \in \Lambda_0$. In this case, we define $a^{\mathcal{I}} = \mathsf{p}_\sigma$ for each $a \in \mathsf{core_I}(\sigma)$*
   (b) *For each $\mathsf{p} \in \Delta^{\mathcal{I}}$ with $\mathsf{tail}(\mathsf{p}) \in \Lambda_i$ and each star-type $\sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho)$ with $\sigma' \in \Lambda_{i+1}$ and $\rho \in \mathsf{tail}(\mathsf{p})$, if σ' is not blocked then $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, \sigma' \rangle$, otherwise, $\Delta^{\mathcal{I}}$ contains a path $\mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(\sigma') \rangle$,*
2. *For each concept name A, $A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} \mid A \in \mathsf{core_C}(\mathsf{tail}(\mathsf{p}))\}$*
3. *For each role name R, $R^{\mathcal{I}} = \mathbf{R}_1 \cup \mathbf{R}_2 \cup \mathbf{R}_3$ where*
   (a) $\mathbf{R}_1 = \{\langle \mathsf{p}_\sigma, \mathsf{p}_{\sigma'} \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}_\sigma), \rho),\ R \in \mathsf{tie}(\rho)\}$
   (b) $\mathbf{R}_2 = \{\langle \mathsf{p}, \mathsf{p}' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, \sigma' \rangle, \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho), R \in \mathsf{tie}(\rho), \sigma' \text{ is not blocked}\}$
   (c) $\mathbf{R}_3 = \{\langle \mathsf{p}, \mathsf{p}' \rangle \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \mathsf{p}' = \langle \mathsf{p}, \mathsf{b}(\sigma') \rangle, \sigma' \in \Omega(\mathsf{tail}(\mathsf{p}), \rho), R \in \mathsf{tie}(\rho), \sigma' \text{ is blocked}\}$

To prove that $\mathcal{I}$ is a model of $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, we have to prove that $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$, all GCIs in $\mathcal{T}$ and all link keys in $\mathcal{LK}$.

We start by proving that $\mathcal{I}$ satisfies all assertions in $\mathcal{A}$. Assume that $a \approx b \in \mathcal{A}$. According to Item 22.1, 20.1 and 17.6 there exists a star-type $\sigma \in \Lambda_0$ such that $a, b \in \mathsf{core_I}(\sigma)$. According to the definition of $\mathcal{I}$, we have $a^{\mathcal{I}} = \mathsf{p}_\sigma$ and $b^{\mathcal{I}} = \mathsf{p}_\sigma$. Thus $a^{\mathcal{I}} = b^{\mathcal{I}}$. Assume that $R(a, b) \in \mathcal{A}$, then by Item 3 in Definition 20 there exist nominal star-types $\sigma, \sigma' \in \Lambda_0$ such that $a \in \mathsf{core_I}(\sigma)$ and $b \in \mathsf{core_I}(\sigma')$. According to Item 17.9 and 20.4, there exists a triple $\rho \in \sigma$ such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail_I}(\rho)$ and $\sigma' \in \Omega(\sigma, \rho)$. By the definition of $\mathcal{I}$, we have $a^{\mathcal{I}} = \mathsf{p}_\sigma$ and $b^{\mathcal{I}} = \mathsf{p}_{\sigma'}$, $\langle \mathsf{p}_\sigma, \mathsf{p}_{\sigma'} \rangle \in R^{\mathcal{I}}$, thus $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$. Assume $a \not\approx b \in \mathcal{A}$. By Item 20.3 there exist two star-types $\sigma, \sigma' \in \Lambda_0$ such that $a \in \mathsf{core_I}(\sigma)$ and $b \in \mathsf{core_I}(\sigma')$. Suppose that $a^{\mathcal{I}} = b^{\mathcal{I}}$ then by the definition of $\mathcal{I}$, we have $a, b \in \mathsf{core_I}(\sigma)$ but according to item 17.6 this is contradicts the validity of σ. Then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Assume $E(w) \in \mathcal{A}$. To show $w^{\mathcal{I}} \in E^{\mathcal{I}}$, we need to show a stronger claim:

$$\text{For all } \mathsf{p} \in \Delta^{\mathcal{I}},\ \text{if } C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p})) \text{ then } \mathsf{p} \in C^{\mathcal{I}} \tag{5.1}$$

$E(w) \in \mathcal{A}$ then $w \in \mathbf{I}$, by Item 20.3 there exists a star-type $\sigma \in \Lambda_0$ such that $E \in \mathsf{core_C}(\sigma)$. From the definition of the unravelled interpretation, there is some $\mathsf{p}_\sigma \in \Delta^{\mathcal{I}}$ such that $\mathsf{p}_\sigma = w^{\mathcal{I}}, \mathsf{last}(\mathsf{p}_\sigma) = \sigma$ and $E \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}_\sigma))$. So by Claim 5.1 we have $w^{\mathcal{I}} \in E^{\mathcal{I}}$.
We now show the claim 5.1. Let us proceed by induction on the length of the concept C.

1. Assume that $C = A$ with a concept name A and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. We have $C^{\mathcal{I}} = A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} | A \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))\}$ by the definition of $\mathcal{I}$. Hence, as $A \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ so $\mathsf{p} \in A^{\mathcal{I}}$.

2. Assume that $C = C_1 \sqcap C_2$ and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. We have $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is valid then by Item 2 of Definition 17 we get $C_1 \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ and $C_2 \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ then by induction hypothesis we get $\mathsf{p} \in C_1^{\mathcal{I}}$ and $\mathsf{p} \in C_2^{\mathcal{I}}$ then $\mathsf{p} \in (C_1 \sqcap C_2)^{\mathcal{I}}$.

3. Assume that $C{=}C_1 \sqcup C_2$ and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. We have $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is valid then by Item 3 of Definition 17 we get $C_1 \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ or $C_2 \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ then by induction hypothesis we have $\mathsf{p} \in C_1^{\mathcal{I}}$ or $\mathsf{p} \in C_2^{\mathcal{I}}$ then $\mathsf{p} \in (C_1 \sqcup C_2)^{\mathcal{I}}$.

4. Assume that $C = \forall R.D$ and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. We have to prove that $\mathsf{p} \in C^{\mathcal{I}}$.

   Let $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $\langle \mathsf{p}, \mathsf{p}' \rangle \in R^{\mathcal{I}}$. We have to prove that $\mathsf{p}' \in D^{\mathcal{I}}$. Since $\mathsf{last}(\mathsf{p})$ is valid then by Item 5 of Definition 14 there exists a triple $\rho \in \mathsf{last}(\mathsf{p})$ such that $R \in \mathsf{tie}(\rho)$ and $D \in \mathsf{tail_C}(\rho)$. From the definition of $\mathcal{I}$ we have two cases:

   (a) $\langle \mathsf{p}, \mathsf{p}' \rangle \in R^{\mathcal{I}}$ where $\mathsf{last}(\mathsf{p}') \in \Omega(\mathsf{last}(\mathsf{p}), \rho)$ and $\mathsf{last}(\mathsf{p}')$ is not blocked. We have $D \in \mathsf{tail}(\rho)$ and as $\mathsf{tail}(\mathsf{p}') \in \Omega(\mathsf{last}(\mathsf{p}), \rho)$ we get $\mathsf{tail}(\rho) = \mathsf{core_C}(\mathsf{last}(\mathsf{p}'))$ then $D \in \mathsf{core_C}(\mathsf{tail}(\mathsf{p}))$, by the induction hypothesis we get $\mathsf{p}' \in D^{\mathcal{I}}$.

   (b) $\langle \mathsf{p}, \mathsf{p}' \rangle \in R^{\mathcal{I}}$ where $\mathsf{last}(\mathsf{p}')$ is blocked by $\mathsf{b}(\mathsf{last}(\mathsf{p}'))$. Due to the blocking condition, it follows that $\mathsf{core_C}(\mathsf{last}(\mathsf{p}')) \subseteq \mathsf{core_C}(\mathsf{b}(\mathsf{last}(\mathsf{p}')))$ we also have $\mathsf{tail}(\rho) = \mathsf{core_C}(\mathsf{last}(\mathsf{p}'))$ as $\mathsf{last}(\mathsf{p}') \in \Omega(\mathsf{last}(\mathsf{p}), \rho)$, then $D \in \mathsf{core_C}(\mathsf{b}(\mathsf{last}(\mathsf{p}')))$ so by the induction hypothesis we get $\mathsf{p}' \in D^{\mathcal{I}}$.

   In both cases, we get that $\mathsf{p} \in C^{\mathcal{I}}$.

5. Assume that $C = \exists R.D$ and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. According to Item 4 in Definition 17 there is a triple $\rho \in \mathsf{last}(\mathsf{p})$ such that $R \in \mathsf{tie}(\rho)$ and $D, C_{\exists R.D} \in \mathsf{tail_C}(\rho)$.

   According to Algorithm 3 there exists a star-type $\mathsf{last}(\mathsf{p}') \in \Omega(\mathsf{last}(\mathsf{p}), \rho)$. From the definition of $\mathcal{I}$ we have two cases:

   (a) $\mathsf{last}(\mathsf{p}')$ is not blocked and we have $\mathsf{tail_C}(\rho){=}\mathsf{core_C}(\mathsf{last}(\mathsf{p}'))$ as $\mathsf{last}(\mathsf{p}') \in \Omega(\mathsf{last}(\mathsf{p}), \rho)$ so $D, C_{\exists R.D} \in \mathsf{tail_C}(\rho)$, then $D, C_{\exists R.D} \in \mathsf{core_C}(\sigma')$. By induction hypothesis $\mathsf{p}' \in D^{\mathcal{I}}$.

   (b) $\mathsf{last}(\mathsf{p}')$ is blocked by $\mathsf{b}(\mathsf{last}(\mathsf{p}'))$ then $\mathsf{core_C}(\mathsf{last}(\mathsf{p}')) \subseteq \mathsf{core_C}(\mathsf{b}(\mathsf{last}(\mathsf{p}')))$, and $D \in \mathsf{core_C}(\mathsf{tail}(\mathsf{p}'))$ as $\mathsf{tail_C}(\rho) = \mathsf{core_C}(\mathsf{last}(\mathsf{p}'))$ then $D \in \mathsf{core_C}(\mathsf{b}(\mathsf{last}(\mathsf{p}')))$ so by induction hypothesis $\mathsf{p}' \in D^{\mathcal{I}}$.

6. Assume that $C = {\sim}D$ and $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. We have to show that $\mathsf{p} \notin D^{\mathcal{I}}$. If D is a concept name then, due to validity of $\mathsf{last}(\mathsf{p})$, $D \notin \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. By the definition of $D^{\mathcal{I}}$, we get $\mathsf{p} \notin D^{\mathcal{I}}$. We can proceed similarly by induction on the length of D to show that $\mathsf{p} \notin D^{\mathcal{I}}$ if D is not atomic concept.

We now show that $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$. Let $C \sqsubseteq D$ and $\mathsf{p} \in C^{\mathcal{I}}$ we have to show that $\mathsf{p} \in D^{\mathcal{I}}$. Due to the validity of $\mathsf{last}(\mathsf{p})$ we have $\sim C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ or $D \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$. If $\sim C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ then we get $\mathsf{p} \notin C^{\mathcal{I}}$, which contradicts $\mathsf{p} \in C^{\mathcal{I}}$. Then $D \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$, and by Claim 5.1, we get $\mathsf{p} \in D^{\mathcal{I}}$.

We now show that $\mathcal{I}$ satisfies each link key in $\mathcal{LK}$. Assume that $\lambda = (\{\langle P_i, Q_i\rangle\}_{i=1}^{n}$ linkkey $\langle C, D\rangle) \in \mathcal{LK}$. Let us prove that $\mathcal{I}$ satisfies λ. Let $\mathsf{p}, \mathsf{q}, \mathsf{p}_1, \ldots, \mathsf{p}_n \in \Delta^{\mathcal{I}}$ such that $\mathsf{p} \in C^{\mathcal{I}}$, $\mathsf{q} \in D^{\mathcal{I}}$, and $(\mathsf{p}, \mathsf{p}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{p}_i) \in Q_i^{\mathcal{I}}$ for $1 \leq i \leq n$. We have to prove that $\mathsf{p} = \mathsf{q}$. By the definition of $\mathcal{I}$, we have $\mathsf{last}(\mathsf{p}), \mathsf{last}(\mathsf{q}) \in \Lambda_0$, and $\mathsf{last}(\mathsf{p}_i) \in \Lambda_0$ for all $1 \leq i \leq n$. According to Item 3 of Defintion 22 we have $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ or $\sim C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ and $D \in \mathsf{core_C}(\mathsf{last}(\mathsf{q}))$ or $\sim D \in \mathsf{core_C}(\mathsf{last}(\mathsf{q}))$. If $\sim C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ or $\sim D \in \mathsf{core_C}(\mathsf{last}(\mathsf{q}))$ then by Claim 5.1 we get $\mathsf{p} \in \sim C^{\mathcal{I}}$ and $\mathsf{q} \in \sim D^{\mathcal{I}}$ which contradicts $\mathsf{p} \in C^{\mathcal{I}}$ and $\mathsf{q} \in D^{\mathcal{I}}$ so $C \in \mathsf{core_C}(\mathsf{last}(\mathsf{p}))$ and $D \in \mathsf{core_C}(\mathsf{last}(\mathsf{q}))$ and $\langle \mathsf{last}(\mathsf{p}), \mathsf{last}(\mathsf{p}_i)\rangle \in P_i^{\mathcal{I}}$, $\langle \mathsf{last}(\mathsf{q}), \mathsf{last}(\mathsf{p}_i)\rangle \in {Q_i}^{\mathcal{I}}$. According to Item 2 in Definition 22, λ is satisfied. We have checked that $\mathcal{I}$ verifies all assertions, axioms and link keys of $\mathcal{O}$. Therefore, $\mathcal{I}$ is a model of $\mathcal{O}$. This completes the proof. □

5.5.2 Completeness

In this subsection, we prove that the existence of a model of an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O}$ allows Algorithm 3 to return YES, i.e. it can build a compressed tableau for $\mathcal{O}$.

Lemma 6. *If an $\mathcal{ALC}$+$\mathcal{LK}$ ontology $\mathcal{O}$ with a set of individuals* $\mathbf{I}$ *is consistent, then Algorithm 3 returns* YES *from $\mathcal{O}$.*

Proof sketch. Since $\mathcal{O}$ is consistent, there is a model $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ of $\mathcal{O}$. We now show that Algorithm 3 returns YES, i.e. it finds a non empty structure $\mathcal{CT}$ after applying pruning to $\mathcal{CT}$ by Line 14 in Algorithm 3.

1. Algorithm 3 (Lines 1-2) builds all possible valid star-types from $\mathcal{O}$. Then, it puts nominal and non nominal star-types in $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$ respectively. The success of this step does not depend on the existence of $\mathcal{I}$. Line 3 chooses a subset of nominal valid star-types $\Lambda_0 \subseteq \Sigma_N(\mathcal{O})$ such that for each $a \in \mathbf{I}$ there is a unique star-type σ in Λ_0 such that $a \in \mathsf{core_I}(\sigma)$. Thanks to the existence of $\mathcal{I}$, this step is always successful. Indeed, since $\mathcal{I}$ is a model of $\mathcal{O}$, there is some $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each $a \in \mathbf{I}$ from which we builds a star-type $\sigma(a^{\mathcal{I}})$ such that :
   (a) $b \in \mathsf{core_I}(\sigma(a^{\mathcal{I}}))$ iff $b \in a^{+}$, and $C \in \mathsf{core_C}(\sigma(a^{\mathcal{I}}))$ iff $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for each $C \in \mathsf{sub}(\mathcal{O})$.
   (b) For each $\exists R.C \in \mathsf{core_C}(\sigma(a^{\mathcal{I}}))$, there is an individual $t \in \Delta^{\mathcal{I}}$ such that $(s,t) \in R^{\mathcal{I}}$ and $t \in C^{\mathcal{I}}$ since $\mathcal{I}$ is a model. If $\sigma(a^{\mathcal{I}})$ has already a triple ρ such that $R \in \mathsf{tie}(\rho)$ and $C \in \mathsf{core_C}(\sigma(a^{\mathcal{I}}))$, then we add $C_{\exists R.C}$ to $\mathsf{tail_C}(\rho)$. Otherwise, it adds to $\sigma(a^{\mathcal{I}})$ a triple ρ with $\mathsf{head}(\rho) = \langle \mathsf{core_I}(\sigma(a^{\mathcal{I}})), \mathsf{core_C}(\sigma(a^{\mathcal{I}}))\rangle$, $\mathsf{tie}(\rho) = \{R\} \cup \{S \in \mathbf{R} \mid (s,t) \in S^{\mathcal{I}}\}$, and $\mathsf{tail_C}(\rho) = \{C, C_{\exists R.C}\}$. To complete it, we add to $\mathsf{tail_C}(\rho)$ all concept $D \in \mathsf{sub}(\mathcal{O})$ such that $t \in D^{\mathcal{I}}$, and add to $\mathsf{tail_I}(\rho)$ all individual $b \in \mathbf{I}$ such that $b^{\mathcal{I}} = t$.
   (c) For each $R(a,b) \in \mathcal{A}$, we have $(a,b) \in R^{\mathcal{I}}$ since $\mathcal{I}$ is a model. If $\sigma(a^{\mathcal{I}})$ has no triple ρ such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail_I}(\rho)$, then we add to $\sigma(a^{\mathcal{I}})$ a triple ρ with

$\mathsf{head}(\rho) = \langle \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I})), \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))\rangle$, $\mathsf{tie}(\rho) = \{R\} \cup \{S \in \mathbf{R} \mid (a^\mathcal{I}, b^\mathcal{I}) \in S^\mathcal{I}\}$. To complete it, we add to $\mathsf{tail}_\mathsf{C}(\rho)$ all concept $D \in \mathsf{sub}(\mathcal{O})$ such that $t \in D^\mathcal{I}$, and add to $\mathsf{tail}_\mathsf{I}(\rho)$ all individual $b' \in \mathbf{I}$ such that $b'^\mathcal{I} = b^\mathcal{I}$.

(d) We add $\sigma(a^\mathcal{I})$ to Λ_0. If there are $\sigma(a^\mathcal{I}), \sigma(b^\mathcal{I}) \in \Lambda_0$ and $\rho \in \sigma(a^\mathcal{I})$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$ and $b \in \mathsf{core}_\mathsf{I}(\sigma(b^\mathcal{I})) \cap \mathsf{tail}_\mathsf{I}(\rho)$, then we define $\Omega(\sigma(a^\mathcal{I}), \rho) = \{\sigma(b^\mathcal{I})\}$.

By construction, each $\sigma(a^\mathcal{I})$ is valid. By construction Λ_0 satisfies the conditions of Lines 4 and 6 in Algorithm 3.

2. From a current layer Λ_i, each star-type $\sigma(s) \in \Lambda_i$ with $s \in \Delta^\mathcal{I}$ and each triple $\rho \in \sigma(s)$, Algorithm 3 can always choose all valid star-types from $\Sigma_C(\mathcal{O})$ to put on the next layer thanks to the existence of an individual $t \in \Delta^\mathcal{I}$ such that $(s, t) \in R^\mathcal{I}$ with $R \in \mathsf{tie}(\rho)$. By construction, for each star-type $\sigma(s) \in \Lambda_i$ and each triple $\rho \in \sigma$, there is always some $\sigma(t) \in \Sigma_C(\mathcal{O})$ such that $\mathsf{core}(\sigma(t)) = \mathsf{tail}(\sigma(s))$. This implies that pruning returns true when Algorithm 3 reaches a layer Λ_n which contains either blocked star-types, or dummy star-types. □

Proof. Since $\mathcal{O}$ is consistent, there is a model $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I}\rangle$ of $\mathcal{O}$. We show that Algorithm 3 returns a compressed tableau $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega\rangle$ of $\mathcal{O}$ with $\mathbf{\Lambda} = \langle \Lambda_0, \cdots, \Lambda_n\rangle$.

First, we show that if Algorithm 3 returns a non empty structure $\mathcal{CT}$ then it satisfies Items 1, 2 and 3 of Definition 20. Indeed, each star-type in $\mathcal{CT}$ is valid due to Lines 1-2 in Algorithm 3. In addition, for each individual $a \in \mathbf{I}$ there is a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma_a)$ due to Line 3, Ω is well defined over Λ_0 due to Line 4, and each link key in $\mathcal{LK}$ is satisfied over Λ_0 due to Line 6. Furthermore, the layers are correctly connected by Ω where Λ_n contains only blocked or dummy star-types. We now show that Algorithm 3 returns a non empty structure $\mathcal{CT}$.

1. Lines 1 and 2 of Algorithm 3 builds two sets of valid star-types $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$ as follows. It picks up from $\mathsf{sub}(\mathcal{O})$ two subsets of concepts (recall that an individual a is considered as a concept $\{a\}$), and from $\mathbf{R}$ a subset of roles for creating a triple $\rho = \langle\langle X, U\rangle, P, \langle Y, V\rangle\rangle$ where X, Y contain all individuals, and U, V contain all concepts. As a result, this task generates the set of all triples, denoted $\mathbf{TR}$. Then, for each subset L of $\mathsf{sub}(\mathcal{O})$ and each concept of the form $\exists R.C \in L$, take a triple ρ from $\mathbf{TR}$ such that $\mathsf{head}(\rho) = L$ and $R \in \mathsf{tie}(\rho)$, and $\{C, C_{\exists R.C}\} \subseteq \mathsf{tail}_\mathsf{C}(\rho)$ and for each role assertion $R(a, b) \in \mathcal{A}$ such that $a \in L$, we take a triple ρ from $\mathbf{TR}$ such that $\mathsf{head}(\rho) = L$ and $R \in \mathsf{tie}(\rho)$, and $b \in \mathsf{tail}_\mathsf{I}(\rho)$. We obtain a candidate star-type σ for validity. To check validity of σ, the algorithm checks each property in Definition 17. Then, it puts nominal and non nominal star-types in $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$, respectively.

2. Line 3 chooses a set of valid nominal star-types $\Lambda_0 \subseteq \Sigma_N(\mathcal{O})$ that satisfies the condition in Lines 4 and 6 in Algorithm 3. We show that such a choice is always possible thanks to the existence of $\mathcal{I}$. For this purpose, it suffices to show that there exists such a set $\Lambda_0 \subseteq \Sigma_N(\mathcal{O})$ by building it from $\mathcal{I}$.

   Since $\mathcal{I}$ is a model of $\mathcal{O}$, there is some $a^\mathcal{I} \in \Delta^\mathcal{I}$ for each $a \in \mathbf{I}$. For each $a^\mathcal{I}$, we builds a star-type $\sigma(a^\mathcal{I})$ such that :

   (a) $b \in \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$ iff $b \in a^+$.

(b) $C \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$ iff $a^\mathcal{I} \in C^\mathcal{I}$ for each $C \in \mathsf{sub}(\mathcal{O})$.

(c) For each $\exists R.C \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$, there is an individual $t \in \Delta^\mathcal{I}$ such that $(s,t) \in R^\mathcal{I}$ and $t \in C^\mathcal{I}$ since $\mathcal{I}$ is a model. If $\sigma(a^\mathcal{I})$ has already a triple ρ such that $R \in \mathsf{tie}(\rho)$ and $C \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$, then we add $C_{\exists R.C}$ to $\mathsf{tail}_\mathsf{C}(\rho)$. Otherwise, it adds to $\sigma(a^\mathcal{I})$ a triple ρ with $\mathsf{head}(\rho) = \langle \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I})), \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))\rangle$, $\mathsf{tie}(\rho) = \{R\} \cup \{S \in \mathbf{R} \mid (s,t) \in S^\mathcal{I}\}$, and $\mathsf{tail}_\mathsf{C}(\rho) = \{C, C_{\exists R.C}\}$. To complete it, we add to $\mathsf{tail}_\mathsf{C}(\rho)$ all concept $D \in \mathsf{sub}(\mathcal{O})$ such that $t \in D^\mathcal{I}$, and add to $\mathsf{tail}_\mathsf{I}(\rho)$ all individual $b \in \mathbf{I}$ such that $b^\mathcal{I} = t$.

(d) For each $R(a,b) \in \mathcal{A}$, we have $(a,b) \in R^\mathcal{I}$ since $\mathcal{I}$ is a model. If $\sigma(a^\mathcal{I})$ has no triple ρ such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}_\mathsf{I}(\rho)$, then we add to $\sigma(a^\mathcal{I})$ a triple ρ with $\mathsf{head}(\rho) = \langle \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I})), \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))\rangle$, $\mathsf{tie}(\rho) = \{R\} \cup \{S \in \mathbf{R} \mid (a^\mathcal{I}, b^\mathcal{I}) \in S^\mathcal{I}\}$. To complete it, we add to $\mathsf{tail}_\mathsf{C}(\rho)$ all concept $D \in \mathsf{sub}(\mathcal{O})$ such that $t \in D^\mathcal{I}$, and add to $\mathsf{tail}_\mathsf{I}(\rho)$ all individual $b' \in \mathbf{I}$ such that $b'^\mathcal{I} = b^\mathcal{I}$.

(e) We add $\sigma(a^\mathcal{I})$ to Λ_0.

(f) If there are $\sigma(a^\mathcal{I}), \sigma(b^\mathcal{I}) \in \Lambda_0$ and $\rho \in \sigma(a^\mathcal{I})$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$ and $b \in \mathsf{core}_\mathsf{I}(\sigma(b^\mathcal{I})) \cap \mathsf{tail}_\mathsf{I}(\rho)$, then we define $\Omega(\sigma(a^\mathcal{I}), \rho) = \{\sigma(b^\mathcal{I})\}$.

We now show the following properties.

- $\sigma(a^\mathcal{I})$ is valid. We have to check all properties in Definition 17. Properties 1-9 hold due to the construction of $\sigma(a^\mathcal{I})$, the proof of properties 1, 2, 3, 6, 7 and 8 concerns only the cores of star-types. Similarly, this can be shown for the tails of the triples of the star-types.

  (a) Assume that $C \sqsubseteq D \in \mathcal{T}$ and $\sigma(a^\mathcal{I})$ with $C \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$. We have $a \in C^\mathcal{I}$ then $a \in D^\mathcal{I}$ since $\mathcal{I}$ is a model. Then by construction we have $D \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$.

  (b) If $C_1 \sqcap C_2 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$. We have $a \in (C_1 \sqcap C_2)^\mathcal{I}$, $\mathcal{I}$ is the model then $a \in C_1^\mathcal{I}$ and $a \in C_2^\mathcal{I}$. Then by construction we have $C_1 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$ and $C_2 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$.

  (c) If $C_1 \sqcup C_2 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$. We have $a \in (C_1 \sqcup C_2)^\mathcal{I}$, $\mathcal{I}$ is the model then $a \in C_1^\mathcal{I}$ or $a \in C_2^\mathcal{I}$. Then by construction we have $C_1 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$ or $C_2 \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$.

  (d) Suppose that we have $a \approx b \in \mathcal{A}$ then $b \in a^+$, by construction we have $b \in \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$.

  (e) Assume that we have $a \not\approx b \in \mathcal{A}$ and $\{a,b\} \subseteq \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$ then we have $b \in a^+$ which is a contradiction. Then by construction $a, b \not\subseteq \mathsf{core}_\mathsf{I}(\sigma(a^\mathcal{I}))$.

  (f) Assume that $\{A, \sim A\} \subseteq \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$, then we have $a \in A^\mathcal{I}$ and $a \in (\sim A)^\mathcal{I}$ but $\mathcal{I}$ is a model of $\mathcal{O}$, so $a \in (\sim A)^\mathcal{I}$ and hence by construction $\sim A \notin \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$.

  For instance, we check Property 4. Since each concept $\exists R.C \in \mathsf{core}_\mathsf{C}(\sigma(a^\mathcal{I}))$ is handled once by construction, there is a unique triple ρ added to $\sigma(a^\mathcal{I})$ such that $\{C, C_{\exists R.C}\} \subseteq \mathsf{tail}_\mathsf{C}(\rho)$. Similarly, we can check Property 9. Hence, $\sigma(a^\mathcal{I})$ is pre-valid. By construction, $\sigma(a^\mathcal{I})$ is also valid since if any triple is removed from $\sigma(a^\mathcal{I})$ then $\sigma(a^\mathcal{I})$ will no longer be pre-valid.

- $\sigma(a^{\mathcal{I}}) \in \Sigma_N(\mathcal{O})$. This holds since $\sigma(a^{\mathcal{I}})$ is valid, and each concept and role added to $\sigma(a^{\mathcal{I}})$ is contained in $\mathsf{sub}(\mathcal{O})$ and $\mathbf{R}$.
- Let $\sigma(a^{\mathcal{I}}), \sigma(b^{\mathcal{I}}) \in \Lambda_0$ with $\rho \in \sigma(a^{\mathcal{I}})$ such that $b \in \mathsf{tail}_{\mathsf{I}}(\rho)$. By construction, $b \in \mathsf{core}_{\mathsf{I}}(\sigma(b^{\mathcal{I}}))$, and $\sigma(b^{\mathcal{I}}) \in \Omega(\sigma(a^{\mathcal{I}}), \rho)$. So Λ_0 satisfies Item 1 of Definition 22.
- Λ_0 satisfies Item 2, Definition 22. Let λ be a link key. If λ entails some $a \approx b$ then we have $a^{\mathcal{I}} = b^{\mathcal{I}}$ since $\mathcal{I}$ is a model of λ. By construction, $\sigma(a^{\mathcal{I}}) = \sigma(b^{\mathcal{I}})$.

3. For each non blocked star-type $\sigma \in \Lambda_i$ with $i \geq 0$ and $\rho \in \sigma$, algorithm 3 can determine all valid star-types σ', and add them to $\Omega(\sigma, \rho)$ and Λ_{i+1} as described in lines $8 \rightarrow 13$ in algorithm 3. However, the algorithm can due to the non-determinism add a valid star-type which cannot be matched to another valid star-type and it is non blocked then Definition 28 will not be satisfied. However, the algorithm can detect by pruning, algorithm 2 such star-types and removes them from the compressed tableau as well as there predecessors, such removal may lead to remove a nominal star-type by the progressive manner such removal may violate item 3 in Definition 20, for that the algorithm checks again that for every individual there is a star-type. By construction, there is some $t \in \Delta^{\mathcal{I}}$ such that $\mathsf{tail}_{\mathsf{C}}(\rho) = \{C \mid C \in \mathsf{sub}(\mathcal{O}) \wedge t \in C^{\mathcal{I}}\}$. By using the same procedure described in the previous step, it can build $\sigma(t)$ such that $\mathsf{core}_{\mathsf{C}}(\sigma(t)) = \mathsf{tail}_{\mathsf{C}}(\rho)$. If there is some $\sigma' \in \Lambda_{i+1}$ such that $\sigma' = \sigma(t)$ then it adds σ' to $\Omega(\sigma, \rho)$, otherwise, add $\sigma(t)$ to $\Omega(\sigma, \rho)$ and Λ_{i+1}. By construction, we have $\sigma(t) \in \Sigma_C(\mathcal{O})$.

4. Algorithm 3 can find some Λ_n such that each star-type $\sigma \in \Lambda_n$ is blocked. Since there are at most $2^{|\mathsf{sub}(\mathcal{O})|}$ different subsets of $\mathsf{sub}(\mathcal{O})$, the algorithm terminates when reaching a layer Λ_n with $n \leq 2^{C_1 \ell^2}$ such that each star-type in Λ_n is blocked or dummy.

5. The obtained layers of star-types $\langle \Lambda_0, \cdots, \Lambda_n \rangle$ form a compressed tableau for $\mathcal{O}$. We have to check all conditions in Definitions 20 and 22. All conditions in Definition 20 are straightforward by construction. We show Condition 2 in Definitions 22. Let $\lambda = \{\langle P_i, Q_i \rangle\}_{i=1}^{n} \mathsf{linkkey}\langle C, D \rangle$ be a link key. Assume that there are star-types σ, σ' and $\sigma_1, \cdots, \sigma_n$ with $\rho_1, \cdots, \rho_n \in \sigma$ and $\rho'_1, \cdots, \rho'_n \in \sigma'$ such that $C \in \mathsf{core}_{\mathsf{C}}(\sigma)$, $D \in \mathsf{core}_{\mathsf{C}}(\sigma')$ $P_i \in \mathsf{tie}(\rho_i)$, $Q_i \in \mathsf{tie}(\rho'_i)$, and $\sigma_i \in \Omega(\sigma, \rho_i) \cap \Omega(\sigma', \rho'_i)$ for $1 \leq i \leq n$. We have shown above that λ is satisfied over Λ_0. We show Condition 3 in Definitions 22. By construction, there are individuals $t, t' \in \Delta^{\mathcal{I}}$ such that $\sigma = \sigma(t)$ and $\sigma' = \sigma(t')$. Since $\mathcal{I}$ is a model, we have $t \in C^{\mathcal{I}}$ or $t \in (\neg C)^{\mathcal{I}}$, and $t' \in D^{\mathcal{I}}$ or $t' \in (\neg D)^{\mathcal{I}}$. By the construction of σ and σ', we have $\mathsf{core}_{\mathsf{C}}(\sigma) \cap \{C, \neg C\} \neq \emptyset$, and $\mathsf{core}_{\mathsf{C}}(\sigma') \cap \{D, \neg D\} \neq \emptyset$. □

□

5.5.3 Complexity

This subsection analyzes and provides the complexity of the Algorithm 3 for checking consistency of an ontology in $\mathcal{ALC}$+$\mathcal{LK}$. We begin by formulating and proving a lemma which provides fundamental elements for analyzing the complexity of the algorithm.

Lemma 7. *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{LK}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ be a compressed-tableau for $\mathcal{O}$ with $\mathbf{\Lambda} = \langle \Lambda_0, \cdots, \Lambda_n \rangle$. The size of $\mathcal{CT}$ is bounded by an exponential function in the size of $\mathcal{O}$.*

Proof. We denote $\ell = \|\mathcal{O}\|$. Hence, $r = |\mathbf{R}| \leq \ell$.

By construction, $\mathsf{sub}(\mathcal{O})$ is the set of all sub-concepts occurring in $\mathcal{O}$ and all concepts of the form $C_{\exists R.C}$ added by Item 4 in Definition 17. It holds that if a concept is represented as string then a sub-concept is represented as a substring. The number of substrings build from a string of size ℓ is $\ell(\ell+1)/2$. Then the number of sub-concepts is bounded by $\ell(\ell+1)/2$. In addition, the number of concepts of the form $C_{\exists R.C}$ is bounded by the number of existential restrictions of the form $\exists R.C$ in $\mathcal{O}$, then it is bounded by ℓ, the size of the input ontology. We get that $c = |\mathsf{sub}(\mathcal{O})| \leq \ell(\ell+1)/2 + \ell \leq C_1\ell^2$ where $C_1 = 2$ for all $\ell \geq 2$.

We now calculate the size of a triple ρ built from $\mathcal{O}$. Since the size of a concept or role is bounded by ℓ, the size of a triple, composed from two pairs of concepts and individuals built from $\mathcal{O}$ and a set of roles denoted $\|\rho\|$, is bounded by $(2c\ell + r\ell) \leq (2C_1\ell^3 + \ell^2) < (2C_1 + 1)\ell^3$ for all $\ell \geq 2$. Hence, $\|\rho\| < C_2\ell^3$ with $C_2 = 2C_1 + 1$.

We now calculate the size of a star-type σ built from $\mathcal{O}$. According to Definition 20, all star-types in $\mathcal{CT}$ are valid. Due to Property 10, Definition 17, the number of triples of a valid star-type σ is bounded by $|\mathsf{core}(\sigma)| + |\mathcal{A}| \leq C_1\ell^2 + \ell \leq 2C_1\ell^2$ for all $\ell \geq 2$. It follows that the size of a valid star-type, denoted by $\|\sigma\|$, is bounded by $2C_1\ell^2\|\rho\| \leq C_3\ell^5$ with $C_3 = 2C_1(2C_1 + 1)$ for all $\ell \geq 2$.

Finally, we calculate the size of the compressed tableau $\mathcal{CT}$. To calculate the size of $\mathcal{CT}$ we have to calculate the number of star-types in each layer Λ_i, denoted $|\Lambda_i|$, with $0 \leq i \leq n$, and the number of layers n. Since each valid star-type has at most $2C_1\ell^2$ triples, we obtain $|\Lambda_i| \leq 2^{2C_1\ell^2}$. Moreover, any star-type in a layer Λ_k with $k = 2^{C_1\ell^2}$ must be blocked since, otherwise, there must exist k star-types whose cores are different, which is not possible. This implies that $n \leq 2^{C_1\ell^2}$. Hence, the size of $\mathcal{CT}$ is bounded by $n|\Lambda_i|\|\sigma\| \leq 2^{C_1\ell^2}2^{2C_1\ell^2}C_3\ell^5$, we have $2^{C_1\ell^2}2^{2C_1\ell^2}C_3\ell^5 \leq 2^{C\ell^2}$ where C is a constant large enough. This completes the proof. □

Lemma 8 (Complexity)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals* **I***. Algorithm 3 runs in exponential time in the size of $\mathcal{O}$.*

Proof sketch. Algorithm 3 first builds the set of all valid star-types from $\mathcal{O}$. It picks up from $\mathsf{sub}(\mathcal{O})$ two subsets of concepts, and from $\mathbf{R}$ a subset of roles for building a triple. According to Lemma 16 (in the proof), the cardinality of $\mathsf{sub}(\mathcal{O})$ and $\mathbf{R}$ is polynomial in the size of $\mathcal{O}$. This implies that $|\mathbf{TR}|$ is exponential in the size of $\mathcal{O}$ where $\mathbf{TR}$ denotes the set of all triples. Moreover, according to Lemma 16 again, a valid star-type contains at most a polynomial number of triples from $\mathbf{TR}$. Hence, the algorithm can choose a polynomial number of triples from $\mathbf{TR}$ to form a valid star-type. Thus, the complexity of this step (Lines 1-2) for building $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$, is exponential in the size of $\mathcal{O}$. Then, Line 3 chooses a polynomial set Λ_0 of nominal valid star-types from $\Sigma_N(\mathcal{O})$ such that for each individual $a \in \mathbf{I}$ there is a unique star-type $\sigma_a \in \Lambda_0$ whose core contains a. Thus, there is at most an exponential number of different choices. The algorithm searches in $\Sigma_N(\mathcal{O})$ for establishing Ω for each nominal star-type in Λ_0, and checks all link key assertions over Λ_0. Since the size of $\Sigma_N(\mathcal{O})$ is exponential, the complexity of this step is also exponential. After this step, the algorithm traverses each triple of each star-type of each layer Λ_i and $\Sigma_C(\mathcal{O})$ to build the next layer. According to Lemma 16, the cardinality of Λ_i is exponential in the size of $\mathcal{O}$. Hence, the complexity of this step is exponential in the size of $\mathcal{O}$. Finally, the algorithm calls pruning which traverses each star-type of each layer Λ_i and removes each non-blocked star-type without successor from each

layer and then checks that for each individual $a \in \mathbf{I}$ there is a unique star-type in Λ_0 whose core contains a. Hence the overall complexity of the algorithm is exponential. □

Proof. Assume that $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is built by the algorithm 3 with $\mathbf{\Lambda} = \langle \Lambda_0, \cdots, \Lambda_n \rangle$. We denote $\ell = \|\mathcal{O}\|$. Hence, $r = |\mathbf{R}| \leq \ell$ and $q = |\mathbf{I}| \leq \ell$. We analyze the complexity of each step of the algorithm.

1. Complexity of step 1 (lines 1 and 2): construction of $\Sigma_N(\mathcal{O})$ and $\Sigma_C(\mathcal{O})$.

   (a) Construction of all subsets of $\mathsf{sub}(\mathcal{O})$: $\mathsf{sub}(\mathcal{O})$ is the set of all sub-concepts occurring in $\mathcal{O}$. The size of $\mathcal{O}$ denoted by $|\mathcal{O}|$ is less than ℓ. Let c be the size of $\mathsf{sub}(\mathcal{O})$. It holds that if a concept is represented as string then a sub-concept is represented as a substring, then c is bounded by $\ell(\ell+1)/2 + \ell$, and approximated by $C_1\ell^2$, where C_1 is a constant that is large enough. To build the set of all subconcepts we have to loop over every concept s in $\mathsf{sub}(\mathcal{O})$ and add every subconcept of s to $\mathsf{sub}(\mathcal{O})$ so the complexity of this step is $C_1\ell^2$

   (b) Complexity of construction of all triples: each triple is composed of two pairs of sets of concepts and individuals and a set of roles. Building the set of subsets of concepts and individuals and the set of all subsets of roles needs respectively 2^{2c} and 2^r. Building the sets of all triples is then bounded by $2^{2c}2^r \leq 2^{2C_1\ell^2}2^\ell \leq 2^{2C_1\ell^2+\ell}$.

   (c) Construction of all star-types: For each subset L of $\mathsf{sub}(\mathcal{O})$ and each concept of the form $\exists R.C \in \mathsf{L}$, take a triple ρ from $\mathbf{TR}$ such that $\mathsf{head}(\rho) = \mathsf{L}$ and $R \in \mathsf{tie}(\rho)$, and $\{C, C_{\exists R.C}\} \subseteq \mathsf{tail}_\mathsf{C}(\rho)$ and for each role assertion $R(a,b) \in \mathcal{A}$ for each individual $a \in L$ take a triple ρ from $\mathbf{TR}$ such that $\mathsf{head}(\rho) = L$ and $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}_\mathsf{I}(\rho)$. Since $c = \mathsf{sub}(\mathcal{O})$ then complexity of this step is bounded by $c \times |\mathbf{TR}|$. We obtain a candidate star-type σ and we have now to check its validity.

   (d) Checking validity of a star-type σ: let $\mathsf{L_m}$ be a subset of $\mathsf{sub}(\mathcal{O})$, such that $\mathsf{L_m} \in \mathsf{core_C}(\sigma)$ or $\mathsf{L_m} \in \mathsf{tail_C}(\rho)$, for every $\rho \in \sigma$. We need to check all properties in Definition 17. For each of the properties 1, 2 and 3 we need to check the core and the tail of each concept. Each checking task needs at most $|\mathsf{L_m}|^2 + m|\mathsf{L_m}|^2$ operations, where m is the number of triples in σ. Property 4 and 9 is guaranteed by (c). Checking property 5 concerns only the core of σ needs at most $m|\mathsf{L_m}|^2$ operations. For property 6 and property 7 we need at most $\ell q^2 + m\ell q^2$ operations each. Checking property 8 needs at most $|\mathsf{L_m}|^2 + m|\mathsf{L_m}|^2$ operations. So this step executes in $3(|\mathsf{L_m}|^2 + m|\mathsf{L_m}|^2) + m|\mathsf{L_m}|^2 + 2(\ell q^2 + m\ell q^2) + (|\mathsf{L_m}|^2 + m|\mathsf{L_m}|^2) = 4|\mathsf{L_m}|^2 + 5m|\mathsf{L_m}|^2 + 2\ell q^2 + 2m\ell q^2$ we have $|\mathsf{L_m}| \leq C\ell^2$ and $m \leq \ell$ then the complexity of this step is $4C^2\ell^4 + 5\ell^5C^2 + 2\ell q^2 + \ell^2 q^2$.

   (e) There are at most $2^n 2^{\mathsf{L_m}}$ different valid star-types where $\mathsf{L_m}$ is the max cardinality of a set of concepts. Because we have $2^{\mathsf{L_m}}$ different set of concepts from which we can build 2^n star-types, where n is the maximal number of existential restrictions in each set of concepts plus the maximal number of role assertions in $\mathcal{A}$ so $n \leq \ell$. Then we have at most $2^{C_1\ell^2+\ell}$ star-types. Then we obtain that $|\Sigma_N(\mathcal{O}) \cup \Sigma_C(\mathcal{O})| \leq 2^{C_1\ell^2+\ell}$.

2. Complexity of lines 3→22.

(a) Line 3 uses a vector $(w_1, \cdots, w_k)$ to represent a possible choice of Λ_0 where each w_i corresponds to a star-type containing an individual a_i. So, $k \leq q \leq \ell$. Moreover, each w_i takes at most 2^ℓ different values. Hence, there are at most $(2^\ell)^\ell$ different vectors $(w_1, \cdots, w_k)$. This implies that line 3 performs at most $(2^\ell)^\ell$ operations.

(b) Lines 4-5 need to traverse each star-type σ in Λ_0, each triple ρ in σ, for checking if there exists a σ' in Λ_0 such that $\mathsf{tail}(\rho)$ equals $\mathsf{core}(\sigma')$. So the complexity of this step is bounded by $|\Lambda_0|^2|\sigma| \leq \ell^2 \times 2C_1\ell^2 \leq 2C_1\ell^4$.

(c) Lines 6-7 need to traverse each link key in $\mathcal{LK}$, each pair of star-types at $\sigma, \sigma' \in \Lambda_0$ and respectively each triple $\rho, \rho' \in \sigma, \sigma'$ and star-type in $\Omega(\sigma, \rho)$ and $\Omega(\sigma', \rho')$. So the complexity of this step is bounded by $|\mathcal{LK}||\Lambda_0|^2|\sigma|^2|\Lambda_0|^2 \leq \ell\ell^2(2C_1\ell^2)^2\ell^2 \leq 4C_1^2\ell^9$.

(d) Lines 8→13: For each star-type σ in Λ_i and for each triple $\rho \in \sigma$ such that the set of tail individuals of ρ is empty, we have to check if σ can be matched to an already existing star-type at Λ_{i+1}, otherwise we have to choose a new star-type from $\Sigma_C(\mathcal{O})$ and add it to Λ_{i+1}. So the complexity of this step is bounded by $n|\Lambda_i|\|\sigma\|(|\Lambda_{i+1}| + |\Sigma_C(\mathcal{O})|)$. According to Lemma 16, we have $n \leq 2^{C_1\ell^2}$, $|\Lambda_i| \leq 2^{2C_1\ell^2}$, $\|\sigma\| \leq C_3\ell^5$. In addition, we have obtained from the computing of the complexity of Line 1 that $|\Sigma_C(\mathcal{O})| \leq 2^{C_1\ell^2+1}$. Then $n|\Lambda_i|\|\sigma\|(|\Lambda_{i+1}|+|\Sigma_C(\mathcal{O})|) \leq 2^{C_1\ell^2}2^{2C_1\ell^2}C_3\ell^5(2^{2C_1\ell^2} + 2^{C_1\ell^2+1}) \leq 2^{3C_1\ell^2}C_3\ell^5 + 2^{3C_1\ell^2+1}C_3\ell^5 \leq 2^{C_4\ell^2}$ with C_4 a constant large enough.

(e) Lines 14→16 (Pruning): For each layer prunning checks first if there exists a non blocked star-type with no successor, then it removes it from the current layer and from the successors of its predecessors as well. The complexity of the first step is bounded by $n|\Lambda_i|\|\sigma\|^2|\Lambda_{i+1}| \leq (2^{2C_1\ell^2})^2 \times (C_3\ell^5)^2 \leq 2^{4C_1\ell^2} \times C_3^2\ell^{10}$. The complexity of the second step is bounded by $|\Lambda_{i-1}||\sigma| \leq 2^{2C_1\ell^2} \times C_1\ell^2$.

Pruning checks as well that for every individual in $\mathbf{I}$ there is a star-type in Λ_0 so the complexity of this step is bounded by ℓ^2.

Then the total complexity of pruning is $2^{4C_1\ell^2} \times C_3^2\ell^{10} \times 2^{2C_1\ell^2} \times C_1\ell^2 + \ell^2 \leq 2^{C_5\ell^2}$ with C_5 a constant large enough.

We have proved that the complexity of each step of the algorithm is bounded by $2^{C_i\ell^2}$ for some constant C_i. This implies that the complexity of the algorithm is exponential in the size of $\mathcal{O}$. This completes the proof. □

Theorem 6. *Consistency of $\mathcal{ALC}$+$\mathcal{LK}$ ontology can be decided in* EXPTIME.

Proof of Theorem 6

Lemma 5 proved that if Algorithm 3 returns YES from an ontology in $\mathcal{ALC}$+$\mathcal{LK}$ then this ontology is consistent. Conversely, Lemma 6 proved that if an ontology $\mathcal{O}$ in $\mathcal{ALC}$+$\mathcal{LK}$ is consistent then Algorithm 3 returns YES. Moreover, Lemma 8 provides the complexity of Algorithm 3 which also ensures its termination. These results prove that Algorithm 3 is a decision procedure for the consistency problem in $\mathcal{ALC}$+$\mathcal{LK}$. Moreover, Lemma 8 proved that the complexity of this algorithm is exponential in the size of an input ontology. This completes the proof of Theorem 6. □

It is known that $\mathcal{ALC}$ is EXPTIME-complete [57, 58]. Therefore, EXPTIME is a lower bound complexity for $\mathcal{ALC}$+$\mathcal{LK}$.

5.6 Conclusion

In this chapter, we presented a non-directed compressed tableau algorithm for deciding the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. This algorithm avoids the exponential blow-up of the standard tableau algorithm given in Chapter 4 that results from the application of the non-deterministic rule. It has an EXPTIME complexity. We also prove that this algorithm is sound and complete. As a result, this algorithm proves that reasoning in the description logic $\mathcal{ALC}+\mathcal{LK}$ is EXPTIME.

However, the construction of all possible star-types from the input ontology is immense. Also, the creation of non-nominal star-types must be uniquely done for satisfying the existential assertions. For these reasons, the proposed algorithm is not practical for implementation. In the next chapter, we introduce a worst-case optimal **tableau** for reasoning algorithm for $\mathcal{ALC}+\mathcal{LK}$ based on the algorithm presented in this chapter. A tableau algorithm is directed by the application of completion rules which, in contrast to the non-directed one, avoids unnecessary generation of useless star-types.

Chapter 6

A worst-case optimal EXPTIME tableau algorithm for reasoning in the description logic $\mathcal{ALC}$ extended with link keys and individual equalities

6.1 Introduction

This chapter provides a worst-case optimal tableau algorithm for reasoning in the description logic $\mathcal{ALC}$+$\mathcal{LK}$. Similar to the non-directed algorithm given in the previous chapter, this algorithm is based on compressed tableau algorithm for the description logic $\mathcal{SHOIQ}$ [16] and runs in EXPTIME. It aims at building a compressed tableau for the input ontology. In contrast to the non-directed algorithm, it applies a set of completion rules for accomplishing its goal. The two sources of inefficiency in the previous algorithm is the construction of all possible star-types from the input ontology and checking their validity. This exhaustive construction of star-types may not serve for the construction of the compressed tableau. However, the application of completion rules reduces the unnecessary construction of star-types. The star-types are exclusively created for satisfying concept and role assertions in the input ABox or as a result of rule application.

The remainder of this chapter is organized as follows. Section 6.2 provides the definitions required to understand the compressed tableau structure. Section 6.3 presents the compressed tableau algorithm for the description logic $\mathcal{ALC}$+$\mathcal{LK}$. In Section 6.4, we provide an example of an $\mathcal{ALC}$+$\mathcal{LK}$ ontology and we execute the algorithm on it. Section 6.5 gives and proves the theoretical properties of the algorithm. Section 6.6 provides a preliminary study on extending the compressed tableau algorithm for $\mathcal{ALC}$+$\mathcal{LK}$ to do reasoning in $\mathcal{ALCI}$+$\mathcal{LK}$. We provide our concluding remarks in Section 6.7.

6.2 A Compressed Tableau for the logic $\mathcal{ALC}$+$\mathcal{LK}$

We use the same definitions of subconcepts, triples, layer, matching function, link key satisfaction, and compressed tableau given in Section 5.2 of Chapter 5. However, we substitute the definition of valid star-type by the definitions of saturated and clash-free star-types. A saturated

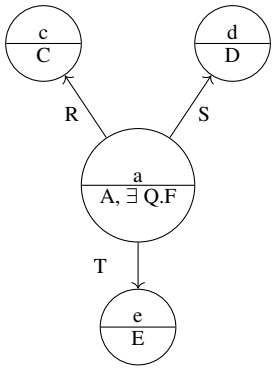
star-type is a star-type where no completion rule is applicable (Definition 24). A clash-free star-type is a star-type that does not contain any contradiction (Definition 25). We also give the definition of merging a pair of star-types that results from the application of equality or link key rule (Definition 26).

Definition 24 (Saturated star-type). *Let $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\sigma \in \Sigma$. The star-type σ is* saturated *if the following hold:*

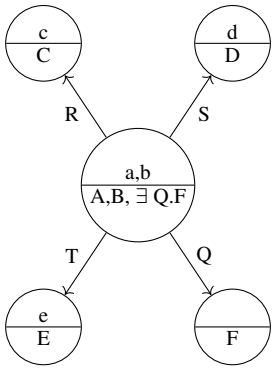
1. *if $C \sqcap D \in \mathsf{core}_{\mathsf{C}}(\sigma)$ then $\{C, D\} \subseteq \mathsf{core}_{\mathsf{C}}(\sigma)$;*
2. *if $C \sqcup D \in \mathsf{core}_{\mathsf{C}}(\sigma)$ then $\{C, D\} \cap \mathsf{core}_{\mathsf{C}}(\sigma) \neq \emptyset$;*
3. *if $\forall R.C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ then, for every triple $\tau \in \sigma$, if $R \in \mathsf{tie}(\tau)$ then $C \in \mathsf{tail}_{\mathsf{C}}(\tau)$;*
4. *if $\exists R.C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ then there exists a triple $\tau \in \sigma$ such that $C \in \mathsf{tail}_{\mathsf{C}}(\tau)$ and $R \in \mathsf{tie}(\tau)$;*
5. *if $C \sqsubseteq D \in \mathcal{T}$ then $\{\mathsf{nnf}(C), D\} \cap \mathsf{core}_{\mathsf{C}}(\sigma) \neq \emptyset$.*

Example 13. *Consider the input ontology $\mathcal{O}$ where:*

- $\mathcal{A} = \{A(a), B(b), a{\approx}b, C(c), R(a, c), D(d), S(b, d), E(e), T(a, e)\}$,
- $\mathcal{T} = \{A \sqsubseteq B\}$,
- $\mathcal{LK} = \emptyset$.



(a) The star-type σ is not saturated.



(b) A saturated star-type σ satisfying the GCI $A \sqsubseteq B \in \mathcal{T}$, the concept $\exists Q.F \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and the equality assertion $a \approx b \in \mathcal{A}$.

Figure 6.1 – The figure on the right hand side is not saturated star-type while figure on the left hand side is saturated.

Definition 25 (Clash-free star-type). *Let $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\sigma \in \Sigma$. σ is* clash-free *if the following hold:*

1. *if A is a concept name in $\mathcal{O}$ then $\{A, {\sim}A\} \not\subseteq \mathsf{core}_{\mathsf{C}}(\sigma)$ and, for every triple $\tau \in \sigma$, $\{A, {\sim}A\} \not\subseteq \mathsf{tail}_{\mathsf{C}}(\tau)$;*

2. *if* $a \not\approx b \in \mathcal{A}$ *then* $\{a, b\} \not\subseteq \mathsf{core}_{\mathsf{I}}(\sigma)$ *and, for every triple* $\tau \in \sigma$, $\{a, b\} \not\subseteq \mathsf{tail}_{\mathsf{I}}(\tau)$.

The merging operation as described below allows to merge only nominal triples.

Definition 26 (**Merging of star-types**). *Let* $\mathcal{O}$ *be an* $\mathcal{ALC}$+$\mathcal{LK}$ *ontology and* Σ *be a set of star-types over* $\mathcal{O}$. *For* $\sigma_1, \sigma_2 \in \Sigma$, *a set* $\sigma_1 \oplus \sigma_2 = \{\omega_1, \omega_2\}$ *resulting from merging* σ_1 *into* σ_2 *is defined as follows:*

1. $\mathsf{core}(\omega_i) = \mathsf{core}(\sigma_1) \cup \mathsf{core}(\sigma_2)$ *for all* $1 \leq i \leq 2$;
2. $\rho \in \omega_i$ *iff*
   (a) *either there is some* $\rho' \in \sigma_i$ *with* $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$ *and* $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$;
   (b) *or there is some* $\rho' \in \sigma_j$ *with* $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$, $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$, $1 \leq i \neq j \leq 2$ *such that there are* $R(a, b) \in \mathcal{A}$, $a \in \mathsf{core}_{\mathsf{I}}(\sigma_j)$, $R \in \mathsf{tie}(\rho')$, $b \in \mathsf{tail}_{\mathsf{I}}(\rho')$ *and there is no* $\rho'' \in \sigma_i$ *with* $a \in \mathsf{head}_{\mathsf{I}}(\rho'')$, $R \in \mathsf{tie}(\rho'')$, $b \in \mathsf{tail}_{\mathsf{I}}(\rho'')$.

We are now ready to present the compressed tableau algorithm for $\mathcal{ALC}$+$\mathcal{LK}$.

6.3 Compressed Tableau Algorithm

We present a goal-directed algorithm based on compressed tableau for checking consistency of an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. This algorithm uses a set of *completion* rules to build a structure, called *pre-compressed tableau*, for approximating a compressed tableau, i.e. such a pre-compressed tableau is expanded by applying completion rules until a compressed tableau included in the pre-compressed tableau is detected. Each completion rule corresponds to a logical constructor defined for $\mathcal{ALC}$+$\mathcal{LK}$. For the sake of readability, the set of completion rules is presented as two subsets of rules in Figures 6.2 and 6.3. The first set of rules (Figure 6.2) aims to satisfy $\mathcal{ALC}$ concepts occurring in the core of a star-type while each rule in the second set (Figure 6.3) concerns a set of star-types involved in a link keys or an equality assertion.

As mentioned previously, the compressed tableau algorithm initialises a pre-compressed tableau and makes it evolve until a compressed tableau inside the pre-compressed tableau is detected. The main role of the pre-compressed tableau is to maintain the neighborhood relationship between star-types when an application of a completion rule leads to add new star-types. Note that the compressed tableau algorithm is monotonic, i.e. it adds to the current pre-compressed tableau some new pieces of the structure such as new star-types or extension of the matching function, and it never removes anything from the pre-compressed tableau. When adding new star-types to the current pre-compressed tableau, the algorithm performs necessary changes in such a way that the neighborhood relationship between star-types is re-established over the new pre-compressed tableau. The following definition characterises a pre-compressed tableau.

Definition 27 (**Pre-compressed tableau**). *Let* $\mathcal{O}$ *be an* $\mathcal{ALC}$+$\mathcal{LK}$ *ontology with* $\mathbf{I}$ *a set of individuals. Let* $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^{n}$ *be a finite sequence of layers. with a matching function over* $\mathbf{\Lambda}$. *A pair* $\langle \mathbf{\Lambda}, \Omega \rangle$ *is called a pre-compressed tableau of* $\mathcal{O}$ *if the following conditions are satisfied:*

1. *For every* $a \in \mathbf{I}$, *there exists a star-type* $\sigma \in \Lambda_0$ *such that* $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$.

Rule $\rightarrow_{\sqcap}$
Condition: If $C_1 \sqcap C_2 \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and $\{C_1, C_2\} \not\subseteq \mathsf{core}_{\mathsf{C}}(\sigma)$ where $\sigma \in \Lambda_i$.
Action 1: create a copy σ' of σ,
Action 2: $\mathsf{core}_{\mathsf{C}}(\sigma') \leftarrow \mathsf{core}_{\mathsf{C}}(\sigma) \cup \{C_1, C_2\}$,
Action 3: call $\mathsf{matchCore}(\sigma, \sigma', i)$

Rule $\rightarrow_{\sqcup}$
Condition: If $C_1 \sqcup C_2 \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and $\{C_1, C_2\} \cap \mathsf{core}_{\mathsf{C}}(\sigma) = \emptyset$ where $\sigma \in \Lambda_i$.
Action 1: create two copies σ' and σ'' of σ,
Action 2: $\mathsf{core}_{\mathsf{C}}(\sigma') \leftarrow \mathsf{core}_{\mathsf{C}}(\sigma) \cup \{C_1\}$ and $\mathsf{core}_{\mathsf{C}}(\sigma'') \leftarrow \mathsf{core}_{\mathsf{C}}(\sigma) \cup \{C_2\}$.
Action 3: call $\mathsf{matchCore}(\sigma, \sigma', i)$ and $\mathsf{matchCore}(\sigma, \sigma'', i)$

Rule $\rightarrow_{\sqsubseteq}$
Condition: If $\mathcal{T}$ contains $C \sqsubseteq D$ and $\mathsf{nnf}(\sim C \sqcup D) \notin \mathsf{core}_{\mathsf{C}}(\sigma)$. where $\sigma \in \Lambda_i$
Action 1: create a copy σ' of σ,
Action 2: $\mathsf{core}_{\mathsf{C}}(\sigma') \leftarrow \mathsf{core}_{\mathsf{C}}(\sigma) \cup \{\mathsf{nnf}(\sim C \sqcup D)\}$,
Action 3: call $\mathsf{matchCore}(\sigma, \sigma', i)$

Rule $\rightarrow_{\forall}$
Condition: If $\forall R.C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and there exists a triple $\rho \in \sigma$ such that $R \in \mathsf{tie}(\rho)$ and $C \notin \mathsf{tail}_{\mathsf{C}}(\rho)$ where $\sigma \in \Lambda_i$.
Action 1: create a copy σ' of σ
Action 2: create a copy ρ' of ρ, add C to $\mathsf{tail}_{\mathsf{C}}(\rho')$, and replace ρ in σ' with ρ',
Action 3: call $\mathsf{matchTriple}(\sigma, \rho, \sigma', \rho', i)$

Rule $\rightarrow_{\exists}$
Condition: If $\exists R.C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and there exists no triple $\rho \in \sigma$ such that $R \in \mathsf{tie}(\rho)$ and $C \in \mathsf{tail}(\rho)$ and σ is not blocked where $\sigma \in \Lambda_i$.
Action 1: create a star-type σ', a copy of σ,
Action 2: create a triple $\rho = \langle \mathsf{core}(\sigma), \{R\}, \langle \emptyset, \{C\} \rangle \rangle$, and add ρ to σ',
Action 3: call $\mathsf{matchTriple}(\sigma, \mathsf{null}, \sigma', \rho, i)$

Figure 6.2 – Completion rules applied on a star-type σ.

Rule $\rightarrow_{\mathbf{chLK}_1}$
Condition 1: σ_1 and σ_2 weakly satisfy the condition of $(\{\langle P_k, Q_k\rangle\}_{k=1}^{n}$ linkkey $\langle C, D\rangle)$ where $\sigma_1, \sigma_2 \in \Lambda_0$,
Condition 2: $\mathsf{core}_{\mathsf{C}}(\sigma_1) \cap \{C, {\sim}C\} = \emptyset$.
Action 1: create two copies ${\sigma'}_1$ and σ''_1 of σ_1,
Action 2: add C to $\mathsf{core}_{\mathsf{C}}(\sigma'_1)$ and ${\sim}C$ to $\mathsf{core}_{\mathsf{C}}(\sigma''_1)$,
Action 3: call $\mathsf{matchCore}(\sigma_1, \sigma'_1, 0)$ and $\mathsf{matchCore}(\sigma_1, \sigma''_1, 0)$.

Rule $\rightarrow_{\mathbf{chLK}_2}$
Condition 1: σ_1 and σ_2 weakly satisfy the condition of $(\{\langle P_k, Q_k\rangle\}_{k=1}^{n}$ linkkey $\langle C, D\rangle)$ where $\sigma_1, \sigma_2 \in \Lambda_0$,
Condition 2: $\mathsf{core}_{\mathsf{C}}(\sigma_2) \cap \{D, {\sim}D\} = \emptyset$.
Action 1: create two copies σ'_2 and σ''_2 of σ_2,
Action 2: add D to $\mathsf{core}_{\mathsf{C}}(\sigma'_2)$ and ${\sim}D$ to $\mathsf{core}_{\mathsf{C}}(\sigma''_2)$,
Action 3: call $\mathsf{matchCore}(\sigma_1, \sigma'_2, 0)$ and $\mathsf{matchCore}(\sigma_2, \sigma''_2, 0)$.

Rule $\rightarrow_{\mathbf{LK}}$
Condition 1: σ_1 and σ_2 satisfy the condition of a link key λ where $\sigma_1, \sigma_2 \in \Lambda$. *Condition 2*: There are no star-types $\sigma_0, \sigma'_0 \in \Lambda_0$ such that:
• $\mathsf{core}(\sigma_1) \subseteq \mathsf{core}(\sigma_0)$, $\mathsf{core}(\sigma_2) \subseteq \mathsf{core}(\sigma_0)$, and each R-successor of σ_1 is an R-successor of σ_0,
• $\mathsf{core}(\sigma_1) \subseteq \mathsf{core}(\sigma'_0)$, $\mathsf{core}(\sigma_2) \subseteq \mathsf{core}(\sigma'_0)$, and each R-successor of σ_2 is an R-successor of σ'_0.

Action 1: create $\sigma_1 \oplus \sigma_2$,
Action 2: call $\mathsf{matchMerge}(\sigma_1, \sigma_2, \sigma_1 \oplus \sigma_2)$.

Rule $\rightarrow_{\approx}$
Condition 1: $\mathcal{A} \cap \{a \approx b : a \in \mathsf{core}_{\mathsf{I}}(\sigma_i) \wedge b \in \mathsf{core}_{\mathsf{I}}(\sigma_j) \wedge 1 \leq i < j \leq 2\} \neq \emptyset$. *Condition 2*: There is no star-types $\sigma_0, \sigma'_0 \in \Lambda_0$ such that:
• $\mathsf{core}(\sigma_1) \subseteq \mathsf{core}(\sigma_0)$, $\mathsf{core}(\sigma_2) \subseteq \mathsf{core}(\sigma_0)$, and each R-successor of σ_1 is an R-successor of σ_0,
• $\mathsf{core}(\sigma_1) \subseteq \mathsf{core}(\sigma'_0)$, $\mathsf{core}(\sigma_2) \subseteq \mathsf{core}(\sigma'_0)$, and each R-successor of σ_2 is an R-successor of σ'_0.

Action 1: create $\sigma_1 \oplus \sigma_2$,
Action 2: call $\mathsf{matchMerge}(\sigma_1, \sigma_2, \sigma_1 \oplus \sigma_2)$.

Figure 6.3 – Link key and equality rules applied to star-types in Λ_0.

2. *For every* $a \in \mathbf{I}$ *and every star-type* $\sigma \in \Lambda_0$ *such that* $a \in \mathsf{core}_\mathsf{I}(\sigma)$,

   (a) *if* $C(a) \in \mathcal{A}$ *then* $C \in \mathsf{core}_\mathsf{C}(\sigma)$,

   (b) *if* $R(a,b) \in \mathcal{A}$ *then there is a triple* $\tau \in \sigma$ *such that* $R \in \mathsf{tie}(\tau)$ *and* $b \in \mathsf{tail}_\mathsf{I}(\tau)$.

3. *For every* $R(a,b) \in \mathcal{A}$, *there exist* $\sigma, \sigma' \in \Lambda_0$ *such that* $a \in \mathsf{core}_\mathsf{I}(\sigma)$, $b \in \mathsf{core}_\mathsf{I}(\sigma')$, *and* $\sigma' \in \Omega(\sigma, \tau)$ *where* $\tau \in \sigma$, $R \in \mathsf{tie}(\tau)$ *and* $b \in \mathsf{tail}_\mathsf{I}(\tau)$.

4. *For every* $\sigma \in \mathbf{\Lambda}$,

   (a) *if* $\sigma \in \Lambda_0$ *then* σ *is nominal,*

   (b) *if* $\sigma \in \Lambda_k$ *and* $k \geq 1$ *then* σ *is non-nominal and clash-free,*

   (c) *if* $\sigma \in \Lambda_n$ *and* $n \geq 1$ *then* σ *is either blocked or dummy.*

To initialise a pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$, the compressed tableau algorithm calls another algorithm called init. Init (Algorithm 4) builds a set of nominal star-types and adds it to the first layer Λ_0 of $\mathbf{\Lambda}$, also it connects star-types in Λ_0 according to the definition of the matching function Ω. Then the algorithm calls $\mathcal{ALC}$ and $\mathcal{LK}$ completion rules in Figures 6.2 and 6.3 which in turn call a set of matching algorithms matchCore (Algorithm 5), matchTriple (Algorithm 6) and matchMerge (Algorithm 7) until there is a compressed built for the input ontology detected by check (Algorithm 8). The main role of the matching algorithms is to check the clash-freeness of a new star-type obtained by applying a completion rule, before adding it to the pre-compressed tableau and to ensure that the neighbourhood relationship in the compressed tableau is maintained after its addition. Finally, check (Algorithm 8) algorithm examines if there is a compressed tableau contained inside the input pre-compressed tableau. In what follows, we explain each algorithm mentioned above and give its pseudocode.

Init uses a labelling function L that associates to each individual a in the input set of individuals $\mathbf{I}$ a set of concepts $L(a)$ such that $C \in L(a)$ for every $C(a) \in \mathcal{A}$.

Lemma 9. *Let* $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ *be an* $\mathcal{ALC} + \mathcal{LK}$ *ontology. Let* $\mathbf{I}$ *be the set of individuals in* $\mathcal{O}$ *and* L *be the labelling function defined over* $\mathbf{I}$.

1. *Algorithm 4 initializes* Λ_0 *and* Ω *such that:*

   (a) *for each individual* $a \in \mathbf{I}$ *there is a unique star-type* $\sigma \in \Lambda_0$ *such that* $\mathsf{core}_\mathsf{I}(\sigma)=\{a\}$ *and* $\mathsf{core}_\mathsf{C}(\sigma)=L(a)$.

   (b) *for each* $R(a,b) \in \mathcal{A}$ *there exists a unique triple* $\rho \in \sigma$, $\sigma \in \Lambda_0$ *and* $a \in \mathsf{core}_\mathsf{I}(\sigma)$ *with* $R \in \mathsf{tie}(\rho)$, $\mathsf{tail}_\mathsf{I}(\rho) = \{b\}$ *and* $\mathsf{tail}_\mathsf{C}(\rho)=L(b)$.

   (c) *for each* $R(a,b) \in \mathcal{A}$ *if* $a \in \mathsf{core}_\mathsf{I}(\sigma)$ *and* $b \in \mathsf{core}_\mathsf{I}(\omega)$ *then* $\omega \in \Omega(\sigma, \rho)$, ρ *is a triple of* σ *where* $R \in \mathsf{tie}(\rho)$ *and* $b \in \mathsf{tail}_\mathsf{I}(\rho)$.

2. *Algorithm 4 runs in polynomial time in the size of* $\mathcal{O}$.

Proof. 1. (a) Due to the condition of Line 8, for each $a \in \mathbf{I}$ there exists a star-type σ such that $a \in \mathsf{core}_\mathsf{I}(\sigma)$. σ is exclusively built by Line 4 or Line 12 . Hence, σ is unique. Then Item 1a of Lemma 9 is proved.

Algorithm 4: init algorithm

Input: An ABox $\mathcal{A}$, a set $\mathbf{I}$ of individuals and a labelling function L over $\mathbf{I}$.
Output: A set of nominal star-types Λ_0 and a matching function Ω over Λ_0.

1 $\Lambda_0 \leftarrow \emptyset$;
2 **foreach** *individual* $a \in \mathbf{I}$ **do**
3 **if** *there exists no* $R(a, b)$ *in* $\mathcal{A}$ **then**
4 $\sigma \leftarrow \{\langle\langle\{a\}, L(a)\rangle, \emptyset, \langle\emptyset, \emptyset\rangle\rangle\}$;
5 **end**
6 **else**
7 **foreach** $R(a, b) \in \mathcal{A}$ **do**
8 **if** *there exists a star-type* $\sigma \in \Lambda_0$ *such that* $a \in \mathsf{core}_\mathsf{I}(\sigma)$ **then**
9 $\sigma \leftarrow \sigma \cup \{\langle\langle\{a\}, L(a)\rangle, \{R\}, \langle\{b\}, L(b)\}$;
10 **end**
11 **else**
12 $\sigma \leftarrow \{\langle\langle\{a\}, L(a)\rangle, \{R\}, \langle\{b\}, L(b)\rangle\rangle\}$;
13 **end**
14 **end**
15 **end**
16 **end**
17 Initialise $\Omega \leftarrow \emptyset$;
18 **foreach** *role assertion* $R(a, b) \in \mathcal{A}$ *and* $\sigma, \sigma' \in \Lambda_0$ **do**
19 **if** $a \in \mathsf{core}_\mathsf{I}(\sigma)$ *and* $b \in \mathsf{core}_\mathsf{I}(\sigma')$ **then**
20 add σ' to $\Omega(\sigma, \rho)$ where ρ is the only triple in σ such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}(\rho)$;
21 **end**
22 **end**
23 **return** $\langle\Lambda_0, \Omega\rangle$;

(b) Let σ be a star-type in Λ_0 with $a \in \mathsf{core}_\mathsf{I}(\sigma)$. For each role assertion of the form $R(a, b) \in \mathcal{A}$ the algorithm adds only one triple ρ to σ with $\mathsf{tie}(\rho) = \{R\}$ and $\mathsf{tail}_\mathsf{I}(\rho) = \langle\{b\}, L(b)\rangle$ as shown in Line 10 or Line 12. Item 1b of Lemma 9 is proved.

(c) Let $R(a, b)$ be a role assertion in $\mathcal{A}$ and σ, ω be two star-types such that $a \in \mathsf{core}_\mathsf{I}(\sigma)$ and $b \in \mathsf{core}_\mathsf{I}(\omega)$, due to Item 1b there exists a triple $\rho \in \sigma$ such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}_\mathsf{I}(\rho)$. According to Line 21, ω is added to $\Omega(\sigma, \rho)$. Item 1c of Lemma 9 is proved.

2. We now analyze the complexity of Algorithm 4. Let ℓ be the size of $\mathcal{O}$, then the size of $\mathcal{A}$ is bounded by ℓ.

(a) Lines 2, 3 and 7 traverses the components of $\mathcal{O}$. Then its complexity is bounded by ℓ.

(b) Line 8 traverses each star-type σ in Λ_0 and each individual in $\mathsf{core}_\mathsf{I}(\sigma)$. Since Λ_0 contains a unique star-type for each individual in $\mathcal{I}$. Then complexity of Line 8 is bounded by ℓ^2.

(c) Line 18 traverses each role assertion in $\mathcal{A}$ and each pair of star-types $\sigma, \sigma' \in \Lambda_0$. It follows that the complexity of this Line is bounded by ℓ^3.

So the complexity Algorithm 4 is bounded by $\ell \times (\ell + \ell^3) + \ell^3 \times \ell^2$. Hence, Algorithm 4 runs in polynomial time in the size of $\mathcal{O}$.

□

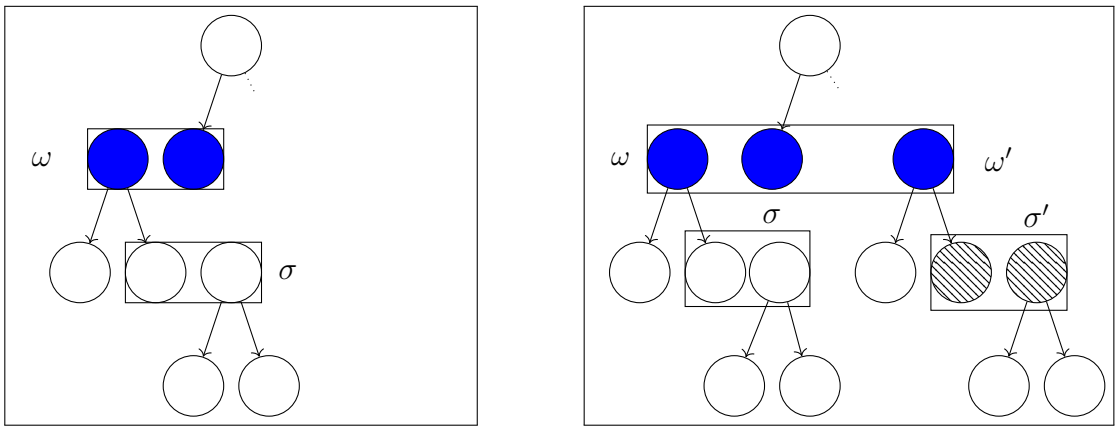
• matchCore is called for adding new star-types generated by rules that change the core of a star-type and updating the matching function Ω to ensure that the added star-type σ' has a neighboor. This can lead to add new star-types. This situation is illustrated in Figure 6.4. When a rule changes the core of σ (in the left-hand side) and transforms it to σ' (in the right-hand side), matchCore should transform a predecessor ω of σ into a predecessor ω' such that ω' is a predecessor of σ'.

Algorithm 5: matchCore algorithm

Input: A pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$ and two star-types σ, σ' such that $\sigma \in \Lambda_i$ and σ' is a copy of σ with a modified core.

Output: Λ_i contains σ', and the updated $\langle \mathbf{\Lambda}, \Omega \rangle$ remains a pre-compressed tableau.

if σ' *is clash-free* **then**
  Add σ' to Λ_i;
  foreach $\omega \in \Lambda_j$ *and* $\rho \in \omega$ *with* $j{=}i{-}1$ *or* $j{=}i{=}0$ *such that* $\sigma \in \Omega(\omega, \rho)$ **do**
    Create a copy ω' of ω and a copy ρ' of ρ;
    Set $\mathsf{tail}(\rho') \leftarrow \mathsf{core}(\sigma')$ and replace ρ by ρ' in ω' ;
    foreach $\psi \in \Lambda_h$ *and* $\nu \in \psi$ *with* $h{=}j{-}1$ *or* $h{=}j{=}i{=}0$ *such that* $\omega \in \Omega(\psi, \nu)$ **do**
      Add ω' to $\Omega(\psi, \nu)$;
    end
    foreach $\nu \in \omega'$ *with* $\nu{\neq}\rho'$ **do**
      Set $\Omega(\omega', \nu) \leftarrow \Omega(\omega, \nu)$;
    end
    Add ω' to Λ_j and σ' to $\Omega(\omega', \rho')$;
  end
  foreach $\rho \in \sigma$ *and* $\psi \in \Omega(\sigma, \rho)$ **do**
    Add ψ to $\Omega(\sigma', \rho')$ where $\mathsf{tie}(\rho'){=}\mathsf{tie}(\rho)$ and $\mathsf{tail}(\rho'){=}\mathsf{tail}(\rho)$;
  end
end



(a) A part of the compressed tableau before the application of Algorithm 5.

(b) A part of the compressed tableau after the application of Algorithm 5.

Figure 6.4 – Application of matchCore algorithm

Lemma 10. *Let $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$ be a pre-compressed tableau. Assume that Algorithm 5 takes two star-types σ and σ' as input with $\sigma \in \Lambda_i$.*

1. *If σ' is clash-free, then Algorithm 5 adds σ' to Λ_i. In this case, each predecessor ω of σ is changed to ω' that is added to $\mathbf{\Lambda}$ such that ω' is a predecessor of σ', and each predecessor of ω is a predecessor of ω'.*
2. *The updated $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau.*
3. *Algorithm 5 runs in a polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.*

Proof. 1. If σ' is clash-free, Line 2 of Algorithm 5 adds σ' to $\Lambda_i, i \geq 0$.

   (a) Let ω be a ρ-predecessor of σ. Line 4 creates a copy ω' of ω and a copy ρ' of ρ. Line 5 assigns $\mathsf{tail}(\rho')$=$\mathsf{core}(\sigma')$ and replaces $\rho \in \omega'$ by ρ'. Line 9 adds ω' to Λ_j and σ' to $\Omega(\omega', \rho')$ for j=$i-1$ if $i > 0$, or j=i=0. Hence, ω' is a ρ'-predecessor of σ'.

   (b) Let ψ be a ν-predecessor of ω with $\psi \in \Lambda_h$ for h=$j-1$ if $i > 0$, or h=j=i=0. The loop between Lines 6 and 8 adds ω' to $\Omega(\psi, \nu)$. Hence, ψ is a predecessor of ω'.

   Therefore, Property 1. of the lemma is proved.

2. Let $\langle \mathbf{\Lambda}', \Omega' \rangle$ be the structure obtained from an input $\langle \mathbf{\Lambda}, \Omega \rangle$ by the algorithm where $\mathbf{\Lambda}'$ is extended from $\mathbf{\Lambda}$ with new star-types, and Ω' is extended from Ω for the new star-types. First, we show that Ω' is well defined over $\mathbf{\Lambda}'$. Let σ_0 be a star-type in $\mathbf{\Lambda}$. If σ_0 and all its neighbors are not changed by the algorithm, then Ω' is well defined for σ_0 over $\mathbf{\Lambda}'$. Assume that σ_0 or one of its neighbors is changed by the algorithm. Since Ω concerns neighbors, it suffices to consider the following cases:

   (a) σ_0=σ and σ_0 is changed to σ_0'=σ'. By Property 1. of the lemma, σ' has a predecessor. Let ψ be a τ-successor of σ. By Line 15, we have $\Omega'(\sigma', \rho') \neq \emptyset$ for all $\rho' \in \sigma'$.

   (b) σ_0 is a ρ-predecessor of σ, and σ_0 is changed to σ_0'. By Property 1. of the lemma, σ_0' is a predecessor of σ'. By Line 7, each predecessor of σ_0 is a predecessor of σ_0'. Moreover, by Line 10, each ν- successor of σ_0 with $\nu \neq \rho$ is a ν-successor of σ_0'. Hence, $\Omega'(\sigma_0', \nu) \neq \emptyset$ for all $\nu \in \sigma_0'$ with $\nu \neq \rho$.

   (c) σ_0 is a ρ-successor of σ. Algorithm 5 never changes σ_0 and its successors. Thus, if $\Omega(\sigma_0, \rho) \neq \emptyset$ then $\Omega'(\sigma_0, \rho) \neq \emptyset$ for all $\rho \in \sigma_0$.

   We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1, 2, 3 and 4 of Definition 27.

   (a) Since the algorithm never removes anything from $\langle \mathbf{\Lambda}, \Omega \rangle$, it holds that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1 and 3 of Definition 27.

   (b) We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 2 of Definition 27.

      i. For Property 2a, if a nominal star-type σ_0 is changed to σ_0' where $a \in \mathsf{core}_{\mathsf{I}}(\sigma_0')$ and $C(a) \in \mathcal{A}$ then $C \in \mathsf{core}_{\mathsf{C}}(\sigma_0')$ since $C \in \mathsf{core}_{\mathsf{C}}(\sigma_0)$ as $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau and the algorithm never removes any concept from $\mathsf{core}_{\mathsf{C}}(\sigma_0')$.

ii. For Property 2b, the proof is similar to the proof of Property 2a.

(c) We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 4 of Definition 27.

i. For Property 4a, if a star-type σ_0 is changed to $\sigma'_0 \in \Lambda'_0$ then by Item 1 of Lemma 10 we have $\sigma_0 \in \Lambda_0$. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau, we get that σ_0 is nominal. It holds that $\mathsf{core}_{\mathsf{I}}(\sigma_0) = \mathsf{core}_{\mathsf{I}}(\sigma'_0) \neq \emptyset$ and thus σ'_0 is nominal. This implies that Property 4a is satisfied for $\langle \mathbf{\Lambda}', \Omega' \rangle$.

ii. For Property 4b, if a star-type σ_0 is changed to $\sigma'_0 \in \Lambda'_k, k \geq 1$ then by Item 1 of Lemma 10 we have $\sigma_0 \in \Lambda_k, k \geq 1$. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ as is a pre-compressed tableau, we get that σ_0 is non-nominal. It holds that $\mathsf{core}_{\mathsf{I}}(\sigma_0) = \mathsf{core}_{\mathsf{I}}(\sigma'_0) = \emptyset$, we get σ'_0 is non-nominal. Besides, by Item 1 of Lemma 11, σ'_0 is clash-free. This implies that Property 4b is satisfied for $\langle \mathbf{\Lambda}', \Omega' \rangle$.

iii. For Property 4c, if a dummy star-type σ_0 is changed to σ'_0 then σ'_0 is also dummy since Algorithm 5 never adds a new triple or changes the tie and tail of a dummy triple. This implies that Property 4c. of Definition 27 is also satisfied for $\langle \mathbf{\Lambda}', \Omega' \rangle$.

We have proved that Ω' is well defined over $\mathbf{\Lambda}'$ and all properties of Definition 27 are verified. Therefore, $\langle \mathbf{\Lambda}', \Omega' \rangle$ is a pre-compressed tableau.

3. We now analyze the complexity of Algorithm 5. Since the non loop lines in the algorithm need to traverse only components of a star-type or equality assertions, the complexity of these lines is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. We analyze the complexity of the loops.

(a) The loop between Lines 3 and 13 traverses all star-types of a layer Λ_j and its neighbors. Hence, the number of iterations of this loop is bounded by $|\sigma| \times |\Lambda|^2$ where $|\sigma|$ is the greatest number of triples of a star-type, and $|\Lambda|$ is the greatest number of star-types of a layer.

(b) The two loops between Lines 6 and 8, and Lines 9 and 11 traverses all star-types of a layer Λ_h and its neighbors, or all triples of a star-type. Hence, the number of iterations of these loops is also bounded by $|\sigma| \times |\Lambda|^2$.

(c) Since, the two loops between Lines 6 and 8, and Lines 9 and 11 are nested in the loop between Lines 3 and 13, the total number of iterations of this loop is bounded by $2|\sigma|^2 \times |\Lambda|^4$.

(d) Finally, the loop between Lines 14 and 16 visits all neighbors of a star-type. Hence, the number of iterations of this loop is bounded by $|\sigma| \times |\Lambda|^2$.

As a result, the complexity of the loops in Algorithm 5 is bounded by $3|\sigma|^2 \times |\Lambda|^4$. This means that the complexity of Algorithm 5 is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

□

• matchTriple is called by rules for adding new star-types generated by rules that change the tail of a triple of a star-type, and updating the matching function Ω to ensure that the added star-type σ' has a neighbour. This is illustrated in Figures 6.5 and 6.6. Figure 6.5 shows the behaviour of the algorithm when a star-type σ' (the transformation of an input star-type σ) of triple ρ' is added to $\mathbf{\Lambda}$ and $\rho' \notin \sigma$. Figure 6.6 shows when a rule changes the tail of a triple ρ in a

star-type σ (in the left-hand side), σ is transformed to σ' where ρ is replaced by the transformed triple ρ' (in the right-hand side), matchTriple should transform a ρ-successor ω of σ into ω', a ρ'-successor of σ'.

Algorithm 6: matchTriple algorithm

Input: A pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$, two star-types σ, σ' with $\sigma \in \Lambda_i$ and two triples $\rho \in \sigma$, $\rho' \in \sigma'$ where σ' is copy of σ such that $\rho \in \sigma$ is replaced with $\rho' \in \sigma'$ and ρ' is a copy of ρ with a modified tail.

Output: Λ_i contains σ', and the updated $\langle \mathbf{\Lambda}, \Omega \rangle$ remains a pre-compressed tableau.

if σ' *is clash-free* **then**
  Add σ' to Λ_i;
  foreach *star-type* $\tau \in \Lambda_j$, $j{=}i{-}1$ *or* $j{=}i{=}0$ *and triple* $\nu \in \tau$ *such that* $\sigma \in \Omega(\tau, \nu)$ **do**
    add σ' to $\Omega(\tau, \nu)$;
  end
  if $\rho = \mathsf{null}$ **then**
    if $i = n$ *and there is no individual in* $\mathsf{tail}_{\mathsf{I}}(\rho')$ **then**
      Add an empty layer Λ_{n+1} to $\mathbf{\Lambda}$;
    end
    Create a star-type ω such that $\mathsf{core}(\omega) = \mathsf{tail}(\rho')$, and $\mathsf{tie}(\rho'') = \emptyset$, $\mathsf{tail}(\rho'') = \emptyset$ for all $\rho'' \in \omega$;
    Add ω to Λ_{i+1} and $\Omega(\sigma', \rho')$;
  end
  else
    foreach $\omega \in \Omega(\sigma, \rho)$ **do**
      Create a copy ω' of ω and set $\mathsf{core}(\omega') \leftarrow \mathsf{tail}(\rho')$;
      Add ω' to $\Omega(\sigma', \rho')$ and Λ_j where $j{=}i{+}1$ or $j{=}i{=}0$;
      foreach $\nu \in \omega'$ **do**
        Set $\Omega(\omega', \nu) \leftarrow \Omega(\omega, \nu')$ where $\mathsf{tie}(\nu) = \mathsf{tie}(\nu')$ and $\mathsf{tail}(\nu) = \mathsf{tail}(\nu')$;
      end
      foreach $\nu \in \sigma'$ *such that* $\nu \neq \rho'$ **do**
        Set $\Omega(\sigma', \nu) \leftarrow \Omega(\sigma, \nu)$;
      end
    end
  end
end

In the following figure, the pre-compressed tableau on the left-hand side figure shows three star-types σ, ω a successor σ and a successor of ω. Algorithm 6 changes as shown in the right-hand side of the figure ω to ω' and adds ω' to $\mathbf{\Lambda}$. Besides, ω' is now a successor of σ' and the successor of ω is now a successor of ω'.

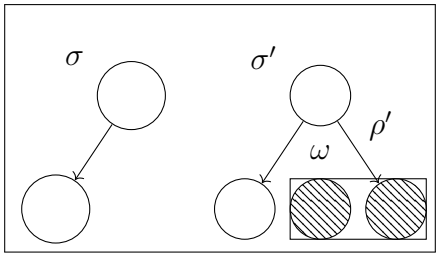


Figure 6.5 – When there is a triple is $\rho' \in \sigma'$ and $\rho' \notin \sigma$ the algorithm creates a ρ'-successor of σ' and adds it to $\mathbf{\Lambda}$.

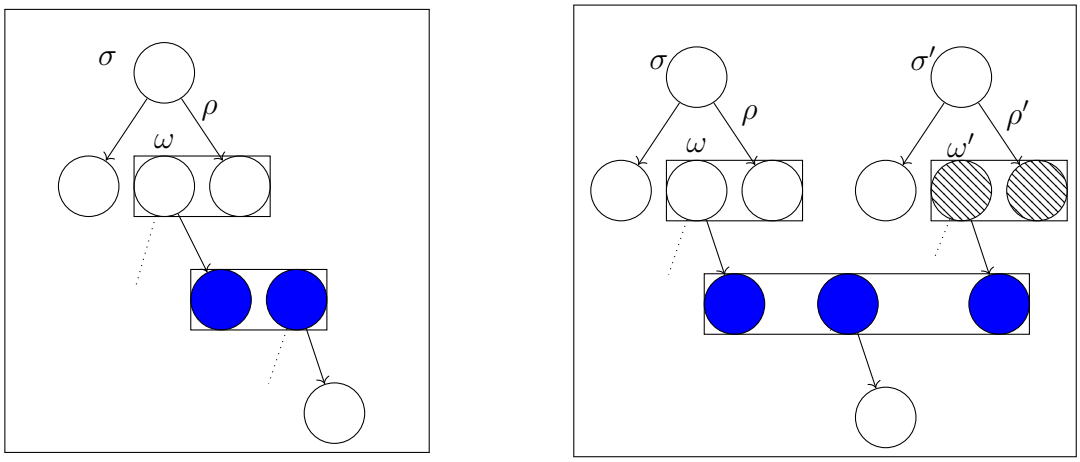


Figure 6.6 – A part of the pre-compressed tableau before and after applying Algorithm 6.

Lemma 11. *Let $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$ be a pre-compressed tableau. Assume that Algorithm 6 takes two star-types σ and σ' and two triples $\rho \in \sigma$, $\rho' \in \sigma'$ as input with $\sigma \in \Lambda_i$.*

1. *If σ' is clash-free, then Algorithm 6 adds σ' to Λ_i.*
   (a) *If $\rho' \notin \sigma$, then Algorithm 6 creates a star-type ω' and adds it to Λ_{i+1} such that ω' is an ρ'-successor of σ'.*
   (b) *Otherwise, each ρ-successor ω of σ is changed to ω' that is added to $\mathbf{\Lambda}$ such that ω' is a ρ'-successor of σ', and each R-successor of ω is an R-successor of ω'.*
2. *The updated $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau.*
3. *Algorithm 6 runs in a polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.*

Proof. 1. If σ' is clash-free, Line 2 of Algorithm 6 adds σ' to $\Lambda_i, i \geq 0$.

(a) If $\rho' \notin \sigma$ this means that the Algorithm performs Line 7 to Line 12. According to Line 7 if $i{=}n$ and $\mathsf{tail}_{\mathsf{I}}(\rho') \neq \emptyset$ the algorithm adds a new layer Λ_{i+1} and adds it to $\mathbf{\Lambda}$. After, Lines 10 and 11 of the algorithm creates a star-type ω' such that $\mathsf{core}(\omega'){=}\mathsf{tail}(\rho')$ and adds ω' to Λ_{i+1} and $\Omega(\sigma', \rho')$. Hence, ω' is a ρ'-successor of σ'.

(b) Otherwise, let ω be a ρ-successor of σ. Line 15 of the algorithm creates a copy ω' of ω with $\mathsf{core}(\omega'){=}\mathsf{tail}(\rho')$ and Line 16 adds it to $\Lambda_j, j{=}0$ or $j{=}i{=}0$ and to $\Omega(\sigma', \rho')$.

Hence, ω' is a ρ'-successor of σ. Since ω' is a copy of ω of modified core we have for every triple $\nu' \in \omega$ there exists a triple $\nu \in \omega'$ with $\mathsf{tie}(\nu)$=$\mathsf{tie}(\nu')$ and $\mathsf{tail}(\nu)$=$\mathsf{tail}(\nu')$. The loop between Lines 20 to 22 in the algorithm assigns $\Omega(\omega, \nu')$ to $\Omega(\omega', \nu)$. Hence, each R-successor of ω is now an R-successor of ω'.

Therefore, Property 1 of the lemma is proved.

2. Let $\langle \mathbf{\Lambda}', \Omega' \rangle$ be the structure obtained from an input $\langle \mathbf{\Lambda}, \Omega \rangle$ by the algorithm where $\mathbf{\Lambda}'$ is extended from $\mathbf{\Lambda}$ with new star-types and Ω' is extended from Ω for the new star-types. First, we show that Ω' is well defined over $\mathbf{\Lambda}'$. Let σ_0 be a star-type in $\mathbf{\Lambda}$. If σ_0 and all its neighbours are not changed by the algorithm, then Ω' is well defined for σ_0 over $\mathbf{\Lambda}'$. Assume that σ_0 or one of its neighbours is changed by the algorithm. Since Ω concerns neighbours, it suffices to consider the following cases:

   (a) $\sigma_0 = \sigma$ and σ_0 is changed to $\sigma'_0 = \sigma'$. Property 1a of Lemma 11 for every triple ρ in σ' there is a ρ-successor of σ'. By the loop from Line 17 to 19 each ν-predecessor of σ becomes a ν'-predecessor of σ'.

   (b) σ_0 is a ρ-predecessor of σ. Algorithm 6 never changes σ_0 and adds σ' to $\Omega(\sigma_0, \rho)$.

   (c) σ_0 is a ρ-successor of σ and σ_0 is changed to σ'_0. By Property 1.b. of the lemma, σ' is a predecessor of σ'_0 and each R-successor of σ_0 is an R-successor of σ'_0.

   We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1, 2, 3 and 4 of Definition 27.

   (a) Since the algorithm never removes anything from $\langle \mathbf{\Lambda}, \Omega \rangle$, it holds that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1. and 3. of Definition 27.

   (b) We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 2 of Definition 27.

      i. For Property 2a, if a nominal star-type σ_0 is changed to σ'_0 such that $a \in \mathsf{core}_\mathsf{I}(\sigma'_0)$ and $C(a) \in \mathcal{A}$ then $C \in \mathsf{core}_\mathsf{C}(\sigma'_0)$ since $C \in \mathsf{core}_\mathsf{C}(\sigma_0)$ as $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau and the algorithm never remove any concept from $\mathsf{core}_\mathsf{C}(\sigma'_0)$.

      ii. For Property 2b, the proof is similar to the proof of Property 2a.

   (c) We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 4 of Definition 27.

      i. For Property 4a, if a star-type σ_0 is changed to $\sigma'_0 \in \Lambda'_0$ then by Item 1 of Lemma 11 we have $\sigma_0 \in \Lambda_0$. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau, we get that σ_0 is nominal. It holds that $\mathsf{core}_\mathsf{I}(\sigma_0) = \mathsf{core}_\mathsf{I}(\sigma'_0) \neq \emptyset$ and thus σ'_0 is nominal. This implies that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 4a.

      ii. For Property 4b, if a star-type σ_0 is changed to $\sigma'_0 \in \Lambda'_k, k \geq 1$ then by Item 1 of Lemma 11 we have $\sigma_0 \in \Lambda_k, k \geq 1$. Also, σ_0 is non-nominal as $\langle \mathbf{\Lambda}, \Omega \rangle$ as is a pre-compressed tableau and since $\mathsf{core}_\mathsf{I}(\sigma_0) = \mathsf{core}_\mathsf{I}(\sigma'_0) = \emptyset$ then σ'_0 is non-nominal. Besides, by Item 1 of Lemma 11 we get that σ'_0 is clash-free. This implies that$\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 4b of Definition 27.

      iii. For Property 4c. of Definition 27, if a dummy star-type σ_0 is changed to σ'_0 then σ'_0 is also dummy since Algorithm 6 never adds a new triple or changes the tie and tail of a dummy triple. This implies that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 4c of Definition 27.

We have proved that Ω' is well defined over $\mathbf{\Lambda}'$ and all properties of Definition 27 are verified. Therefore, $\langle \mathbf{\Lambda}', \Omega' \rangle$ is a pre-compressed tableau.

3. We now analyze the complexity of Algorithm 6. Since the non loop lines in the algorithm need to traverse only components of a star-type, its neighbours or the equality assertions in $\mathcal{A}$, the complexity of these lines is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. We analyze the complexity of the loops.

   (a) The loop between Lines 3 and 5 traverses each star-type τ in Λ_{i-1} or Λ_0 and each triple in τ. Hence the number of iteration in this loop is bounded by $(|\Lambda_0|+|\Lambda_{i-1}|)\times |\tau|$, where $|\tau|$ is the greatest number of triples of a star-type and $|\Lambda|$ is the greatest number of star-types of a layer.

   (b) Line 14 traverses all the successors of σ, hence the number of iteration in this loop is bounded by $2|\sigma||\Lambda|$.

   (c) The loop between Lines 17 and 19 traverses each triple of a star-type. Hence the number of iteration in this loop is bounded by $|\sigma|$.

   (d) The loop between Lines 20 and 22 traverses all the triples of a star-type. Hence the number of iteration in this loop is bounded by $|\sigma|$.

   Thus the complexity of the loops are all bounded polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. This means that the complexity of Algorithm 6 is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

□

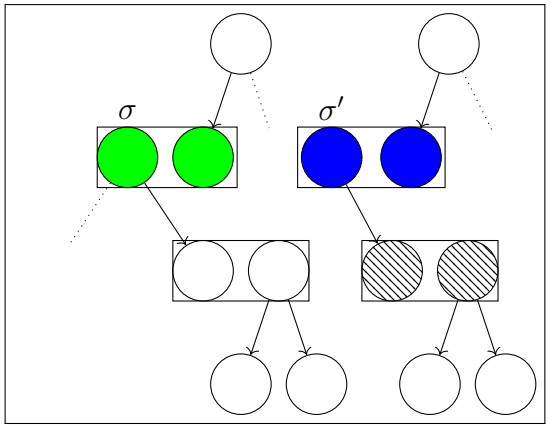
• matchMerge is called for adding new star-types generated by rules that merges two star-types and updating the matching function Ω. This situation is illustrated in Figure 6.7. When a rule merges two star-types σ and σ' (in left-hand side) into a star-type $\sigma \oplus \sigma'$ (in the right-hand side), the predecessors of σ and σ' are transformed into new predecessors of $\sigma \oplus \sigma'$.

Algorithm 7: matchMerge algorithm

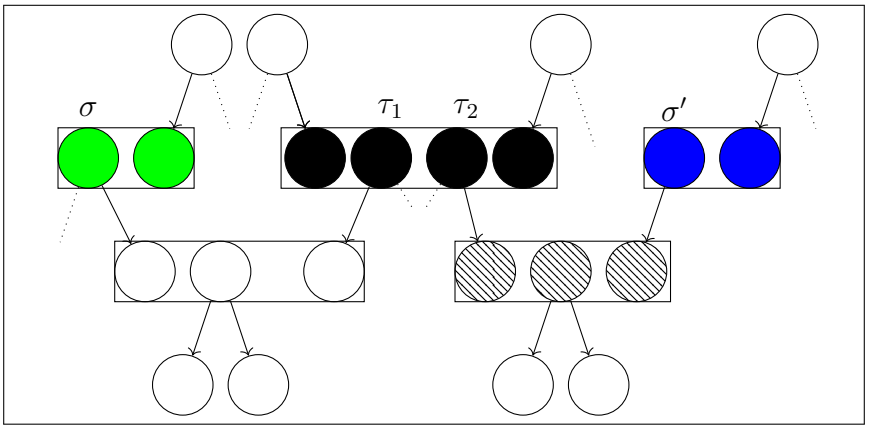
Input: A pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$, two star-types $\sigma, \sigma' \in \Lambda_0$ and a set of star-types $\sigma \oplus \sigma'$

Output: Λ_0 contains $\sigma \oplus \sigma'$ and the updated $\langle \mathbf{\Lambda}, \Omega \rangle$ remains a pre-compressed tableau.

1 **foreach** *clash-free star-type* $\tau \in \sigma \oplus \sigma'$ **do**
2 Add τ to Λ_0;
3 **foreach** $\rho \in \tau$ *and* $\rho' \in \omega$ *with* $\omega \in \{\sigma, \sigma'\}$ *such that* $\mathsf{tie}(\rho) = \mathsf{tie}(\rho')$ *and* $\mathsf{tail}(\rho) = \mathsf{tail}(\rho')$ **do**
4 Set $\Omega(\tau, \rho) \leftarrow \Omega(\omega, \rho')$;
5 **end**
6 **foreach** $\omega' \in \Lambda_0$ *and* $\rho \in \omega'$ *such that* $\omega \in \Omega(\omega', \rho)$ *with* $\omega \in \{\sigma, \sigma'\}$ **do**
7 Create a copy ψ of ω', create a copy ρ' of ρ, set $\mathsf{tail}(\rho') = \mathsf{core}(\tau)$ and replace ρ by ρ' in ψ ;
8 Add ψ to Λ_0 and add τ to $\Omega(\psi, \rho')$;
9 **foreach** $\tau' \in \Lambda_0$ *and* $\nu \in \tau'$ *such that* $\omega' \in \Omega(\tau', \nu)$ **do**
10 Add ψ to $\Omega(\tau', \nu)$;
11 **end**
12 **foreach** $\nu \in \psi$ *such that* $\nu \neq \rho'$ **do**
13 Set $\Omega(\psi, \nu) \leftarrow \Omega(\omega', \nu)$;
14 **end**
15 **end**
16 **end**



(a)



(b)

Figure 6.7 – matchMerge transforms the predecessors of star-types $\sigma \oplus \sigma' = \{\tau_1, \tau_2\}$

Lemma 12. *Let $\langle \mathbf{\Lambda}, \Omega \rangle$ with $\mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^{n}$ be a pre-compressed tableau. Assume that Algorithm 7 takes two star-types $\sigma, \sigma' \in \Lambda_0$ as input.*

1. *For each clash-free star-type τ in $\sigma \oplus \sigma'$, Algorithm 7 adds τ to Λ_0. In this case, each ρ-predecessor ω' of σ or σ' is changed to ψ with a new triple ρ' such that τ is a ρ'-successor of ψ, $\mathsf{tail}(\rho') = \mathsf{core}(\tau)$, and each predecessor of ω' is a predecessor of ψ.*

2. *The updated $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau.*

3. *Algorithm 7 runs in a polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.*

Proof. 1. Line 2 of the algorithm adds each clash-free star-type τ in $\sigma \oplus \sigma'$ to Λ_0. Let ω' be a ρ-predecessor of σ or σ'. In Line 7, the algorithm creates a copy ψ of ω' and replaces ρ with ρ' such that $\mathsf{tail}(\rho') = \mathsf{core}(\tau)$ and adds it to Λ_0 and adds τ to $\Omega(\psi, \rho')$. Hence, ψ be a ρ'-predecessor of τ. Let τ' be a ν-predecessor of ω' with $\tau' \in \Lambda_0$. The loop between Lines 9 and 11 adds ψ to $\Omega(\tau', \nu)$. Hence, τ' is a predecessor of ψ.

2. Let $\langle \mathbf{\Lambda}', \Omega' \rangle$ be the structure obtained from an input $\langle \mathbf{\Lambda}, \Omega \rangle$ by the algorithm where $\mathbf{\Lambda}'$ is extended from $\mathbf{\Lambda}$ with new star-types, and Ω' is extended from Ω for the new star-types. First, we show that Ω' is well defined over $\mathbf{\Lambda}'$. Let σ_0 be a star-type in $\mathbf{\Lambda}$. If σ_0 and all

its neighbours are not changed by the algorithm, then Ω' is well defined for σ_0 over $\mathbf{\Lambda}'$. Assume that σ_0 or one of its neighbours is changed by the algorithm. Since Ω concerns neighbours, it suffices to consider the following cases:

(a) $\sigma_0{=}\sigma$ and σ_0 is changed to $\sigma'_0 \in \sigma \oplus \sigma'$. By Property 1. of the lemma, σ'_0 has a predecessor. Let ψ be a ρ-successor of σ. By Line 4, we have $\Omega'(\sigma'_0, \rho') \neq \emptyset$ for all $\rho' \in \sigma'_0$ such that $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$ and $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$.

(b) $\sigma_0{=}\sigma'$, analogously.

(c) σ_0 is a ρ-predecessor of σ or σ', and σ_0 is changed to σ'_0. By Property 1. of the lemma, σ'_0 is a predecessor of τ in $\sigma \oplus \sigma'$ and each predecessor of σ_0 is a predecessor of σ'_0. Moreover, by the loop between Lines 12 and 14, each ν-successor of σ_0 with $\nu \neq \rho'$ is a ν-successor of σ'_0. Hence, $\Omega'(\sigma'_0, \nu) \neq \emptyset$ for all $\nu \in \sigma'_0$ with $\nu \neq \rho$.

(d) σ_0 is a ρ-successor of σ or σ'. The algorithm never changes σ_0 and its successors. Thus, if $\Omega(\sigma_0, \rho) \neq \emptyset$ then $\Omega'(\sigma_0, \rho) \neq \emptyset$ for all $\rho \in \sigma_0$.

We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1, 2, 3 and 4 of Definition 27.

(a) Since the algorithm never removes anything from $\langle \mathbf{\Lambda}, \Omega \rangle$, it holds that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Properties 1 and 3 of Definition 27.

(b) We now prove that $\langle \mathbf{\Lambda}', \Omega' \rangle$ satisfies Property 2 of Definition 27.

i. We first prove that Property 2a holds in $\langle \mathbf{\Lambda}, \Omega \rangle$.

A. If a star-type $\sigma_0 = \sigma$ is changed to $\sigma'_0 \in \sigma \oplus \sigma'$ where $a \in \mathsf{core}_\mathsf{I}(\sigma'_0)$ and $C(a) \in \mathcal{A}$. If $a \in \mathsf{core}_\mathsf{I}(\sigma)$ we get that $a \in \mathsf{core}_\mathsf{I}(\sigma_0)$ and thus $C \in \mathsf{core}_\mathsf{C}(\sigma_0)$ as $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau and the algorithm never removes any concept from $\mathsf{core}_\mathsf{C}(\sigma'_0)$. Otherwise, if $a \in \mathsf{core}_\mathsf{I}(\sigma')$ then $C \in \mathsf{core}_\mathsf{C}(\sigma')$ as $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau and according to Definition 26, $C \in \mathsf{core}_\mathsf{C}(\sigma'_0)$.

B. If $\sigma_0 = \sigma'$, analogously.

ii. For Property 2b, the proof is similar to the proof of Property 2a.

(c) We now prove Item 4 of Definition 27.

i. For Property 4a, if $\sigma_0 = \sigma$ and σ_0 is changed to $\sigma'_0 \in \Lambda'_0$. It holds that $\sigma \in \Lambda_0$ and since $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau, we get that σ is nominal. Besides, according to Definition 26 it holds that $\mathsf{core}_\mathsf{I}\sigma_0 \subseteq \mathsf{core}_\mathsf{I}\sigma'_0 \neq \emptyset$. We get that σ'_0 is nominal. Analogously, if $\sigma_0 = \sigma'$.

ii. Property 4b and 4c hold since Algorithm 7 never adds a star-type to $\Lambda_k \in \mathbf{\Lambda}, k \geq 1$.

We have proved that Ω' is well defined over $\mathbf{\Lambda}'$ and all properties of Definition 27 are verified. Therefore, $\langle \mathbf{\Lambda}', \Omega' \rangle$ is a pre-compressed tableau.

3. We now analyze the complexity of Algorithm 7. Since the non loop lines in the algorithm need to traverse only components of a star-type, its neighbours or the equality assertions in $\mathcal{A}$, the complexity of these lines is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. We analyze the complexity of the loops.

(a) The loop between Lines 3 and 5 traverses all triples of star-types σ, σ' and all triples of the star-types in $\sigma \oplus \sigma'$. Hence, the number of iterations of this loop is bounded by $|\sigma|^4$ where $|\sigma|$ is the greatest number of triples of a star-type.

(b) The loop between Lines 6 and 15. Line 6 traverses all star-types of the layer Λ_0, all their triples and successors. Hence, the number of iterations of performed by this line is bounded by $|\Lambda_0|^3|\sigma|$. Similarly, the complexity of the loop between Lines 9 and 11 is also bounded by $|\Lambda_0|^3|\sigma|$. The complexity of the loop between Lines 12 and 14 is bounded by $|\sigma|$. Hence, the number of iterations of these loops is also bounded by $|\Lambda_0|^6|\sigma|^3$ where $|\Lambda_0|$ is the greatest number of star-types of in Λ_0.

As a result, the complexity of the loops in Algorithm 7 is bounded by $|\sigma|^4 \times (1 + |\Lambda_0|^6)$. This means that the complexity of Algorithm 7 is polynomial in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

□

Finally, check verifies if some subset Σ' of Λ_0 of size $|\mathbf{I}|$ allows to build a compressed tableau involved in the current pre-compressed tableau.

Lemma 13. *Let $\langle \mathbf{\Lambda}, \Omega \rangle$ be a pre-compressed tableau of an $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O}$. Let* $\mathbf{I}$ *be the set of individuals in $\mathcal{O}$.*

1. *Algorithm 8 returns* true *with $\mathcal{O}$,* $\mathbf{I}$ *and $\langle \mathbf{\Lambda}, \Omega \rangle$ as input iff there exists a compressed tableau $\langle \mathbf{\Lambda}', \Omega' \rangle$ of $\mathcal{O}$ included in $\langle \mathbf{\Lambda}, \Omega \rangle$.*

2. *Algorithm 8 runs in polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.*

Proof. 1. "$\Longrightarrow$": Assume that Algorithm 8 returns true with $\mathcal{O}$, $\mathbf{I}$ and $\langle \mathbf{\Lambda}, \Omega \rangle$, where $\mathbf{\Lambda} = \{\Lambda_k\}_{k=0}^n$.

Let $\langle \mathbf{\Lambda}', \Omega' \rangle$, where $\mathbf{\Lambda}' = \{\Lambda'_k\}_{k=0}^n$, be the structure built by the algorithm. For that we have to prove that Ω' is well-defined over $\mathbf{\Lambda}'$ and each condition in Definition 32 is satisfied by $\langle \mathbf{\Lambda}', \Omega' \rangle$. By construction, $\langle \mathbf{\Lambda}', \Omega' \rangle$ is included in $\langle \mathbf{\Lambda}, \Omega \rangle$, i.e. for every $k \in \{0, \ldots, n\}$, $\Lambda'_k \subseteq \Lambda_k$, and Ω' is a restriction of Ω, i.e. for every $\sigma, \sigma' \in \mathbf{\Lambda}'$ and $\tau \in \sigma$, if $\sigma' \in \Omega'(\sigma, \tau)$ then $\sigma \in \Omega(\sigma, \tau)$. Let us first prove that Ω' is indeed a matching function over $\mathbf{\Lambda}'$. First, we have to prove that there is no star-type $\sigma \in \mathbf{\Lambda}'$ and no non dummy triple $\tau \in \sigma$ such that $\Omega'(\sigma, \tau) = \emptyset$. By contradiction, imagine that such σ and τ exist. Then, there exists $l \in \{0, \ldots, n\}$ such that $\sigma \in \Lambda'_l$. Since the algorithm returns true, the while loop of Line 15 is executed for every layer Λ'_k ($k = n, \ldots, 0$), in particular, for l. Since σ and τ satisfy the condition of the while loop, then its body is executed, i.e. σ is removed from Λ'_l, which contradicts the fact that $\sigma \in \Lambda'_l$.

Now, Condition 1 of Definition 28 holds because Ω is a matching function and it extends Ω'. Let us prove that Condition 2 holds. Let $\sigma \in \mathbf{\Lambda}'$, $\tau \in \sigma$ and $\sigma' \in \Omega'(\sigma, \tau)$. The star-type σ' is added to $\Omega'(\sigma, \tau)$ either in Line 6 of the algorithm, which implies $\sigma, \sigma' \in \Lambda'_0$, or in Line 10, which implies $\sigma \in \Lambda'_k$ and $\sigma' \in \Lambda'_{k+1}$ for all $k \in \{0, \ldots, n-1\}$.

Therefore, Condition 2 holds. This proves that Ω' is a matching function over $\mathbf{\Lambda}'$.

We check now each of the conditions of Definition 32.

(a) We first prove that Condition 1 of Definition 32 is satisfied in $\langle \mathbf{\Lambda}', \Omega' \rangle$ by showing that conditions 1 to 5 of Definition 30 are all satisfied in $\langle \mathbf{\Lambda}', \Omega' \rangle$.

Algorithm 8: check algorithm

Input: An $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, a set $\mathbf{I}$ of individuals and a pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$ of $\mathcal{O}$, where $\mathbf{\Lambda} = \{\Lambda_k\}_{k=0}^{n}$.

Output: true if there exists a compressed tableau $\langle \mathbf{\Lambda}', \Omega' \rangle$ included in $\langle \mathbf{\Lambda}, \Omega \rangle$ and false otherwise.

$\Lambda'_k \leftarrow \emptyset$ for every $k \in \{0, \ldots, n\}$, $\mathbf{\Lambda}' \leftarrow \langle \Lambda'_0, \ldots, \Lambda'_n \rangle, \Omega' \leftarrow \emptyset$;
Let $\langle a_1, \ldots, a_m \rangle$ be an enumeration of $\mathbf{I}$;
foreach $\langle \sigma_1, \ldots, \sigma_m \rangle$ *such that, for every* $i \in \{1, \ldots, m\}$, *either* $\sigma_i = \mathsf{null}$ *or* $\sigma_i \in \Lambda_0$, σ_i *is saturated and clash-free, and there exists a unique* $j \in \{1, \ldots, m\}$ *such that* $a_i \in \mathsf{core}_\mathsf{I}(\sigma_j)$ **do**
  $\Lambda'_0 \leftarrow \{\sigma_i : 1 \leq i \leq m \text{ and } \sigma_i \neq \mathsf{null}\}$;
  if *for every* $\sigma \in \Lambda'_0$ *and* $\rho \in \sigma$ *such that* $\mathsf{tail}_\mathsf{I}(\rho) \neq \emptyset$ *there exists* $\sigma' \in \Lambda'_0$ *such that* $\sigma' \in \Omega(\sigma, \rho)$ **then**
    extend Ω' by letting $\sigma' \in \Omega'(\sigma, \rho)$ for every $\sigma, \sigma' \in \Lambda'_0$ and $\rho \in \sigma$ such that $\mathsf{tail}_\mathsf{I}(\rho) \neq \emptyset$ and $\sigma' \in \Omega(\sigma, \rho)$;
    if *none of the rules of Figure 6.3 can be applied in* Λ'_0 **then**
      if $n \geq 1$ **then**
        foreach $k = 0, \ldots, n-1$ **do**
          add σ' to Λ'_{k+1} and extend Ω' by letting $\sigma' \in \Omega'(\sigma, \rho)$ for every $\sigma \in \Lambda'_k$, $\sigma' \in \Lambda_{k+1}$ and $\rho \in \sigma$ such that $\mathsf{tail}_\mathsf{I}(\rho) = \emptyset$ and $\sigma' \in \Omega(\sigma, \rho)$;
        end
      end
      $k \leftarrow n$;
      while $k \geq 0$ **do**
        while *there exists a star-type* $\sigma \in \Lambda'_k$ *such that* σ *is non blocked and not saturated, or there exists a non-dummy triple* $\rho \in \sigma$ *such that* $\Omega'(\sigma, \rho) = \emptyset$ **do**
          remove σ from Λ'_k;
          remove σ from $\Omega'(\sigma', \rho')$ for every $\sigma' \in \Lambda'_l$, where $l = \max(k-1, 0)$, and $\rho' \in \sigma'$ such that $\sigma \in \Omega'(\sigma', \rho')$;
        end
        $k \leftarrow k - 1$;
      end
      if *for every individual* $a \in \mathbf{I}$ *there exists a star-type* σ *in* Λ'_0 *such that* $a \in \mathsf{core}_\mathsf{I}(\sigma)$ **then**
        return true;
      end
    end
  end
end
return false;

i. For every star-type σ in $\mathbf{\Lambda}'$, σ is added to $\mathbf{\Lambda}'$ by Line 4 or 10. If σ is added to $\mathbf{\Lambda}'$ by Line 4 then it satisfies the condition of Line 3, which means that σ is clash-free and saturated. If σ is added to $\mathbf{\Lambda}'$ by Line 10 then $\sigma \in \Lambda'_k$ for some $k = 1, \ldots, n$. In this case, σ is saturated or blocked because, if σ is not saturated and not blocked then it is removed from Λ'_k by Line 16. In addition, since $\sigma \in \Lambda_k$ and $k \geq 1$, then, by Item 4b of Definition 27, σ is clash-free.

ii. "$\Longrightarrow$": Assume that $\sigma \in \Lambda'_0$. Then, σ is added to Λ'_0 by Line 4, which means that σ satisfies the condition of Line 3. Thus, $\mathsf{core}_\mathsf{I}(\sigma) \neq \emptyset$, i.e. σ is nominal.
"$\Longleftarrow$": Assume that $\sigma \in \mathbf{\Lambda}'$ and that σ is nominal. We have to prove that $\sigma \in \Lambda'_0$. We proceed by contradiction. Assume that $\sigma \notin \Lambda'_0$. Then, there exists $k \in \{1, \ldots, n\}$ such that $\sigma \in \Lambda'_k$. This implies $\sigma \in \Lambda_k$. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau, then, by Item 4a of Definition 27, it follows that σ is non-nominal, which is a contradiction. Therefore, $\sigma \in \Lambda'_0$.

iii. By Lines 3 and 4, Algorithm 8 starts with a first layer such that for every individual there exists a unique star-type in it that contains the individual in its core. The algorithm never adds new star-types to this first layer, but may remove some of them (Line 16). However, since the algorithm returns true, then the condition of Line 21 is satisfied. This means that no star-type is removed from the first layer. Therefore, for every $a \in \mathbf{I}$, there exists a unique start type $\sigma_a \in \Lambda'_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma_a)$.

iv. Assume that $C(a) \in \mathcal{A}$. By 1(a)iii above, there exists a unique $\sigma_a \in \Lambda'_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma_a)$. Now, since $\langle \mathbf{\Lambda}', \Omega' \rangle$ is included in $\langle \mathbf{\Lambda}, \Omega \rangle$, then $\sigma_a \in \Lambda_0$. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ is a pre-compressed tableau, then, by Condition 2a of Definition 27, it follows $C \in \mathsf{core}_\mathsf{C}(\sigma_a)$.

v. Assume $R(a, b) \in \mathcal{A}$. By 1(a)iii above, there exist two unique star-types $\sigma_a, \sigma_b \in \Lambda'_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma)$ and $b \in \mathsf{core}_\mathsf{I}(\sigma_b)$. In addition, since $\langle \mathbf{\Lambda}', \Omega' \rangle$ is included in $\langle \mathbf{\Lambda}, \Omega \rangle$, which is a pre-compressed tableau, then by Condition 2b of Definition 27, there exists a triple $\tau \in \sigma_a$ such that $R \in \mathsf{tie}(\tau)$ and $b \in \mathsf{tail}_\mathsf{I}(\tau)$. Since the algorithm returns true, the condition of Line 5 is satisfied, and, by Line 6, it follows that there exists a star-type $\sigma' \in \Lambda'_0$ such that $\sigma' \in \Omega'(\sigma_a, \tau)$. Since Ω' is matching function, then $\mathsf{tail}_\mathsf{I}(\tau) = \mathsf{core}_\mathsf{I}(\sigma')$. Therefore, $b \in \mathsf{core}_\mathsf{I}(\sigma')$ which implies $\sigma' = \sigma_b$ as σ_b is the only star-type that contains b in its core. Thus, $\sigma_b \in \Omega'(\sigma_a, \tau)$.

(b) We prove respectively below, that Condition 2, 3 and 4 of Definition 32 are satisfied in $\langle \mathbf{\Lambda}, \Omega \rangle$. We proceed by contradiction.

i. Suppose that Condition 2 is not satisfied, this means that there is an equality assertion $a \approx b \in \mathcal{A}$ and there exist two star-types $\sigma, \sigma' \in \mathbf{\Lambda}$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma), b \in \mathsf{core}_\mathsf{I}(\sigma')$ and $\sigma \neq \sigma'$. Which means that the Condition 1 of $\rightarrow_\approx$ rule is satisfied. In addition, since σ and σ' are the unique star-types such that $a \in \mathsf{core}_\mathsf{I}(\sigma), b \in \mathsf{core}_\mathsf{I}(\sigma')$ (by Item 1(a)iii), then Condition 2 of $\rightarrow_\approx$ is also satisfied. Which is not possible since the condition of Line 7 in Algorithm 8 is satisfied. Then Condition 2 is satisfied.

ii. Suppose that condition 3 is not satisfied, this means that there is a link key $\lambda = (\{\langle P_k, Q_k \rangle\}_{k=1}^n \ \mathsf{linkkey} \ \langle C, D \rangle) \in \mathcal{LK}$ and there exist two star-types $\sigma, \sigma' \in \mathbf{\Lambda}$ such that σ and σ' weakly satisfy the condition of λ and $\{C, \mathsf{nnf}(C)\} \cap$

$\mathsf{core}_{\mathsf{C}}(\sigma) = \emptyset$ or $\{D, \mathsf{nnf}(D)\} \cap \mathsf{core}_{\mathsf{C}}(\sigma') = \emptyset$. This means that the conditions of $\rightarrow_{\mathbf{chLK}_1}$ or $\rightarrow_{\mathbf{chLK}_2}$ are satisfied. Which is not possible since the condition of Line 7 in Algorithm 8 is satisfied. Then condition 3 is satisfied.

iii. Suppose that condition 3 is not satisfied, this means that there a link key $\lambda = (\{\langle P_k, Q_k\rangle\}_{k=1}^{n}\ \mathsf{linkkey}\ \langle C, D\rangle) \in \mathcal{LK}$ and there exist two star-types $\sigma, \sigma' \in \mathbf{\Lambda}$ such that σ and σ' satisfy the condition of λ and $\sigma \neq \sigma'$. This means that Condition 1 of $\rightarrow_{\mathbf{LK}}$-rule is satisfied. In addition, Condition 2 of $\rightarrow_{\mathbf{LK}}$-rule is satisfied since star-types in Λ'_0 are unique (by Item 1(a)iii), then there exists no star-type σ_0 such that $\mathsf{core}(\sigma) \subseteq \mathsf{core}(\sigma_0)$ and $\mathsf{core}(\sigma') \subseteq \mathsf{core}(\sigma_0)$. Which is not possible since the condition of Line 7 in Algorithm 8 is satisfied. Then condition 4 is satisfied.

"$\Longleftarrow$": Suppose that there is a compressed tableau $\langle \mathbf{\Lambda}'', \Omega''\rangle$ included in $\langle \mathbf{\Lambda}, \Omega\rangle$. We have to prove that Algorithm 8 returns true. Assume that $\Lambda_0^{''} = \{\sigma_1, \ldots, \sigma_m\}$. Imagine that the loop of Line 3 does not reach $\langle \sigma_1, \ldots, \sigma_m\rangle$. This means that the algorithm returns true with another tuple of star-types, so we are done. So assume that the loop in Line 3 reaches $\langle \sigma_1, \ldots, \sigma_m\rangle$. The condition of the loop is satisfied by $\langle \sigma_1, \ldots, \sigma_m\rangle$ because $\langle \mathbf{\Lambda}'', \Omega''\rangle$ is a compressed tableau and it is included in $\langle \mathbf{\Lambda}, \Omega\rangle$. Let us prove that, for this tuple, the condition of Line 21 is satisfied. We proceed by contradiction. Assume that the condition is not satisfied. This means that there exists $i \in \{1, \ldots, m\}$ such that σ_i is removed from the first layer. This must happen in Line 16, which means that the condition of Line 15 is satisfied. Thus, σ_i is non blocked and not saturated, or there exists a non-dummy triple $\tau \in \sigma_i$ such that $\Omega'(\sigma_i, \tau) = \emptyset$. Since $\sigma_i \in \Lambda_0^{''}$, then it is saturated, so it must be that there exists a non-dummy triple $\tau \in \sigma_i$ such that $\Omega'(\sigma_i, \tau) = \emptyset$. Also, since $\Omega^{''}$ is a matching function, then $\Omega''(\sigma_i, \tau) \neq \emptyset$, so there exists a star-type $\sigma' \in \mathbf{\Lambda}''$ such that $\sigma' \in \Omega''(\sigma_i, \tau)$. We have $\sigma' \in \Omega(\sigma_i, \tau)$ too because $\Omega^{''}$ is a restriction of Ω. Then, either in Line 6 or Line 10, it is set $\sigma' \in \Omega'(\sigma_i, \tau)$. If $\Omega'(\sigma_i, \tau) = \emptyset$ this means that σ' is removed from $\Omega'(\sigma_i, \tau)$ by Line 16. We proceed using the same argument on the successors of σ' until we reach the layer Λ_{n-1}. Suppose that there exists a star-type $\sigma'_{n-1} \in \Lambda'_{n-1}$ a direct or indirect successor of σ' through Ω'' such that σ'_{n-1} is removed from Λ'_{n-1}. This means that either σ'_{n-1} is not saturated or there exits a triple τ'_{n-1} such that $\Omega'(\sigma'_{n-1}, \tau'_{n-1}) = \emptyset$. Since $\sigma'_{n-1} \in \Lambda_{n-1}^{''}$ then it is saturated. Since $\Omega^{''}$ is a matching function, then $\Omega''(\sigma'_{n-1}, \tau'_{n-1}) \neq \emptyset$, so there exists a star-type $\sigma'_n \in \mathbf{\Lambda}''$ such that $\sigma'_n \in \Omega''(\sigma'_{n-1}, \tau'_{n-1})$. We have $\sigma'_n \in \Omega(\sigma'_{n-1}, \tau'_{n-1})$ too because $\Omega^{''}$ is a restriction of Ω. Then, either in Line 6 or Line 10, it is set $\sigma'_n \in \Omega'(\sigma'_{n-1}, \tau'_{n-1})$. If $\Omega'(\sigma'_{n-1}, \tau'_{n-1}) = \emptyset$ this means that σ'_n is removed from $\Omega'(\sigma_i, \tau)$ by Line 16. We have $\sigma'_n \in \Lambda''_n$ then it is either dummy or blocked and it will not be removed by Line 16 as the conditions of Line 15 are not satisfied.

2. We now analyze the complexity of Algorithm 8. We denote by ℓ the size of the ontology $\mathcal{O}$, i.e., ℓ=$|\mathcal{O}|$ and by c is the size of the sub-concepts of $\mathcal{O}$, i.e., c=$|\mathsf{sub}(\mathcal{O})|$.

   (a) We first calculate the complexity of Line 3 of the algorithm. In Line 3, the algorithm uses a vector $\langle \sigma_1, \cdots, \sigma_k\rangle$ to represent a possible choice of Λ'_0 where each σ_i corresponds to a star-type containing an individual $a_i \in \mathbf{I}$. Hence, $k \leq \ell$. Moreover, each σ_i takes at most 2^c different values. Hence, there are at most 2^{c^ℓ} different vectors $\langle \sigma_1, \cdots, \sigma_k\rangle$. Line 3 also checks the saturation and clash-freeness of each

star-type in $\langle \sigma_1, \cdots, \sigma_k \rangle$. We respectively analyze below the complexity of each task:

i. The task of checking if a star-type is saturated needs to traverse the components of the star-type and the GCIs in $\mathcal{T}$. Since, $|\mathcal{T}| \leq |\mathcal{O}|$, then the complexity of this task is bounded by $\ell \|\sigma\|$, where $\|\sigma\|$ is the size of a star-type σ in $\mathbf{\Lambda}$.

ii. The task of checking if a star-type is clash-free needs to traverse the components of a star-type and the equality assertions in $\mathcal{A}$. Since, $|\mathcal{A}| \leq |\mathcal{O}|$, then the complexity of this task is bounded by $\ell \|\sigma\|$.

(b) Lines 5 and 6 iterate over every star-type σ in Λ'_0, every triple $\rho \in \sigma$ and the set of ρ-successors of σ in Λ'_0. Thus the complexity of this task is bounded by $|\Lambda'_0|^2|\sigma|$, where $|\Lambda'_0|$ is the greatest number of star-types in Λ'_0 and $|\sigma|$ is the number of triples in σ.

(c) Line 7 traverses each subset $\{\omega_1, \ldots, \omega_p\}$ where $\omega_i \in \Lambda'_0$, $p \leq q$ and q is the greatest number of star-types involved in a link key. Then it checks if there is a rule in Figure 6.3 applicable on $\{\omega_1, \ldots, \omega_p\}$. The number of different sets of size p is bounded by $|\Lambda'_0|^p$. Checking the applicability of rules in Figure 6.3 on $\{\omega_1, \ldots, \omega_p\}$ necessitates to traverse equality assertions in $\mathcal{A}$, the link keys in $\mathcal{LK}$ and each star-type in $\{\omega_1, \ldots, \omega_p\}$. Thus the complexity of this task is bounded by $|\Lambda'_0|^p(|\mathcal{A}| + |\mathcal{LK}|)\|\sigma\| \leq \ell^2 |\Lambda'_0|^p \|\sigma\|$.

(d) Line 9 iterates over each layer $\Lambda'_k, 1 \leq k \leq n$ in $\mathbf{\Lambda}'$. Line 10 iterates over every star-type σ in Λ'_k, every triple $\rho \in \sigma$ and every ρ-successor of σ. Thus the complexity of the loop between Line 9 and 10 is bounded by $n^2 |\Lambda'_k| |\sigma|$.

(e) The loop in Line 14 executes n times the loop in Line 15. Line 15 iterates over every non-blocked star-type $\sigma \in \Lambda_k, k \leq n$, checks if it is saturated and checks every ρ-successor of σ where $\rho \in \sigma$. The task of checking if σ is blocked needs to check every layer Λ_k in $\mathbf{\Lambda}$ and each star-type ω in Λ_k. Thus the complexity of this task is bounded by $n|\Lambda_k|\|\sigma\|(n|\Lambda_k|\|\sigma\| + \ell\|\sigma\|)$.

(f) Finally, Line 21 iterates over every individual $a \in \mathbf{I}$ and every star-type $\sigma \in \Lambda'_0$ thus the complexity of this line is bounded by $\ell^2 \|\sigma\|$.

Since the size of $\langle \mathbf{\Lambda}', \Omega' \rangle$ is bounded by the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. Then the complexity of Algorithm 8 is polynomially bounded by the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

□

We now give the main algorithm which as previously explained checks the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology $\mathcal{O}$. Algorithm 9 returns Yes when $\mathcal{O}$ is consistent and No otherwise.

Algorithm 9: Compressed tableau algorithm for $\mathcal{ALC}$+$\mathcal{LK}$ ontology consistency checking

Input: an $\mathcal{ALC} + \mathcal{LK}$ ontology $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$, the set $\mathbf{I}$ of individuals in $\mathcal{O}$ and the labelling function L defined over $\mathbf{I}$.

Output: Returns Yes if $\mathcal{O}$ is consistent and No otherwise.

1 $\langle \Lambda_0, \Omega \rangle \leftarrow \mathsf{init}(\mathcal{A}, \mathbf{I}, L), \mathbf{\Lambda} \longleftarrow [\Lambda_0], \mathsf{processed}_{\mathsf{ALC}} \leftarrow \emptyset, \mathsf{processed}_{\mathsf{LK}} \leftarrow \emptyset$;
2 changed $\leftarrow$ true;
3 **while** changed *is* true **do**
4 changed $\leftarrow$ false;
5 **if** check$(\mathcal{A}, \mathcal{LK}, \langle \mathbf{\Lambda}, \Omega \rangle)$ **then**
6 **return** Yes;
7 **end**
8 **if** *there exists a star-type σ in $\Lambda_i, i \geq 0$ and a saturation rule r in Figure 6.2 such that r can be applied on σ and $\sigma \notin \mathsf{processed}_{\mathsf{ALC}}$* **then**
9 add σ to $\mathsf{processed}_{\mathsf{ALC}}$;
10 apply r to σ;
11 changed $\leftarrow$ true;
12 **end**
13 **foreach** $\{\omega_1, \cdots, \omega_n\}$ *with $\omega_i \in \Lambda_0$, $\{\omega_1, \cdots, \omega_n\} \notin \mathsf{processed}_{\mathsf{LK}}$ and n is the greatest number of star-types involved in a link key* **do**
14 **while** *there exists a rule r in Figure 6.3 applicable to $\{\omega_1, \cdots, \omega_n\}$* **do**
15 apply r to $\{\omega_1, \cdots, \omega_n\}$;
16 changed $\leftarrow$ true;
17 **end**
18 add $\{\omega, \cdots, \omega_n\}$ to $\mathsf{processed}_{\mathsf{LK}}$;
19 **end**
20 **end**
21 **return** No;

6.4 Examples

Example 14. *We show the execution of algorithm 9. The following example checks the inference of a chained link generation. It also shows how the algorithm deals with non-determinism and how the blocking mechanism works.*

Entailment:
$(\langle P, R \rangle \text{ linkkey } \langle A, B \rangle), (\langle Q, S \rangle \text{ linkkey } \langle C, D \rangle), C(a), P(a, c), C(c),$
$Q(c, e), B(b), R(b, d), D(d), S(d, e), \exists T.(\exists U.E \sqcup \exists V.(\exists U.E))(e) \models a \approx b.$

Initial knowledge base:
$\mathcal{A}$=$\{A(a), P(a, c), C(c), Q(c, e), B(b), R(b, d), D(d), S(d, e), \exists T.(\exists U.E \sqcup \exists V.(\exists U.E))(e), a \not\approx b\}$,
$\mathcal{T} = \emptyset$, *and*
$\mathcal{LK} = \{(\langle P, R \rangle \text{ linkkey } \langle A, B \rangle), (\langle Q, S \rangle \text{ linkkey } \langle C, D \rangle)\}$

Algorithm:

1. *Algorithm 9 sets* $\mathsf{processed}_{\mathsf{ALC}} \leftarrow \emptyset, \mathsf{processed}_{\mathsf{LK}} \leftarrow \emptyset$ *and calls Algorithm 4 (*init*) as shown in Line 1. Algorithm 4 builds star-types from $\mathcal{A}$ and adds them to Λ_0 and connects*

them by Ω. We get Λ_0=$\{\sigma_a, \sigma_b, \sigma_c, \sigma_d, \sigma_e\}$ where $\sigma_a = \{\{\langle\{a\}, \{A\}\rangle, P, \langle\{c\}, \{C\}\rangle\}\}$, $\sigma_b = \{\{\langle\{b\}, \{B\}\rangle, R, \langle\{d\}, \{D\}\rangle\}\}$, $\sigma_c = \{\{\langle\{c\}, \{C\}\rangle, Q, \langle\{e\}, \{\exists T.(\exists U.E \sqcup \exists V.(\exists U.E))\}\rangle\}\}$, $\sigma_d = \{\{\langle\{d\}, \{D\}\rangle, S, \langle\{e\}, \{\exists T.(\exists U.E \sqcup \exists V.(\exists U.E))\}\rangle\}\}$, $\sigma_e = \{\{\langle\{e\}, \{\exists T.(\exists U.E \sqcup \exists V.(\exists U.E))\}\rangle, \emptyset, \langle\emptyset, \emptyset\rangle\}\}$. Then it connects the star-types according to the matching function Ω. We get that σ_c and σ_d are successors of σ_a and σ_b respectively and σ_e is a common successor of σ_c and σ_d. As σ_e is not saturated since $\rightarrow_\exists$-rule is applicable to σ_e as $\exists T.(\exists U.E \sqcup \exists V.(\exists U.E)) \in \mathsf{core}_\mathsf{C}(\sigma)$, no compressed tableau built for $\mathcal{O}$ by Algorithm 8. Now, Algorithm 9 performs Line 8 and checks if there is a rule in Figure 6.2 or Figure 6.3 applicable on star-types in $\mathbf{\Lambda}$.

2. *σ_e is not saturated since $\rightarrow_\exists$-rule is applicable to σ_e as $\exists T.(\exists U.E \sqcup \exists V.(\exists U.E)) \in \mathsf{core}_\mathsf{C}(\sigma_e)$. According to action 1 and 2 of $\rightarrow_\exists$-rule, a new copy σ_e^1 of σ_e is created and a triple $\{\langle\{e\}, \{\exists T.(\exists U.E \sqcup \exists V.(\exists U.E))\}\rangle, T, \langle\emptyset, \exists U.E \sqcup \exists V.(\exists U.E)\rangle\}$ is added σ_e^1. By action 3 of the rule (Algorithm 6), σ_e^1 is added to Λ_0 as it is clash-free. σ_e^1 is now a new successor of σ_c and σ_d. Also new layer Λ_1 is added to $\mathbf{\Lambda}$ and a successor ω_1 of σ_e^1 with $\mathsf{core}(\omega_1) = \langle\emptyset, \exists U.E \sqcup \exists V.(\exists U.E)\rangle$ is built and added to Λ_1.*

3. *ω_1 is not saturated since $\rightarrow_\sqcup$-rule is applicable to ω_1 as $\exists U.E \sqcup \exists V.(\exists U.E) \in \mathsf{core}_\mathsf{C}(\omega_1)$. According to action 1 and 2 of $\rightarrow_\sqcup$-rule, two new copies ω_1^1 and ω_1^2 of ω_1 with $\mathsf{core}(\omega_1^1) = \langle\emptyset, \{\exists U.E \sqcup \exists V.(\exists U.E), \exists U.E\}\rangle$ and $\mathsf{core}(\omega_1^2) = \langle\emptyset, \{\exists U.E \sqcup \exists V.(\exists U.E), \exists V.(\exists U.E)\}\rangle$ are built. By action 3 (Algorithm 5), ω_1^1 and ω_1^2 are added to Λ_1 as they are clash-free. Also, two predecessors σ_e^{11} and σ_e^{12} of ω_1^1 and ω_1^2 respectively are built and added to Λ_0. This shows that when Algorithm 9 encounters a disjunction between two concepts it does not duplicate the whole structure being built, but it builds instead two copies of the star-type containing this disjunction. In this way, global nondeterminism is reduced to local nondeterminism.*

4. *ω_1^1 is not saturated since $\rightarrow_\exists$-rule is applicable ω_1^1 as $\exists U.E$ in $\mathsf{core}_\mathsf{C}(\omega_1^1)$. According to action 1 and 2 of $\rightarrow_\exists$-rule, a new star-type ω_1^{11} is created. By action 3 (Algorithm 6), ω_1^{11} added to Λ_1. Besides, a new layer Λ_2 is added to $\mathbf{\Lambda}$ and a successor ψ_1 of ω_1^1 with $\mathsf{core}(\psi_1) = \langle\emptyset, \{E\}\rangle$ is built and added to Λ_2.*

5. *ω_1^2 is not saturated since $\rightarrow_\exists$ rule is applicable to ω_1^2 since $\exists V.(\exists U.E) \in \mathsf{core}_\mathsf{C}(\omega_1^2)$. According to action 1 and 2 of $\rightarrow_\exists$-rule, a new star-type ω_1^{21} is with a triple $\{\langle\emptyset, \{\exists V.(\exists U.E)\}\rangle, V, \langle\emptyset, \{\exists U.E\}\rangle\}$ is added σ_e^1. By action 3 (Algorithm 6), ω_1^{21} is added to Λ_1 and a successor ψ_1 is added to Λ_2. ψ_1 is blocked by ω_1^1 as $\mathsf{core}_\mathsf{C}(\psi_1) \subseteq \mathsf{core}_\mathsf{C}(\omega_1^1)$. Now there is no rule in Figure 6.2 applicable on star-types in $\mathbf{\Lambda}$ and the algorithm checks if there exists a rule in Figure 6.3 applicable to some sets of star-types in Λ_0.*

6. *$\rightarrow_{\mathbf{LK}}$ is applicable to $\{\sigma_c, \sigma_d, \sigma_e\}$ as σ_c and σ_d satisfy the link key $(\langle Q, S\rangle$ $\mathsf{linkkey}$ $\langle C, D\rangle)$ through the same successors σ_e. Then the algorithm applies $\rightarrow_{\mathbf{LK}}$ to $\{\sigma_c, \sigma_d, \sigma_e\}$. According to action 1 of $\rightarrow_{\mathbf{LK}}$ a new star-type σ_{cd} where $\sigma_{cd} = \{\{\langle\{c, d\}, \{C, D\}\rangle, Q, \langle\{e\}, \{E\}\rangle\}, \{\langle\{c, d\}, \{C, D\}\rangle, S, \langle\{e\}, \{E\}\rangle\}\}$ is created. By action 3 (Algorithm 7), σ_{cd} is added to Λ_0. Besides, two new star-types predecessors σ'_a and σ'_b are created and added to Λ_0 where $\sigma'_a = \{\{\langle\{a\}, \{A\}\rangle, P, \langle\{c, d\}, \{C, D\}\rangle\}\}$ and $\sigma'_b = \{\{\langle\{b\}, \{B\}\rangle, R, \langle\{c, d\}, \{C, D\}\rangle\}\}$. So σ_{cd} becomes a successor of σ'_a and σ'_b.*

7. *Now, $\rightarrow_{\mathbf{LK}}$ rule is applicable to $\{\sigma'_a, \sigma'_b, \sigma_{cd}\}$ since σ'_a and σ'_b satisfy the link key $(\langle P, R\rangle$ linkkey $\langle A, B\rangle)$ through the same successor σ_{cd}. According to action 1 of $\rightarrow_{\mathbf{LK}}$ a new star-type σ'_{ab} is created where $\sigma'_{ab} = \{\{\langle\{a, b\}, \{A, B\}\rangle, P, \langle\{c, d\}, \{C, D\}\rangle\}, \{\langle\{a, b\}, \{A, B\}\rangle, R, \langle\{c, d\}, \{C, D\}\rangle\}\}$. But σ'_{ab} is not added to Λ_0 by action 3 (Algorithm 7). This because σ'_{ab} is not clash-free as $a \not\approx b \in \mathcal{A}$ and $\{a, b\} \subseteq \mathsf{core}_{\mathsf{I}}(\sigma'_{ab})$.*

Now check *returns* false *and there is no rule in Figure 6.2 or Figure 6.3 applicable on star-types in $\mathbf{\Lambda}$ and there is no compressed tableau built for $\mathcal{O}$. Hence, the entailment is valid.*

Check *returns* false *because for every set of star-types $\{\tau_1, \cdots, \tau_k\}, k \leq 5$ built from Λ_0 by Line 3 of Algorithm 8, either $\rightarrow_{\mathbf{LK}}$ is applicable to $\{\tau_1, \cdots, \tau_{k'}\}, k' \leq k$ with the link key $(\langle Q, S\rangle$ linkkey $\langle C, D\rangle)$ or $(\langle P, R\rangle$ linkkey $\langle A, B\rangle)$ and hence the condition of Line 7 of Algorithm 8 is not satisfied.*

6.5 Properties of the algorithm

6.5.1 Soundness

Lemma 14 (Soundness)**.** *Let $\mathcal{O}$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. If there exists a compressed tableau of $\mathcal{O}$ then $\mathcal{O}$ is consistent.*

Proof. Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK}\rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\mathbf{I}$ and $\mathbf{R}$ be the sets of individuals and roles of $\mathcal{O}$.

Let us assume that there exists a compressed tableau $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega\rangle$ of $\mathcal{O}$, where $\mathbf{\Lambda} = \langle \Lambda_k\rangle_{k=0}^{n}$. We consider the set Path of all paths of star-types in $\mathbf{\Lambda}$, i.e. $\mathsf{Path} = \{\langle \sigma_1, \ldots, \sigma_m\rangle \mid m \geq 1$ and $\sigma_k \in \mathbf{\Lambda}$ for $1 \leq k \leq m\}$. If $\mathsf{p} \in \mathsf{Path}$, $\mathsf{last}(\mathsf{p})$ denotes the last element of p and $\mathsf{length}(\mathsf{p})$ denotes its length; if $\mathsf{p}, \mathsf{p}' \in \mathsf{Path}$, $\mathsf{p} \cdot \mathsf{p}'$ denotes the concatenation of p and p′. If $a \in \mathbf{I}$ then, by (3) of Definition 30, there exists a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. We denote $\langle \sigma_a\rangle \in \mathsf{Path}$ by p_a.

We define an interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}}\rangle$ from $\mathcal{CT}$ in the following way:

1. $\Delta^{\mathcal{I}}$ is inductively defined as follows:
   (a) $\mathsf{p}_a \in \Delta^{\mathcal{I}}$ for every $a \in \mathbf{I}$;
   (b) if $\mathsf{p} \in \Delta^{\mathcal{I}}$ and there exist $\tau \in \mathsf{last}(\mathsf{p})$ and $\sigma \in \mathbf{\Lambda}$ such that $\sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$ then, if σ is not blocked then $\mathsf{p} \cdot \langle\sigma\rangle \in \Delta^{\mathcal{I}}$, otherwise, $\mathsf{p} \cdot \langle\mathsf{b}(\sigma)\rangle \in \Delta^{\mathcal{I}}$.
2. For every $a \in \mathbf{I}$, $a^{\mathcal{I}} = \mathsf{p}_a$.
3. For every concept name A in $\mathcal{O}$, $A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} \mid A \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))\}$.
4. For every role name $R \in \mathbf{R}$, $R^{\mathcal{I}} = \Delta_1^R \cup \Delta_2^R \cup \Delta_3^R$, where

$$\Delta_1^R = \{(\mathsf{p}_a, \mathsf{p}_b) : a, b \in \mathbf{I} \text{ and there exists } \tau \in \sigma_a \text{ such that } \sigma_b \in \Omega(\sigma_a, \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

$$\Delta_2^R = \{(\mathsf{p}, \mathsf{p} \cdot \langle\sigma\rangle) : \mathsf{p} \in \Delta^{\mathcal{I}}, \sigma \in \mathbf{\Lambda}, \sigma \text{ is not blocked and there exists } \tau \in \mathsf{last}(\mathsf{p}) \text{ such that } \sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

$$\Delta_3^R = \{(\mathsf{p}, \mathsf{p} \cdot \langle\mathsf{b}(\sigma)\rangle) : \mathsf{p} \in \Delta^{\mathcal{I}}, \sigma \in \mathbf{\Lambda}, \sigma \text{ is blocked and there exists } \tau \in \mathsf{last}(\mathsf{p}) \text{ such that } \sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

Let us prove that $\mathcal{I}$ is indeed an interpretation. We only need to prove that $\Delta^{\mathcal{I}} \neq \emptyset$ and that, for every $R \in \mathbf{R}$, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Since $\mathbf{I} \neq \emptyset$, then, there exists at least one individual $a \in \mathbf{I}$. Thus, by (1.a), $\mathsf{p}_a \in \Delta^{\mathcal{I}}$. Therefore, $\Delta^{\mathcal{I}} \neq \emptyset$. Now, let $R \in \mathbf{R}$. By (1.a), $\Delta_1^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. By (1.b), $\Delta_2^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $\Delta_3^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This implies $\Delta_1^R \cup \Delta_2^R \cup \Delta_3^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Therefore, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Let us prove that $\mathcal{I}$ is a model of $\mathcal{O}$, i.e. $\mathcal{I}$ satisfies all the assertions of $\mathcal{A}$, all the GCIs of $\mathcal{T}$ and all the link keys of $\mathcal{LK}$. We start by proving that $\mathcal{I}$ satisfies all the assertions of $\mathcal{A}$.

Assume that $a \approx b \in \mathcal{A}$. By the definition of $\mathcal{I}$, $a^{\mathcal{I}} = \mathsf{p}_a$ and $b^{\mathcal{I}} = \mathsf{p}_b$, where $\mathsf{p}_a = \langle \sigma_a \rangle$, $\mathsf{p}_b = \langle \sigma_b \rangle$ and σ_a and σ_b are the unique star-types of Λ_0 such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_b)$. Since $\mathcal{CT}$ is a compressed tableau, by (2) of Definition 32, we have $\sigma_a = \sigma_b$. This implies $\mathsf{p}_a = \mathsf{p}_b$. Therefore, $a^{\mathcal{I}} = b^{\mathcal{I}}$, i.e. $\mathcal{I} \models a \approx b$.

Assume that $a \not\approx b \in \mathcal{A}$. We proceed by contradiction. Assume $\mathcal{I} \not\models a \not\approx b$. This means $a^{\mathcal{I}} = b^{\mathcal{I}}$. Then, by the definition of $\mathcal{I}$, $\mathsf{p}_a = \mathsf{p}_b$. Thus, $\sigma_a = \sigma_b$, where σ_a and σ_b are the unique star-types of Λ_0 such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_b)$. Let $\sigma = \sigma_a = \sigma_b$. Therefore, $\sigma \in \mathbf{\Lambda}$ and $a, b \in \mathsf{core}_{\mathsf{I}}(\sigma)$. By (2) of Definition 25, σ is not clash-free. This contradicts the fact that $\mathcal{CT}$ is a compressed tableau, so it must be $\mathcal{I} \models a \not\approx b$.

Assume that $R(a, b) \in \mathcal{A}$. Since $\mathcal{CT}$ is a compressed tableau, by (5) of Definition 30, there exists $\tau \in \sigma_a$ such that $\sigma_b \in \Omega(\sigma_a, \tau)$ and $R \in \mathsf{tie}(\tau)$. Then, by the definition of $\mathcal{I}$, $(\mathsf{p}_a, \mathsf{p}_b) \in R^{\mathcal{I}}$. Also, $a^{\mathcal{I}} = \mathsf{p}_a$ and $b^{\mathcal{I}} = \mathsf{p}_b$. Therefore, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, which is the same as $\mathcal{I} \models R(a, b)$.

Assume $C(a) \in \mathcal{A}$. To prove $\mathcal{I} \models C(a)$, we use the following:

$$\text{For every } \mathsf{p} \in \Delta^{\mathcal{I}}, \text{ if } E \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p})) \text{ then } \mathsf{p} \in E^{\mathcal{I}} \qquad (\dagger)$$

Assume that (†) is true. Since $C(a) \in \mathcal{A}$, by (4) of Definition 30, we have that $C \in \mathsf{core}_{\mathsf{C}}(\sigma_a)$. Then, $C \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}_a))$. By (†), $\mathsf{p}_a \in C^{\mathcal{I}}$. Since $a^{\mathcal{I}} = \mathsf{p}_a$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$, i.e. $\mathcal{I} \models C(a)$.

We now show that $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$. Let $C \sqsubseteq D \in \mathcal{T}$. Let $\mathsf{p} \in C^{\mathcal{I}}$. Since $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is a compressed tableau and $\mathsf{last}(\mathsf{p}) \in \mathbf{\Lambda}$, by (1) of Definition 30, $\mathsf{last}(\mathsf{p})$ is saturated. Since $C \sqsubseteq D \in \mathcal{T}$, by (5) of Definition 24, then $\mathsf{nnf}(C)$ or D belongs to $\mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. If $\mathsf{nnf}(C) \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$, by (†), $\mathsf{p} \in (\mathsf{nnf}(C))^{\mathcal{I}}$. This contradicts $\mathsf{p} \in C^{\mathcal{I}}$ because $(\mathsf{nnf}(C))^{\mathcal{I}} = (\neg C)^{\mathcal{I}}$ and $(\neg C)^{\mathcal{I}} \cap C^{\mathcal{I}} = \emptyset$. Therefore, it must be $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Then, by (†), $\mathsf{p} \in D^{\mathcal{I}}$. This proves $\mathcal{I} \models C \sqsubseteq D$.

Finally, we show that $\mathcal{I}$ satisfies all link keys in $\mathcal{LK}$. Let $\lambda = (\{\langle P_i, Q_i \rangle\}_{i=1}^n \text{ linkkey } \langle C, D \rangle) \in \mathcal{LK}$. Let $\mathsf{p}, \mathsf{q}, \mathsf{r}_1, \ldots, \mathsf{r}_n \in \Delta^{\mathcal{I}}$ such that $\mathsf{p} \in C^{\mathcal{I}}$, $\mathsf{q} \in D^{\mathcal{I}}$, and $(\mathsf{p}, \mathsf{r}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{r}_i) \in Q_i^{\mathcal{I}}$ for $1 \leq i \leq n$. We have to prove that $\mathsf{p} = \mathsf{q}$.

Assume that there exists $i_0 \in \{1, \ldots, n\}$ such that $\mathsf{length}(\mathsf{r}_{i_0}) \geq 2$. Then, $(\mathsf{p}, \mathsf{r}_{i_0}) \in \Delta_2^{P_{i_0}} \cup \Delta_3^{P_{i_0}}$ and $(\mathsf{q}, \mathsf{r}_{i_0}) \in \Delta_2^{Q_{i_0}} \cup \Delta_3^{Q_{i_0}}$. Therefore, there exist $\sigma, \sigma' \in \mathbf{\Lambda}$ such that $\mathsf{r}_{i_0} = \mathsf{p} \cdot \langle \sigma \rangle$ and $\mathsf{r}_{i_0} = \mathsf{q} \cdot \langle \sigma' \rangle$. Then, $\mathsf{p} \cdot \langle \sigma \rangle = \mathsf{q} \cdot \langle \sigma' \rangle$, and, thus, $\mathsf{p} = \mathsf{q}$.

Assume that $\mathsf{length}(\mathsf{r}_i) = 1$ for every $1 \leq i \leq n$. Then, $(\mathsf{p}, \mathsf{r}_i) \in \Delta_1^{P_i}$ and $(\mathsf{q}, \mathsf{r}_i) \in \Delta_1^{Q_i}$ for every $1 \leq i \leq n$. Therefore, there exist $a, b_1, \ldots, b_n \in \mathbf{I}$ and $\tau_1, \ldots, \tau_n \in \sigma_a$ such that $\mathsf{p} = \mathsf{p}_a$, $\mathsf{r}_i = \mathsf{p}_{b_i}$, $\sigma_{b_i} \in \Omega(\sigma_a, \tau_i)$ and $P_i \in \mathsf{tie}(\tau_i)$ for $1 \leq i \leq n$, and there exist $c, d_1, \ldots, d_n \in \mathbf{I}$ and $\tau_1', \ldots, \tau_n' \in \sigma_a$ such that $\mathsf{q} = \mathsf{p}_c$, $\mathsf{r}_i = \mathsf{p}_{d_i}$, $\sigma_{d_i} \in \Omega(\sigma_a, \tau_i')$ and $Q_i \in \mathsf{tie}(\tau_i')$ for $1 \leq i \leq n$. Therefore, $\mathsf{p}_{b_i} = \mathsf{p}_{d_i}$ for $1 \leq i \leq n$. Since, for any $x \in \mathbf{I}$, $\mathsf{p}_x = \langle \sigma_x \rangle$, then $\sigma_{c_i} = \sigma_{d_i}$ for $1 \leq i \leq n$. Let $\sigma_i = \sigma_{b_i} = \sigma_{d_i}$ for $1 \leq i \leq n$. Then, $\sigma_i \in \Omega(\sigma_a, \tau_i) \cap \Omega(\sigma_c, \tau_i')$, thus $\Omega(\sigma_a, \tau_i) \cap \Omega(\sigma_c, \tau_i') \neq \emptyset$ for $1 \leq i \leq n$. Then, σ_a and σ_c weakly satisfy the condition of λ. Since $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is a compressed tableau, by (3) of Definition 32, we have $\{C, \mathsf{nnf}(C)\} \cap$

$\mathsf{core}_\mathsf{C}(\sigma_a) \neq \emptyset$ and $\{D, \mathsf{nnf}(D)\} \cap \mathsf{core}_\mathsf{C}(\sigma_b) \neq \emptyset$. Assume that $\mathsf{nnf}(C) \in \mathsf{core}_\mathsf{C}(\sigma_c)$. Then, by (†), we have $\mathsf{p} \in (\mathsf{nnf}(C))^\mathcal{I}$, which contradicts $\mathsf{p} \in C^\mathcal{I}$. Thus, it must be $C \in \mathsf{core}_\mathsf{C}(\sigma_a)$. In a similar manner, $D \in \mathsf{core}_\mathsf{C}(\sigma_c)$. Therefore, σ_a and σ_c satisfy the condition of λ. By (4) of Definition 32, $\sigma_a = \sigma_c$. This implies $\mathsf{p}_a = \mathsf{p}_c$, thus $\mathsf{p} = \mathsf{q}$.

It remains to prove (†). Let $\mathsf{p} \in \Delta^\mathcal{I}$ and assume $E \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. We have to prove that $\mathsf{p} \in E^\mathcal{I}$. We will proceed by induction on the length of E, but, before anything else, note that, since $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is a compressed tableau, then $\mathsf{last}(\mathsf{p})$ is saturated and clash-free.

1. Assume $E = A$, where A is a concept name. We have $A \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. By the definition of $\mathcal{I}$, $A^\mathcal{I} = \{\mathsf{p} \in \Delta^\mathcal{I} \mid A \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))\}$. Therefore, $\mathsf{p} \in A^\mathcal{I}$.

2. Assume $E = E_1 \sqcap E_2$. We have that $E_1 \sqcap E_2 \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is saturated, then $E_1, E_2 \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. By induction hypothesis, $\mathsf{p} \in E_1^\mathcal{I}$ and $\mathsf{p} \in E_2^\mathcal{I}$, which implies $\mathsf{p} \in (E_1 \sqcap E_2)^\mathcal{I}$, thus $\mathsf{p} \in E^\mathcal{I}$.

3. Assume $E = E_1 \sqcup E_2$. We have that $E_1 \sqcup E_2 \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is saturated, then $\{E_1, E_2\} \cap \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p})) \neq \emptyset$. Therefore, $E_1 \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$ or $E_2 \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. By induction hypothesis, $\mathsf{p} \in E_1^\mathcal{I}$ or $\mathsf{p} \in E_2^\mathcal{I}$, which implies $\mathsf{p} \in (E_1 \sqcup E_2)^\mathcal{I}$, thus $\mathsf{p} \in E^\mathcal{I}$.

4. Assume $E = \forall R.D$. We have $\forall R.D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. Let $\mathsf{p}' \in \Delta^\mathcal{I}$ such that $(\mathsf{p}, \mathsf{p}') \in R^\mathcal{I}$. We have to prove that $\mathsf{p}' \in D^\mathcal{I}$. Since $(\mathsf{p}, \mathsf{p}') \in R^\mathcal{I}$, by the definition of $\mathcal{I}$, there are three possibilities :

   (a) There exist $a, b \in \mathbf{I}$ and $\tau \in \sigma_a$ such that $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}_a, \mathsf{p}_b)$, $\sigma_b \in \Omega(\sigma_a, \tau)$ and $R \in \mathsf{tie}(\tau)$.

   (b) There exist $\sigma \in \mathbf{\Lambda}$ and $\tau \in \mathsf{last}(\mathsf{p})$ such that σ is non blocked, $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}, \mathsf{p} \cdot \sigma)$, $\sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$ and $R \in \mathsf{tie}(\tau)$.

   (c) There exist $\sigma \in \mathbf{\Lambda}$ and $\tau \in \mathsf{last}(\mathsf{p})$ such that σ is blocked, $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}, \mathsf{p} \cdot \mathsf{b}(\sigma))$, $\sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$ and $R \in \mathsf{tie}(\tau)$.

   Consider the case (b). Since $\mathsf{last}(\mathsf{p})$ is saturated, $\forall R.D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$ and $R \in \mathsf{tie}(\tau)$, then $D \in \mathsf{tail}_\mathsf{C}(\tau)$. Now, since $D \in \mathsf{tail}_\mathsf{C}(\tau)$ and $\sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$, by (1) of Definition 28, $D \in \mathsf{core}_\mathsf{C}(\sigma)$. Therefore, $D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^\mathcal{I}$. This ends the proof in this case. The proof in the case (a) is similar. Consider (c). We can proceed like in (b) to obtain $D \in \mathsf{core}_\mathsf{C}(\sigma)$. Since σ is blocked, then $\mathsf{core}_\mathsf{C}(\sigma) \subseteq \mathsf{core}_\mathsf{C}(\mathsf{b}(\sigma))$. Therefore, $D \in \mathsf{core}_\mathsf{C}(\sigma)$, thus $D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^\mathcal{I}$.

5. Assume $E = \exists R.D$. We have $\exists R.D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))$. We have to prove that $\mathsf{p} \in (\exists R.D)^\mathcal{I}$, i.e. there exists $\mathsf{p}' \in \Delta^\mathcal{I}$ such that $\mathsf{p}' \in D^\mathcal{I}$ and $(\mathsf{p}, \mathsf{p}') \in R^\mathcal{I}$. Since $\mathsf{last}(\mathsf{p})$ is saturated, there exists a triple $\tau \in \mathsf{last}(\mathsf{p})$ such that $R \in \mathsf{tie}(\tau)$ and $D \in \mathsf{tail}_\mathsf{C}(\tau)$. Then, τ is not dummy, and, thus, $\Omega(\mathsf{last}(\mathsf{p}), \tau)$ is not empty, i.e. there exists $\sigma \in \mathbf{\Lambda}$ such that $\sigma \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$. By the definition of Ω, $\mathsf{tail}_\mathsf{C}(\tau) = \mathsf{core}_\mathsf{C}(\sigma)$. Thus, $D \in \mathsf{core}_\mathsf{C}(\sigma)$. There are two cases: σ is blocked or not. Assume that σ is not blocked and let $\mathsf{p}' = \mathsf{p} \cdot \langle \sigma \rangle$. By the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}') \in R^\mathcal{I}$. Also, $D \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, it follows $\mathsf{p}' \in D^\mathcal{I}$. Therefore, $\mathsf{p} \in (\exists R.D)^\mathcal{I}$. Assume that σ is blocked and

let $\mathsf{p}' = \mathsf{p} \cdot \langle \mathsf{b}(\sigma) \rangle$. By the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. Since $\mathsf{core}_{\mathsf{C}}(\sigma) \subseteq \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$, it follows $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$, and, thus, $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. Therefore, $\mathsf{p} \in (\exists R.D)^{\mathcal{I}}$.

6. Assume $E = \neg D$. We have $\neg D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Assume that D is a concept name. Since $\mathsf{last}(\mathsf{p})$ is clash-free, then $D \notin \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$, and, by the definition of $\mathcal{I}$, it follows $\mathsf{p} \notin D^{\mathcal{I}}$, i.e. $\mathsf{p} \in (\neg D)^{\mathcal{I}}$. If D is not a concept name, we can proceed by induction on the length of D to prove that $\mathsf{p} \in (\neg D)^{\mathcal{I}}$.

□

6.5.2 Completeness

Lemma 15 (Completeness)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with $\mathbf{I}$ a set of individuals. If $\mathcal{O}$ is consistent, then Algorithm 9 answers* Yes *with $\mathcal{O}$ as input.*

Proof. Assume that there is a model $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ of $\mathcal{O}$. First, we prove that Algorithm 9 answers Yes if it builds successfully a pre-compressed tableau $\langle \mathbf{\Lambda}, \Omega \rangle$ which involves a compressed tableau. Since $\langle \mathbf{\Lambda}, \Omega \rangle$ evolves monotonically, if there is a compressed tableau involved in $\langle \mathbf{\Lambda}, \Omega \rangle$ at a moment, then Algorithm 8 can detect it whenever Algorithm 8 is called later according to Lemma 13. Moreover, Algorithm 8 is called by each iteration of the loop in Algorithm 9, and there is at least one iteration of the loop executed. This implies that if there is a compressed tableau, then it is detected by Algorithm 8 which makes Algorithm 9 return Yes. It remains to prove that Algorithm 9 can build a compressed tableau involved in $\langle \mathbf{\Lambda}, \Omega \rangle$.

For this purpose, we use Lemmas 10, 11, 12 to guarantee that $\langle \mathbf{\Lambda}, \Omega \rangle$ preserves pre-compressed tableau structure when it is expanded by applying rules in Figures 6.2 and 6.3. In addition, we need a function π that associates a non-empty subset $X \subseteq \Delta^{\mathcal{I}}$ to some star-type σ in $\langle \mathbf{\Lambda}, \Omega \rangle$, i.e. $\pi(\sigma) = X$. This function allows to maintain clash-freeness of the star-types of the domain of π when expanding $\langle \mathbf{\Lambda}, \Omega \rangle$ by rules. For that, π needs to preserve the following conditions when applying a rule:

$$\text{For each } \sigma \in \mathbf{\Lambda}, \text{ if } C \in \mathsf{core}_{\mathsf{C}}(\sigma) \text{ then } \pi(\sigma) \subseteq C^{\mathcal{I}} \tag{6.1}$$

$$\text{For each } \sigma \in \mathbf{\Lambda}, \text{ if } a, b \in \mathsf{core}_{\mathsf{I}}(\sigma) \cap \mathbf{I} \text{ then } \{a^{\mathcal{I}}\} = \{b^{\mathcal{I}}\} = \pi(\sigma) \tag{6.2}$$

$$\text{If } a \not\approx b \in \mathcal{A}, \{a, b\} \cap \mathsf{core}_{\mathsf{I}}(\sigma) \neq \emptyset \text{ with } \sigma \in \mathbf{\Lambda}, \text{ then } \{a^{\mathcal{I}}, b^{\mathcal{I}}\} \not\subseteq \pi(\sigma) \tag{6.3}$$

$$\text{For each } \sigma, \sigma' \in \mathbf{\Lambda}, \text{ if } \sigma' \in \Omega(\sigma, \rho) \text{ and } R \in \mathsf{tie}(\rho) \text{ then } x \in \pi(\sigma), y \in \pi(\sigma') \text{ imply } \langle x, y \rangle \in R^{\mathcal{I}} \tag{6.4}$$

$$\text{For each } \sigma, \sigma' \in \mathbf{\Lambda} \text{ such that } \pi(\sigma) = \pi(\sigma'), a \in \mathsf{core}_{\mathsf{I}}(\sigma) \cap \mathsf{core}_{\mathsf{I}}(\sigma'), \text{ it holds that either } \mathsf{core}(\sigma) \subseteq \mathsf{core}(\sigma') \text{ or } \mathsf{core}(\sigma') \subseteq \mathsf{core}(\sigma) \tag{6.5}$$

Assume that such a function π exists. We show that execution of rules leads to obtaining a subset Σ of the domain of π which forms a compressed tableau.

We use $S = \langle \sigma_1, \ldots, \sigma_n \rangle$ to denote a sequence of star-types included in the domain of π such that σ_i is transformed into σ_{i+1} by applying a rule in Figures 6.2 and 6.3. We denote $\mathsf{head}(S) = \sigma_1$ if σ_1 is created from "scratch" by the algorithm (via Init or matchTriple), and $\mathsf{tail}(S) = \sigma_n$ if σ_n is transformed into a star-type with clash, or no rule which is applicable to

σ_n or leads to changing Ω of σ_n. Note that a star-type may be transformed into another star-type which exists already in the current $\langle \mathbf{\Lambda}, \Omega \rangle$. A sequence S is called *final* if no rule which is applicable to $\mathsf{tail}(S)$ or leads to changing Ω of $\mathsf{tail}(S)$, and *non final* otherwise. We use Σ to denote the set of $\mathsf{tail}(S)$ for all final S. Note that Σ is well defined since $\mathsf{tail}(S)$ always exists for all final sequence S due to termination of Algorithm 9 and non-applicability of the rules to the whole Σ. We prove the following properties.

$$\text{For each } \sigma \in \Sigma, \sigma \text{ is clash-free} \tag{6.6}$$

$$\text{For each } \sigma \in \Sigma \text{ and non dummy } \rho \in \sigma, \text{ there is some } \sigma' \in \Sigma \text{ s.t. } \sigma' \in \Omega(\sigma, \rho) \tag{6.7}$$

$$\text{For each } a \in \mathbf{I}, \text{ there is a final sequence } S \text{ such that } a \in \mathsf{core_I}(\mathsf{head}(S)) \text{ and } a \in \mathsf{core_I}(\mathsf{tail}(S)) \tag{6.8}$$

$$\text{For each } a \in \mathbf{I}, \text{ there is a unique } \sigma \in \Sigma \text{ such that } a \in \mathsf{core_I}(\sigma) \tag{6.9}$$

We show Property (6.6). Let $\sigma \in \Sigma$. Due to Condition (6.1), it is not possible that $\{A, \neg A\} \subseteq \mathsf{core}_C(\sigma)$ since otherwise $\pi(\sigma) \subseteq A^{\mathcal{I}}$, $\pi(\sigma) \subseteq \neg A^{\mathcal{I}}$ and $\pi(\sigma) \neq \emptyset$. Assume that $\sigma \in \Sigma$ and $a, b \in \mathsf{core_I}(\sigma)$. Due to Condition (6.2) we have $a^{\mathcal{I}} = b^{\mathcal{I}} \in \pi(\sigma)$. If $a \not\approx b \in \mathcal{A}$ then $\{a^{\mathcal{I}}, b^{\mathcal{I}}\} \not\subseteq \pi(\sigma)$ due to Condition (6.3). Therefore, σ is clash-free.

We show Property (6.7). Let $\sigma \in \Sigma$. By Lemmas 10, 11 and 12, for each non dummy $\rho \in \sigma$ it holds that $\Omega(\sigma, \rho) \neq \emptyset$. Let $\omega_1 \in \Omega(\sigma, \rho)$ such that there is some rule which is applicable to ω_1. Due to Lemmas 10, 11 and 12 and Property (6.6), ω_1 is transformed into ω_2 without clash and $\omega_2 \in \Omega(\sigma, \rho)$. By applying the same argument for ω_i with $i \geq 2$, we obtain a final sequence $S = \langle \cdots, \omega_1, \cdots, \omega_n \rangle$ such that $\omega_n \in \Omega(\sigma, \rho)$. Hence, Property (6.7) is proved.

We now show Property (6.8). For each individual $a \in \mathbf{I}$, Algorithm 4 generates a star-type σ_a with $a \in \mathsf{core_I}(\sigma_a)$. We initialize S to $\langle \sigma_a \rangle$. Let $\sigma_i \in S$ such that $a \in \mathsf{core_I}(\sigma_i)$, σ_i is clash-free and $a^{\mathcal{I}} \in \pi(\sigma_i)$. If there is some rule is applicable to ω_i, then ω_i is transformed into ω_{i+1} such that $a \in \mathsf{core_I}(\sigma_{i+1})$ (monotonicity). We add σ_{i+1} to S. This implies that there is a sequence S that is final and $a \in \mathsf{core_I}(\mathsf{tail}(S))$.

Let's prove Property (6.9). Let $S = \langle \sigma_1, \cdots, \sigma_n \rangle$ be a sequence of star-types as defined above with $a \in \mathsf{core_I}(\sigma_i)$ for all i. We show that every rule transforms σ_i to a unique σ_{i+1}. Indeed, the non-deterministic rule such as **Rule** $\rightarrow_{\sqcup}$, **Rule** $\rightarrow_{\mathbf{chLK}_1}$, **Rule** $\rightarrow_{\mathbf{chLK}_2}$ can lead to transforming σ_i to 2 different star-types σ_{i+1} and σ'_{i+1} with $C_1 \in \mathsf{core}(\sigma_{i+1})$, $C_2 \in \mathsf{core}(\sigma'_{i+1})$ and $C_1 \neq C_2$. However, this is not possible due to Condition (6.5). This implies that S is unique.

The merging rules **Rule** $\rightarrow_{\mathbf{LK}}$ and **Rule** $\rightarrow_{\approx}$ can lead to transforming σ_i to 2 different star-types σ_{i+1} and σ'_{i+1}, with $\mathsf{core}(\sigma_{i+1}) = \mathsf{core}(\sigma'_{i+1})$. Since $\mathsf{core}(\sigma_{i+1}) = \mathsf{core}(\sigma'_{i+1})$, according to Item 2b of Definition 26, and due to the non-applicability of rules and the definition of merging, we obtain that $\sigma_{i+1} = \sigma'_{i+1}$.

We now show that Σ forms a compressed tableau.

1. *Clash-freeness*. This a consequence of Property (6.6).

2. *Non-applicability of the rules in Figures 6.2 and 6.3*. For the rules in Figure 6.2, non-applicability is due to the definition of Σ. For **Rule** $\rightarrow_{\mathbf{chLK}_i}$, $i \in \{1, 2\}$ and **Rule** $\rightarrow_{\mathbf{LK}}$ in Figure 6.3, if all star-types involved in a link key λ are contained in Σ and λ is applicable, then Algorithm 9 does not terminate. Thus, non-applicability of the rules in Figure 6.3 over Σ is ensured as well. The same argument can be used for **Rule** $\rightarrow_{\approx}$.

3. *Interpretation of individuals*. We show that for each $a \in \mathbf{I}$, there is a unique $\sigma \in \Sigma$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. Indeed, for each individual $a \in \mathbf{I}$ Algorithm 4 generates a star-type σ with $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. Due to Property (6.8), there is a sequence S such that $\mathsf{head}(S) = \sigma$ and $\mathsf{tail}(S) \in \Sigma$. Since Algorithm 9 never removes anything, we have $a \in \mathsf{core}_{\mathsf{I}}(\mathsf{tail}(S))$. Due to Property (6.9), $\mathsf{tail}(S)$ is unique in Σ.

4. *Satisfaction of concept assertions*. We show that for each assertion $C(a) \in \mathcal{A}$ there is a star-type $\sigma \in \Sigma$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$ and $C \in \mathsf{core}_{\mathsf{C}}(\sigma)$. Indeed, for each assertion $C(a)$ Algorithm 4 generates a star-type σ with $C \in \mathsf{core}_{\mathsf{C}}(\sigma)$ and $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. Due to Property (6.8), there is a sequence S such that $\mathsf{head}(S) = \sigma$ and $\mathsf{tail}(S) \in \Sigma$. Since Algorithm 9 never removes anything, we have $a \in \mathsf{core}_{\mathsf{I}}(\mathsf{tail}(S))$ and $a \in \mathsf{core}_{\mathsf{C}}(\mathsf{tail}(S))$.

5. *Satisfaction of role assertions*. We show that for each assertion $R(a, b) \in \mathcal{A}$ there are two star-types $\sigma, \sigma' \in \Sigma$ such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$, $b \in \mathsf{core}_{\mathsf{I}}(\sigma')$ and σ' is an R-successor of σ. Indeed, for each assertion $R(a, b)$ Algorithm 4 generates a star-type ω with $\nu \in \omega$ such that $a \in \mathsf{core}_{\mathsf{I}}(\omega)$, $b \in \mathsf{tail}_{\mathsf{I}}(\nu)$ and $R \in \mathsf{tie}(\nu)$. Due to Property (6.8), there is a final sequence S such that $\omega = \mathsf{head}(S)$, $\mathsf{tail}(S) \in \Sigma$ with $a \in \mathsf{core}_{\mathsf{I}}(\mathsf{tail}(S))$, $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}_{\mathsf{I}}(\rho)$ for some $\rho \in \mathsf{tail}(S)$. Due to Property (6.7), there is some $\sigma' \in \Sigma$ such that $\sigma' \in \Omega(\mathsf{tail}(S), \rho)$. By the definition of Ω, we have $b \in \mathsf{core}_{\mathsf{I}}(\sigma')$.

6. We show Condition 4 in Definition 32. Assume there is a link key λ involves σ_1, σ_2 and other star-types such that all of them are contained in Σ. Due to the definition of link keys and the absence of inverse roles, we have σ_1 and σ_2 must be nominal with $a \in \mathsf{core}_{\mathsf{I}}(\sigma_1)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_2)$ for some individuals a, b. Due to non-applicability of **Rule** $\rightarrow_{\mathbf{LK}}$ over Σ,

   there is a nominal star-type ω such that $\mathsf{core}(\sigma_1) \subseteq \mathsf{core}(\omega)$ and $\mathsf{core}(\sigma_2) \subseteq \mathsf{core}(\omega)$. If $\omega \in \Sigma$ then, by Property (6.9) we have $\sigma_1{=}\sigma_2{=}\omega$. Assume $\omega \notin \Sigma$. There are 3 sequences S_{σ_1}, S_{σ_2} and S_ω containing σ_1, σ_2 and ω respectively such that $\mathsf{tail}(S_{\sigma_1}), \mathsf{tail}(S_{\sigma_2}), \mathsf{tail}(S_\omega) \in \Sigma$ by definition. Due to Property (6.9) we have $\mathsf{tail}(S_{\sigma_1}){=}\mathsf{tail}(S_{\sigma_2}){=}\mathsf{tail}(S_\omega){=}\sigma_1{=}\sigma_2$.

7. We show Condition 2 in Definition 32. Assume that $a \approx b \in \mathcal{A}$ and there are nominal σ_1 and σ_2 with $a \in \mathsf{core}_{\mathsf{I}}(\sigma_1)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_2)$. We can use the same argument for proving Condition 4.

8. Condition 3 in Definition 32 is a consequence of non-applicability of **Rule** $\rightarrow_{\mathbf{chLK_i}}$ over Σ.

9. *Well-definedness of matching function*. This a consequence of Property (6.7).

Before proving presevation of Conditions (6.1-6.5), we have to prove that these condition hold for the initial state. When Algorithm 4 creates a nominal star-type σ_a for each individual $a \in \mathbf{I}$, we define $\pi(\sigma_a) = \{a^{\mathcal{I}}\}$. Hence, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all $C(a) \in \mathcal{A}$ and Conditions (6.1-6.5) are satisfied since $\mathcal{I}$ is a model. If Algorithm 4 adds a non dummy triple ρ to σ_a by Line 20 for some $R(a, b) \in \mathcal{A}$ with $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}_{\mathsf{I}}(\rho)$, then $\sigma_b \in \Omega(\sigma_a, \rho)$. Let $x \in \pi(\sigma_a)$ and $y \in \pi(\sigma_b)$. By the definition of π, we have $x = a^{\mathcal{I}}$ and $y = b^{\mathcal{I}}$. Since $\mathcal{I}$ is a model we have $\langle x, y\rangle = \langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$. This implies that Conditions (6.4) is also satisfied.

We now show that Conditions (6.1-6.5) above are preserved when extending π after each application of rules.

1. **Rule** $\rightarrow_{\sqcap}$ is applied to $\sigma \in \Lambda_i$ due to $C \sqcap D \in \mathsf{core}_{\mathsf{C}}(\sigma)$. According to the rule and Lemma 10, a copy σ' of σ with $C, D \in \mathsf{core}_{\mathsf{C}}(\sigma')$ is created and added to Λ_i. We define $\pi(\sigma)$=$\pi(\sigma')$, and $\pi(\omega)$=$\pi(\omega')$ for each neighbor ω of σ which is changed to ω'.

   (a) We show that Condition (6.1) is preserved. We have $C \sqcap D \in \mathsf{core}_{\mathsf{C}}(\sigma) \subseteq \pi(\sigma)$, and thus $\pi(\sigma) \subseteq (C \sqcap D)^{\mathcal{I}}$ by Condition (6.1). Since $\mathcal{I}$ is a model, we have $(C \sqcap D)^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and $(C \sqcap D)^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. This implies that $\pi(\sigma) \subseteq C^{\mathcal{I}}$ and $\pi(\sigma) \subseteq D^{\mathcal{I}}$. By definition, we have $\pi(\sigma)$=$\pi(\sigma')$, and thus $\pi(\sigma') \subseteq C^{\mathcal{I}}$ and $\pi(\sigma') \subseteq D^{\mathcal{I}}$.

   (b) Conditions (6.2) and (6.4) are preserved since the algorithm never removes anything and $\pi(\sigma)$=$\pi(\sigma')$.

   (c) Condition (6.3) is preserved since this rule does not add or remove any individual and $\pi(\sigma)$=$\pi(\sigma')$.

   (d) We show that Condition (6.5) is preserved. Let ω be a star-type such that $\pi(\omega)$ is defined. Assume $\mathsf{core}(\omega) \subseteq \mathsf{core}(\sigma)$. We have $\mathsf{core}(\omega) \subseteq \mathsf{core}(\sigma) \subseteq \mathsf{core}(\sigma')$. Assume $\mathsf{core}(\sigma) \subset \mathsf{core}(\omega)$. This implies that there are two different nominal sequences S=$\langle \sigma_1, \cdots, \sigma_k \rangle$ and S'=$\langle \omega_1, \cdots, \omega_{k'} \rangle$ such that σ_1=ω_1, σ_k=σ and $\omega_{k'}$=ω. Hence, there is some $\mathsf{max}(k, k') \geq h > 1$ such that $\mathsf{core}(\omega_j)$=σ_j with $j \leq h$ and $\mathsf{core}(\sigma_{h+1}) \not\subseteq \omega_{h+1}$ and $\mathsf{core}(\omega_{h+1}) \not\subseteq \sigma_{h+1}$, which is caused by a non-deterministic rule. This contradicts Condition (6.5). Therefore, $\mathsf{core}(\sigma) \subseteq \mathsf{core}(\omega)$ implies $\mathsf{core}(\sigma)$=$\mathsf{core}(\omega)$, and thus we obtain $\mathsf{core}(\omega) \subseteq \mathsf{core}(\sigma) \subseteq \mathsf{core}(\sigma')$.

2. **Rule** $\rightarrow_{\sqcup}$ is applied to $\sigma \in \Lambda_i$ due to $C_1 \sqcup C_2 \in \mathsf{core}_C(\sigma)$. According to the rule and Lemma 10, two copies σ_1 and σ_2 of σ with $C_j \in \mathsf{core}_C(\sigma_j)$ is created and added to Λ_i with $1 \leq j \leq 2$. Due to Condition 6.1, we have $\pi(\sigma) \subseteq (C_1 \sqcup C_2)^{\mathcal{I}}$. Since $\mathcal{I}$ is a model, we have $(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$. Since $\pi(\sigma) \neq \emptyset$ we have $\pi(\sigma) \cap C_j^{\mathcal{I}} \neq \emptyset$ for some $1 \leq j \leq 2$. Thus, we define $\pi(\sigma_j) = (\pi(\sigma) \cap C_j^{\mathcal{I}}) \cup \{a^{\mathcal{I}}\}$ if $\pi(\sigma) \cap C_j^{\mathcal{I}} \neq \emptyset$ for some $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$. We have $\pi(\sigma_j) \subseteq \pi(\sigma)$ since $a^{\mathcal{I}} \in \pi(\sigma)$ and $\pi(\sigma) \cap C_j^{\mathcal{I}} \subseteq \pi(\sigma)$.

   (a) We show that Condition (6.1) is preserved. Assume $D \in \mathsf{core}_{\mathsf{C}}(\sigma_j)$. If $D \in \mathsf{core}_{\mathsf{C}}(\sigma)$ then $\pi(\sigma_j) \subseteq \pi(\sigma) \subseteq D^{\mathcal{I}}$ due to Condition (6.1). If D=C_j then $\pi(\sigma_j)$=$\pi(\sigma) \cap C_j^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.

   (b) We show that Condition (6.2) is preserved. Let $a, b \in \mathsf{core}_{\mathsf{I}}(\sigma_j)$. This implies that $a, b \in \mathsf{core}_{\mathsf{I}}(\sigma)$ since $\mathsf{core}_{\mathsf{I}}(\sigma_j)$=$\mathsf{core}_{\mathsf{I}}(\sigma)$. Due to Condition (6.2), we have $a^{\mathcal{I}}$=$b^{\mathcal{I}} \in \pi(\sigma)$. By definition, $a^{\mathcal{I}}$=$b^{\mathcal{I}} \in \pi(\sigma_j)$.

   (c) Condition (6.4) is preserved because $\pi(\sigma_j) \subseteq \pi(\sigma)$.

   (d) We show that Condition (6.3) is preserved. Assume $\{a, b\} \cap \mathsf{core}_{\mathsf{I}}(\sigma_j) \neq \emptyset$. This implies that $\{a, b\} \cap \mathsf{core}_{\mathsf{I}}(\sigma) \neq \emptyset$ since $\mathsf{core}_{\mathsf{I}}(\sigma_j) = \mathsf{core}_{\mathsf{I}}(\sigma)$. Due to Condition (6.2), we have $\{a, b\} \not\subseteq \pi(\sigma)$. By definition, we have $\pi(\sigma_j) \subseteq \pi(\sigma)$. This implies that $\{a, b\} \not\subseteq \pi(\sigma_j)$.

   (e) Note that π is defined for one star-type σ_j with some $1 \leq j \leq 2$. Hence, to show Condition (6.5), we can use the same argument for the case of **Rule** $\rightarrow_{\sqcap}$.

3. **Rule** $\rightarrow_{\exists}$ is applied to $\sigma \in \Lambda_i$ due to $\exists R.C \in \mathsf{core}_C(\sigma)$ (i.e. σ is not blocked). The rule creates a copy σ' of σ, and adds new triple ρ to σ' such that $\mathsf{tie}(\rho) = \{R\}$ and

$\mathsf{tail}(\rho) = \{C\}$. Lemma 11 affirms that Algorithm 6 adds to $\mathbf{\Lambda}$ the created star-type σ' and an ρ-successor ω of σ'. We define $\pi(\sigma') = \pi(\sigma)$. If ω is not defined yet, we define $\pi(\omega) = C^{\mathcal{I}}$. Since $\mathcal{I}$ is a model, we have $\pi(\omega) = C^{\mathcal{I}} \neq \emptyset$. Thus, Conditions 6.1, 6.2, 6.4 and 6.3 are satisfied for the extension $\pi(\sigma') = \pi(\sigma)$. They are also satisfied for the extension $\pi(\omega)$ since $\pi(\omega) = C^{\mathcal{I}}$ and $\pi(\omega)$ is fresh. For Condition (6.5), we can use the same argument.

4. **Rule** $\rightarrow_{\forall}$ is applied to $\sigma \in \Lambda_i$ due to $\forall R.C \in \mathsf{core}_C(\sigma)$ and a triple $\rho \in \sigma$ with $R \in \mathsf{tie}(\rho)$. The rule creates a copy σ' of σ, a copy ρ' of ρ, adds C to $\mathsf{tail}(\rho')$ and replaces ρ by ρ' in σ'. Then, Algorithm 6 adds σ' to $\mathbf{\Lambda}$. By Lemma 11, for each ρ-successor ω of σ, the algorithm creates also a copy σ' of σ, performs the change $\mathsf{core}(\omega') = \mathsf{tail}(\rho')$ and adds σ' to $\mathbf{\Lambda}$ such that ω' is an ρ'-successor of σ'. We define $\pi(\sigma') = \pi(\sigma)$ and $\pi(\omega') = \pi(\omega)$.

   By definition, $\pi(\sigma) \neq \emptyset$, $\pi(\omega) \neq \emptyset$. Thus, there are $x \in \pi(\sigma)$ and $y \in \pi(\omega)$. Due to Condition (6.4), we have $\langle x, y\rangle \in R^{\mathcal{I}}$ for all $x \in \pi(\sigma)$ and $y \in \pi(\omega)$. Due to Condition (6.1), we have $x \in (\forall R.C)^{\mathcal{I}}$. Since $\mathcal{I}$ is a model, we have $y \in C^{\mathcal{I}}$. Hence, $\pi(\omega) \subseteq C^{\mathcal{I}}$. This implies that Condition (6.1) is preserved. The other conditions are preserved since $\pi(\sigma') = \pi(\sigma)$ and $\pi(\omega') = \pi(\omega)$. For Condition (6.5), we can use the same argument for the case of **Rule** $\rightarrow_{\sqcap}$.

5. **Rule** $\rightarrow_{\approx}$ is applied to $\sigma_1, \sigma_2 \in \Lambda$ due to $a \in \mathsf{core}_{\mathsf{I}}(\sigma_1)$, $b \in \mathsf{core}_{\mathsf{I}}(\sigma_2)$ and $a \approx b \in \mathcal{A}$, and thus $\sigma_1, \sigma_2 \in \Lambda_0$. The rule creates a set $\sigma_1 \oplus \sigma_2$ containing two star-type σ_0 and σ_0' such that $\mathsf{core}(\sigma_0)$=$\mathsf{core}(\sigma_0')$=$\mathsf{core}(\sigma_1) \cup \mathsf{core}(\sigma_2)$ with triples and Algorithm 7 adds them to $\mathbf{\Lambda}$ such that each successor of $\sigma_i, i \in \{1, 2\}$ is a successor of $\tau \in \{\sigma_0, \sigma_0'\}$. Moreover, for each $\tau \in \{\sigma_0, \sigma_0'\}$, Algorithm 7 creates a copy ω' of each ρ-predecessor ω of σ_i, creates a copy ρ' of ρ, sets $\mathsf{tail}(\rho')$=$\mathsf{core}(\tau)$, replaces ρ by ρ' in ω' and adds it to $\mathbf{\Lambda}$ such that τ is a ρ'-successor of ω' due to Lemma 12. We now define $\pi(\tau) = \pi(\sigma_1) \cup \pi(\sigma_2)$ and $\pi(\omega') = \pi(\omega)$. Analogously, we can check all Conditions (6.1), (6.2), (6.3) and (6.4) . For Condition (6.5), we can use the same argument for the case of **Rule** $\rightarrow_{\sqcap}$.

6. **Rule** $\rightarrow_{\mathbf{chLK}_1}$ is applied to $\sigma_1, \sigma_2, \cdots, \sigma_n \in \Lambda$ and change σ_1, σ_2. The condition of link key implies that $\sigma_1, \sigma_2 \in \Lambda_0$.

   This rule creates 2 copies σ_1 and σ_2 of σ, and add C to $\mathsf{core}(\sigma_1)$ and $\sim C$ to $\mathsf{core}(\sigma_2)$. Then, we define $\pi(\sigma_j) = (\pi(\sigma) \cap C_j^{\mathcal{I}}) \cup \{a^{\mathcal{I}}\}$ if $\pi(\sigma) \cap C_j^{\mathcal{I}} \neq \emptyset$ for some $a \in \mathsf{core}_{\mathsf{I}}(\sigma)$ where $C_j = C$ and $C_j = \sim C$. We can use the same arguments as those for **Rule** $\rightarrow_{\sqcup}$ to show that all conditions are preserved. Note that π is defined for one star-type σ_j with some $1 \leq j \leq 2$. For Condition (6.5), we can use the same argument for the case of **Rule** $\rightarrow_{\sqcap}$.

7. **Rule** $\rightarrow_{\mathbf{chLK}_2}$ is applied to $\sigma_1, \sigma_2, \cdots, \sigma_n \in \Lambda$ and change σ_1, σ_2. Analogously.

8. **Rule** $\rightarrow_{\mathbf{LK}}$ is applied to $\sigma_1, \sigma_2, \cdots, \sigma_n \in \Lambda$ and change σ_1, σ_2 with a link key λ. The condition of link key implies that $\sigma_1, \sigma_2 \in \Lambda_0$. We can use the same argument as given for **Rule** $\rightarrow_{\approx}$.

□

6.5.3 Complexity

This subsection analyzes and provides the complexity of Algorithm 9. Lemmas 16 is fundamental for analyzing the complexity of Algorithm 4, 5, 6, 7, 8, and 9.

Lemma 16 (Size of a compressed tableau)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology. Let $\langle \mathbf{\Lambda}, \Omega \rangle$ be a pre-compressed tableau for $\mathcal{O}$ with $\mathbf{\Lambda}$=$\langle \Lambda_0, \cdots, \Lambda_n \rangle$. The size of $\langle \mathbf{\Lambda}, \Omega \rangle$ is bounded by an exponential function in the size of $\mathcal{O}$.*

Proof. We denote by ℓ the size of $\mathcal{O}$. By construction, $\mathsf{sub}(\mathcal{O})$ is the set of all sub-concepts occurring in $\mathcal{O}$ and their negated form. It holds that if a concept is represented as a string then a sub-concept is represented as a substring. Then the size of $\mathsf{sub}(\mathcal{O})$, denoted by c is bounded by $2 \times (\ell \times (\ell + 1)/2)$, $c \leq 2 \times \ell \times (\ell + 1)/2 \leq c_1 \ell^2$ where c_1 is a constant large enough.

In addition, since the size of a concept or role is bounded by ℓ, the size of a triple, denoted $\|\rho\|$, is bounded by $(2c\ell + \ell)$. Hence, $\|\rho\| \leq c_2 \ell^3$ with $c_2 = 2c_1 + 1$. We now calculate the number of triples in a star-type σ where $\sigma \in \mathbf{\Lambda}$. For every triple $\rho \in \sigma$, ρ is added to σ by init algorithm due to a role assertion in $\mathcal{A}$ or by $\rightarrow_\exists$-rule due to an existential concept in $\mathsf{core}_\mathsf{C}(\sigma)$.

This is also the case for star-types generated by the merging operation defined in Definition 26. Suppose that there exists a triple $\rho \in \omega_i$, if $\mathsf{tail}_\mathsf{I}(\rho) = \emptyset$, this means that there exists a triple $\rho' \in \sigma_i$ with $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$ and $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$ (by Item 2a of Definition 26). Thus ρ' is added to σ_i by $\rightarrow_\exists$-rule, thus ρ' is the unique triple in σ_i satisfying a concept assertion in $\mathsf{core}(\sigma_i)$. As a result, ρ is the unique triple in ω_i satisfying the same concept assertion in $\mathsf{core}(\omega_i)$.

Now, suppose that there exists a triple $\rho \in \omega_i$, such that $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}(\rho)$ ($\mathsf{tail}_\mathsf{I}(\rho) \neq \emptyset$), this means that there is some triple $\rho' \in \sigma_i$ with $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$ and $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$ or $\rho' \in \sigma_j$ with $\mathsf{tie}(\rho') = \mathsf{tie}(\rho)$ and $\mathsf{tail}(\rho') = \mathsf{tail}(\rho)$, $1 \leq i \neq j \leq 2$ and there is no $\rho'' \in \sigma_i$ with $a \in \mathsf{head}_\mathsf{I}(\rho'')$, $R \in \mathsf{tie}(\rho'')$, $b \in \mathsf{tail}_\mathsf{I}(\rho'')$. We get that for each role assertion $R(a, b) \in \mathcal{A}$ there is a unique triple $\rho \in \omega_\mathsf{i}$ where $R \in \mathsf{tie}(\rho)$ and $b \in \mathsf{tail}(\rho)$.

We get that the number of triples in a star-type σ is bounded by the number of role assertions in $\mathcal{A}$ and existential concepts in $\mathsf{core}(\sigma)$. We get that the number of triples in σ is bounded by $c_1 \ell^2 + \ell$.

It follows that the size of a star-type, denoted by $\|\sigma\|$, is bounded by $(c_1 \ell^2 + \ell) \times \|\rho\| \leq c_3 \ell^5$ with c_3 a constant large enough.

To calculate the size of $\langle \mathbf{\Lambda}, \Omega \rangle$ we have to first calculate the size of each layer $\Lambda_i, 0 \leq i \leq n$ and then the number of layers in $\mathbf{\Lambda}$. We now determine the size of each layer Λ_i, denoted $|\Lambda_i|$, with $0 \leq i \leq n$. Since each star-type has at most $c_1 \ell^2 + \ell$ triples, we obtain $|\Lambda_i| \leq 2^{c_1 \ell^2 + \ell}$. Moreover, any star-type in a layer Λ_k with $k = 2^{c_1 \ell^2}$ must be blocked since, otherwise, there must exist k star-types whose cores are different, which is not possible. This implies that $n \leq 2^{c_1 \ell^2}$ where n is the number of layers. Hence, the size of $\mathbf{\Lambda}$ is bounded by $n \times \|\Lambda_i\| \times \|\sigma\| \leq 2^{c_1 \ell^2} \times 2^{c_1 \ell^2 + \ell} \times c_3 \ell^5 \leq 2^{C \ell^2}$ for C a constant large enough. This ends the proof. □

Lemma 17 (Complexity)**.** *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALC}$+$\mathcal{LK}$ ontology with a set of individuals* **I***. Algorithm 9 runs in exponential time in the size of $\mathcal{O}$.*

Proof. Assume that $\langle \mathbf{\Lambda} = \langle \Lambda_i \rangle_{i=0}^n, \Omega \rangle$ is the pre-compressed tableau built by Algorithm 9. We denote by ℓ the size of $\mathcal{O}$, i.e., $\|\mathcal{O}\| = \ell$. We analyze the complexity of each step.

1. Line 1 of the algorithm calls init which according to Item 2 of Lemma 9 runs in polynomial time in ℓ.

2. Line 5 calls check which according to Item 2 of Lemma 13 runs in polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

3. Line 8 traverses each layer in $\Lambda_i \in \mathbf{\Lambda}$ and each star-type σ in $\Lambda_i, i \geq 0$ such that there exists a rule r in Figure 6.2 applied on σ and σ is not in $\mathsf{processed}_{\mathsf{ALC}}$. Checking if there is a rule r in Figure 6.2 necessitates only to traverse the components of σ, the set of GCIs in $\mathcal{T}$. Checking if σ belongs to $\mathsf{processed}_{\mathsf{ALC}}$ is bounded by the size of $\mathsf{processed}_{\mathsf{ALC}}$ which is bounded by the number of star-types in $\mathbf{\Lambda}$. The application of r on σ in Line 10 of the algorithm involves calling matchCore algorithm once if r is $\rightarrow_{\sqsubseteq}$ rule or $\rightarrow_{\sqcap}$-rule and twice if r is $\rightarrow_{\sqcup}$-rule and calling once and matchTriple algorithm if r is $\rightarrow_{\forall}$ or $\rightarrow_{\exists}$-rule. According to Item 3 of Lemma 10, Item 3 of Lemma 11 and Item 3 of Lemma 12, each of these algorithms runs in polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

   Then the complexity of the loop between Lines 8 and 12 is bounded by a polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$.

   • Lines 13 traverses each set $\{\omega_1, \ldots, \omega_n\}$ where $\omega_i \in \Lambda_0$ and n is the greatest number of star-types involved in a link key and $\{\omega_1, \ldots, \omega_n\}$ is not in $\mathsf{processed}_{\mathsf{LK}}$. The number of different sets of star-types of size n in Λ_0 and the size of $\mathsf{processed}_{\mathsf{LK}}$ is bounded by $|\Lambda_0|^n$.

   So the complexity of Lines 13 is bounded by $|\Lambda_0|^n \times |\Lambda_0|^n$. Line 14 checks if their exists a rule r in Figure 6.3 applicable on $\{\omega_1, \ldots, \omega_n\}$. Then its complexity is bounded by $2^n(|\mathcal{LK}|+|\mathcal{A}|)$. The application of r on $\{\omega_1, \ldots, \omega_n\}$ in Line 15 of the algorithm involves calling twice matchCore if r is $\rightarrow_{\mathbf{chLK}_1}$ or $\rightarrow_{\mathbf{chLK}_2}$ and once matchMerge if r is $\rightarrow_{\mathbf{LK}}$ or $\rightarrow_{\approx}$. Each of these algorithms runs in polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. Then the complexity of the loop between Lines 13 and 19 is bounded by $|\Lambda_0|^{2n} \times 2^n(|\mathcal{LK}| + |\mathcal{A}|) \times f(|\mathbf{\Lambda}|)$ where $f(|\mathbf{\Lambda}|)$ is a polynomial function that bounds the complexity of matchCore and matchMerge. By Lemma 16, $|\mathbf{\Lambda}|$ and $|\Lambda_0|$ are bounded an exponential function in the size of $\mathcal{O}$. Moreover, the number of iterations of the loop at Line 3 is bounded $\mathsf{max}(|\mathsf{processed}_{\mathsf{ALC}}|, |\mathsf{processed}_{\mathsf{LK}}|) \leq \mathsf{max}(|\mathbf{\Lambda}|, |\Lambda_0|^n)$. Hence, it follows that Algorithm 9 runs in exponential time in the size of $\mathcal{O}$.

By Lemma 16 we have that the size of $\langle \mathbf{\Lambda}, \Omega \rangle$ is bounded by an exponential function in the size of $\mathcal{O}$. Since Algorithm 9 runs in polynomial time in the size of $\langle \mathbf{\Lambda}, \Omega \rangle$. It follows that Algorithm 9 runs in exponential time in the size of $\mathcal{O}$. □

6.6 Extending $\mathcal{ALC}+\mathcal{LK}$ with inverse roles

The compressed tableau algorithm given in Section 6.3 is inspired from the compressed tableau for the description logic $\mathcal{SHOIQ}$ [16]. This supports it extension for reasoning with link keys in more expressive description logics. In particular, extending $\mathcal{ALC}+\mathcal{LK}$ with inverse roles will increase the scope of applications of link key reasoning. Link keys containing inverse properties are relevant for interlinking a pair of datasets which uses opposite ways to express a fact. For example the link key $(\{\langle parent^{-}, enfant\rangle\}$ linkkey $\langle Person, Personne\rangle)$ is relevant

for linking a pair of datasets using the respectively the properties parent and enfant to express the relation between a child and his parent.

Adding inverse roles to $\mathcal{ALC}+\mathcal{LK}$ necessitates to express the bi-directional relations between individuals. This translates into modifying the matching between star-types in the context of compressed tableau. The first step for extending the compressed tableau algorithm with inverse roles is to change the definition of the matching function and the definitions using it. We give in the appendix all the necessary definitions that allows for defining the compressed tableau for an $\mathcal{ALCI}+\mathcal{LK}$ ontology. We also prove that a model from the compressed tableau defined for $\mathcal{ALCI}+\mathcal{LK}$ can be extracted. These are the first steps towards extending the algorithm given in Section 6.3 for reasoning in $\mathcal{ALCI}+\mathcal{LK}$.

It is worth mentioning that, in contrast to $\mathcal{ALC}+\mathcal{LK}$, in $\mathcal{ALCI}+\mathcal{LK}$ new individuals (non-nominal star-types) can satisfy the condition of a given link key. This is due to the fact they can be matched to the same set of named individuals (nominal star-types). As a result, reasoning in $\mathcal{ALCI}+\mathcal{LK}$ will force to merge these new star-types.

6.7 Conclusion

This chapter provides a worst-case optimal tableau algorithm for deciding the consistency of an $\mathcal{ALC}+\mathcal{LK}$ ontology. This algorithm is based on the compressed tableau algorithm for the description logic $\mathcal{SHOIQ}$ [16] and it is directed by the application of a set of completion rules including link keys and equality rules. The following chapter explains the implementation of this algorithm and provides a number of proof-of-concept evaluations.

Chapter 7

Implementation and Evaluations

" When I am working on a problem, I never think about beauty. I think only of how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong."
B. Fuller

7.1 Introduction

In this chapter, we describe the architecture of StaréLK, the reasoner that implements the compressed tableau algorithm for the description logic $\mathcal{ALC}+\mathcal{LK}$ given in Chapter 6. Then we report on a pair of sets of experiments. The first one aims at testing the different functionalities of the algorithm, notably, the $\mathcal{ALC}$ and $\mathcal{LK}$ completion rules, matching, and blocking. The second one aims at showing the usefulness of reasoning with link keys for detecting inconsistent link keys, before using them for a specific data interlinking task.

7.2 StaréLK architecture

StaréLK [1] is an open-source software written in Java. It is inspired from Staré [2], the reasoner that implements the compressed tableau algorithm the compressed tableau for $\mathcal{SHOIQ}$ [16]. The reasoner architecture is divided into three complementary modules (Figure 7.1). The first module (Subsection 7.2.1) is concerned with parsing the union of the input datasets, ontologies and ontology alignments. The second module (Subsection 7.2.2) parses the input link keys. The third part (Subsection 7.2.3) implements the compressed tableau algorithm given in Chapter 6. The reasoner uses the OWL API and the ALIGNMENT API to parse the inputs.

[1] https://github.com/anonymousReasoner/Stare
[2] https://github.com/cleduc/stare

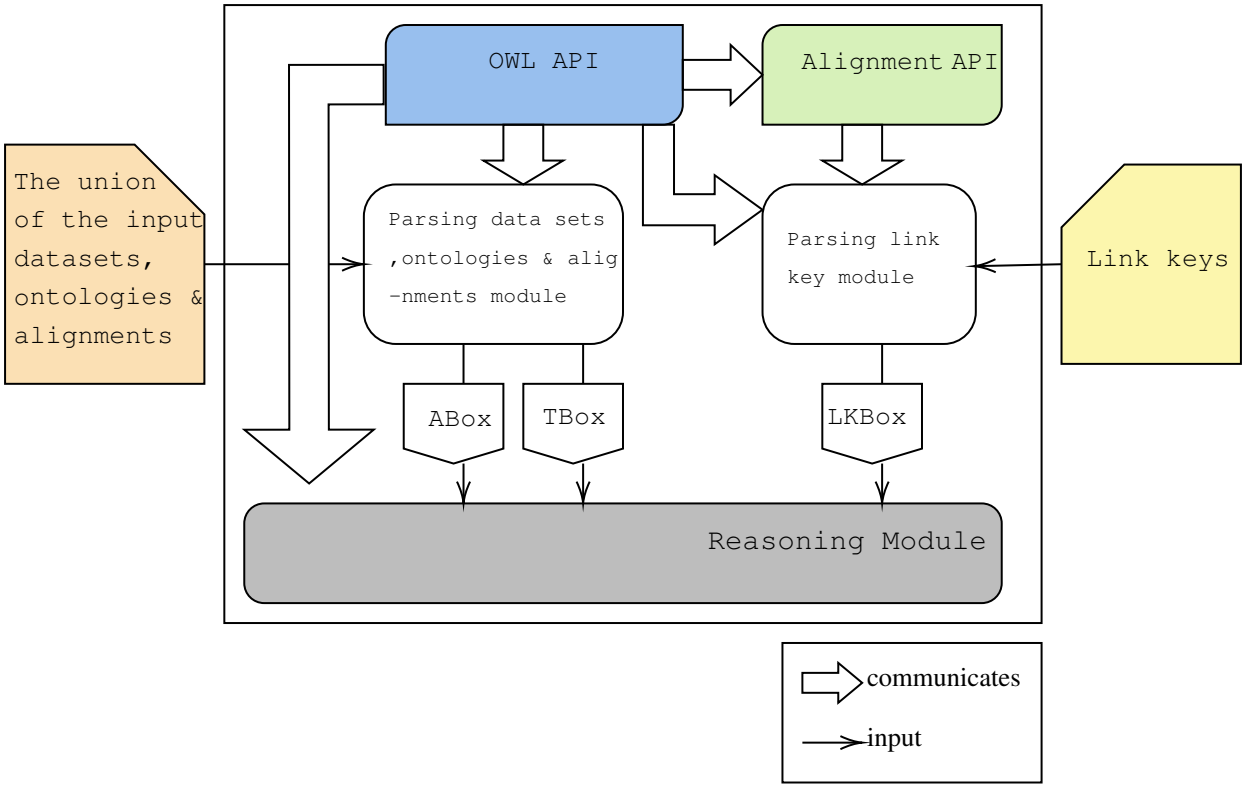


Figure 7.1 – StaréLK architecture.

7.2.1 Datasets, ontologies, and alignments parser module

This module takes as input a file containing the union of the pair of datasets D_1 and D_2, ontologies, and their alignments (in .rdf or .owl extension). The use of a unique namespace for each dataset guarantees their secure fusion

For parsing the input, this module uses the OWL API which is a Java API that has been used as a reference implementation for creating, manipulating, and serialising OWL Ontologies [59]. Since the ontologies and the alignments are combined into one schema, this module does not use the ALIGNMENT API.

For instance, OWLManager class from the OWL API provides a point of convenience for creating an OWLOntologyManager with the standard parsers, storers, etc. This manager is used to load an OWL ontology from a specified file. The input is transformed by this module into two objects the ABox and TBox. These objects are given to the reasoner along with the LKBox from the parsing link key module to perform reasoning.

7.2.2 Link keys parser module

This module loads an input set of link keys. This module uses the ALIGNMENT API that allows for expressing and sharing ontology alignments. Link keys are integrated into the ALIGNMENT API. The Alignment API uses the EDOAL format to express alignments. EDOAL is an Expressive and Declarative Ontology Alignment Language that allows for representing ontology alignments [60]. Link keys are described in EDOAL by specifying their properties and classes.

Note that the ALIGNMENT API uses, as well, the OWL API for executing some of its functionalities. The input set of link keys is described in EDOAL (.rdf extension). The input is transformed by this module into LKBox. The LKBox will be given to the reasoning module.

7.2.3 Reasoning module

This module takes the ABox and TBox from the datasets, schemas, and alignments parser module and the LKBox from the link keys parser module. It implements the compressed tableau algorithm given in Chapter 6. In turn, this module is made-up of three modules (Figure 7.2). The first one is the initialisation module which implements Algorithm 4. The second module is the rule application module and the last one is the checking module. The matching module which implements Algorithms 5, 6, and 7, is embedded into the rule application module. The checking module implements Algorithm 8.

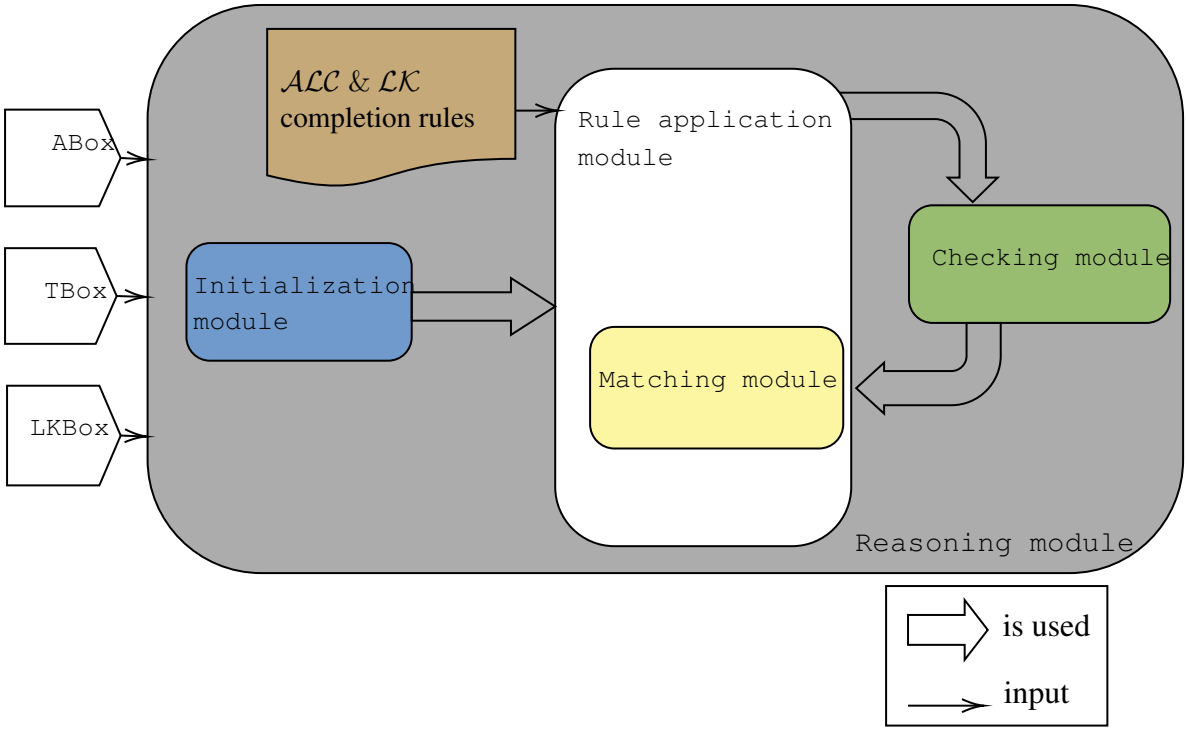


Figure 7.2 – The components of StaréLK reasoning module.

The implementation of these modules necessitates creating several java classes such as the **Linkkey**, **Startype**, **Layer**, and **Match** classes. In the initialisation module, the first layer of the pre-compressed tableau is built. This layer contains a set of nominal star-types each corresponding to an individual contained in the input ontology. Moreover, this module is responsible for the matching the nominal star-types built. This module passes the pre-compressed tableau containing the first layer and the initialised matching function to the rule application module.

The second module is responsible for rule application. The application of $\mathcal{ALC}$ rules is exclusively done in the **Startype** class. The application of $\mathcal{LK}$ rules is exclusively done in the **Linkkey** class. The update of the matching function is done in the **Match** class. The rules changing the core of star-types and thus concerning predecessors call the method

matchingCore. The rules changing the triple of star-types and thus concerning successors call the method **matchTriple**. The merging rules call the method **matchMerge**. These matching functions belong to the **Match** class.

The rule application module communicates constantly with the checking module by exchanging the pre-compressed tableau until the checking module detects a compressed tableau inside the pre-compressed tableau or until there are no more rules to be applied by the rule application module.

7.3 Evaluations

In this section we first test the different functionalities of the reasoner. Then we explain how to integrate link key reasoning in the data interlinking pipeline and we carry out some proof of concept experiments to show the usefulness of this integration for data interlinking.

7.3.1 Correctness of StaréLK

We aim at checking the correctness of StaréLK. The purpose is to test whether the reasoner matches the expected requirements and to ensure that it is defect free.

7.3.1.1 Experimental goals

To check whether the reasoner meets the requirements it is necessary to check its ability to successfully run the algorithm for checking the satisfiability of $\mathcal{ALC}+\mathcal{LK}$. Mainly, it is necessary to check that the completion rules for $\mathcal{ALC}+\mathcal{LK}$ and the corresponding matching algorithms are executed successfully and correctly. More precisely, we aim at testing the various $\mathcal{ALC}$ rules and the corresponding matching algorithms (matchCore and matchTriple) and testing the various $\mathcal{LK}$ rules and the corresponding matching algorithms (matchCore and matchMerge).

7.3.1.2 Experimental setting and results

In order to test if the reasoner meets the requirement we choose the ontologies that stimulate the application of $\mathcal{ALC}+\mathcal{LK}$ completion rules. These ontologies force the algorithm to apply $\mathcal{ALC}$ completion rules that concerns both the cores and the tail of the triples of star-types. These rules call, in turn, respectively matchCore and matchTriple algorithms. These ontologies, as well, force the algorithm to apply the $\rightarrow_{\mathcal{LK}}$ rule and the corresponding matchMerge rule.

We have transformed each of the ontologies of Examples 12 and 14 into respectively two files. The first one contains the ABox and TBox axioms (.owl extension) and the second one contains the LKBox (.rdf extension). We launched the reasoner with them. The files are available at [3]. Example 14 consists of a consistent ontology. Example 12, to the contrary, consists of an inconsistent ontology. Similar to the previous one, it forces the algorithm to apply both $\mathcal{ALC}$ and $\mathcal{LK}$ completion rules and the corresponding matching algorithms. Both examples force the algorithm to create several layers (as a consequence of applying the $\rightarrow_{\exists}$-rule) and to deploy the blocking technique.

[3] https://github.com/anonymousReasoner/Stare/tree/main/test

Experiment 1. *(Running Example 12) We have launched the reasoner with the input ontology and the link key files. The reasoner returned the correct expected answer* true *which means that the input ontology is consistent. It succeeded to detect a compressed tableau from the constructed pre-compressed tableau. The pre-compressed tableau contains three layers. The first layer is nominal, containing a saturated star-type for each individual, and satisfy the link keys in $\mathcal{LK}$. The star-types of the third layer are blocked by star-types of the second layer. Each star-type in the pre-compressed tableau is either dummy, blocked or matched to a non-empty set of star-types.*

Experiment 2. *(Running Example 14) We have launched the reasoner with the input ontology and the link keys files. The reasoner returned the correct answer* false *which means that the input ontology was inconsistent. It succeeded to build a pre-compressed tableau of three layer build the expected number of layers where: The first layer is nominal, containing a saturated star-type for each individual. However, the link keys are not satisfied. The star-types of the third layer are blocked by star-types of the second layer. Each star-type in the pre-compressed tableau is either dummy, blocked or matched to a non-empty set of star-types.*

All the functionalities of the reasoner including blocking, which ensures the termination of the algorithm, are tested and proved to work properly. The reasoner returned the correct and expected answers in both experiments.

7.3.2 Impact of link key reasoning on data interlinking

Link keys are used for data interlinking. Link keys can be extracted from RDF datasets. For extracting link keys, a link key extraction algorithm such as [13] iterates over a pair of RDF datasets (or simply datasets) D and D' to generate another set containing individual pairs and the maximal set of properties on which they share values. After, it derives the set of candidate link keys $\mathcal{LK}$ by replacing each pair of individuals by their types. The quality of the extracted candidate link keys is assessed by the algorithm according to some quality measures like coverage and accuracy. Based on these measures, the best link keys are selected.

However, the extraction algorithms are based on the syntactic equality between the values of properties of the entities contained in D and D'. This comes from the fact that the more the entities share values of properties the more likely they are the same. However, they do not make use of the ontologies O, O' of the datasets D and D', nor the alignments between them. The ontologies O and O' constrain the datasets D and D' and the link keys must between D and D' must follows the constraints O and O'. Since the extraction algorithms do not make use of S, S', nor the alignments then among the extracted link keys there might exist some inconsistent link keys w.r.t O, O' or the alignment between them.

Figure 7.3 shows how link key reasoning can be integrated into the data interlinking pipeline. The aim of this integration is to detect inconsistent link keys returned by the extraction algorithm and to expel them from the link keys used for the data interlinking task.

In module 1 the ontologies O, O', and the alignments between them are unified into one TBox $\mathcal{T}$. Similarly, in module 2, the datasets D and D' are unified into one ABox $\mathcal{A}$. $\mathcal{A}$ and $\mathcal{T}$ now are given to the consistency checking module that checks separately the consistency of each link key in $\mathcal{LK}$. It is worse noting that if a set of link keys are mutually consistent this does not mean that this set of link keys is consistent (Experiment 4).

It is possible, as well, to check the validity of each subset of link keys obtained from $\mathcal{LK}$. This can be done by first building the subsets of $\mathcal{LK}$ and then to launch the reasoner for checking the validity of each subset. This approach has an exponential complexity in the size of the input set of link keys. The reason behind this complexity is the exponential number of subsets of link keys built from the input set of link keys. However, usually, the number of link keys extracted from datasets is not usually huge, thus the exponential complexity of this approach is not real a limitation.

For our experiments we use the link key extraction algorithm given in [14], in which the authors propose a method to extract link keys based on pattern structure. [14] is implemented in a software named Linkex [4].

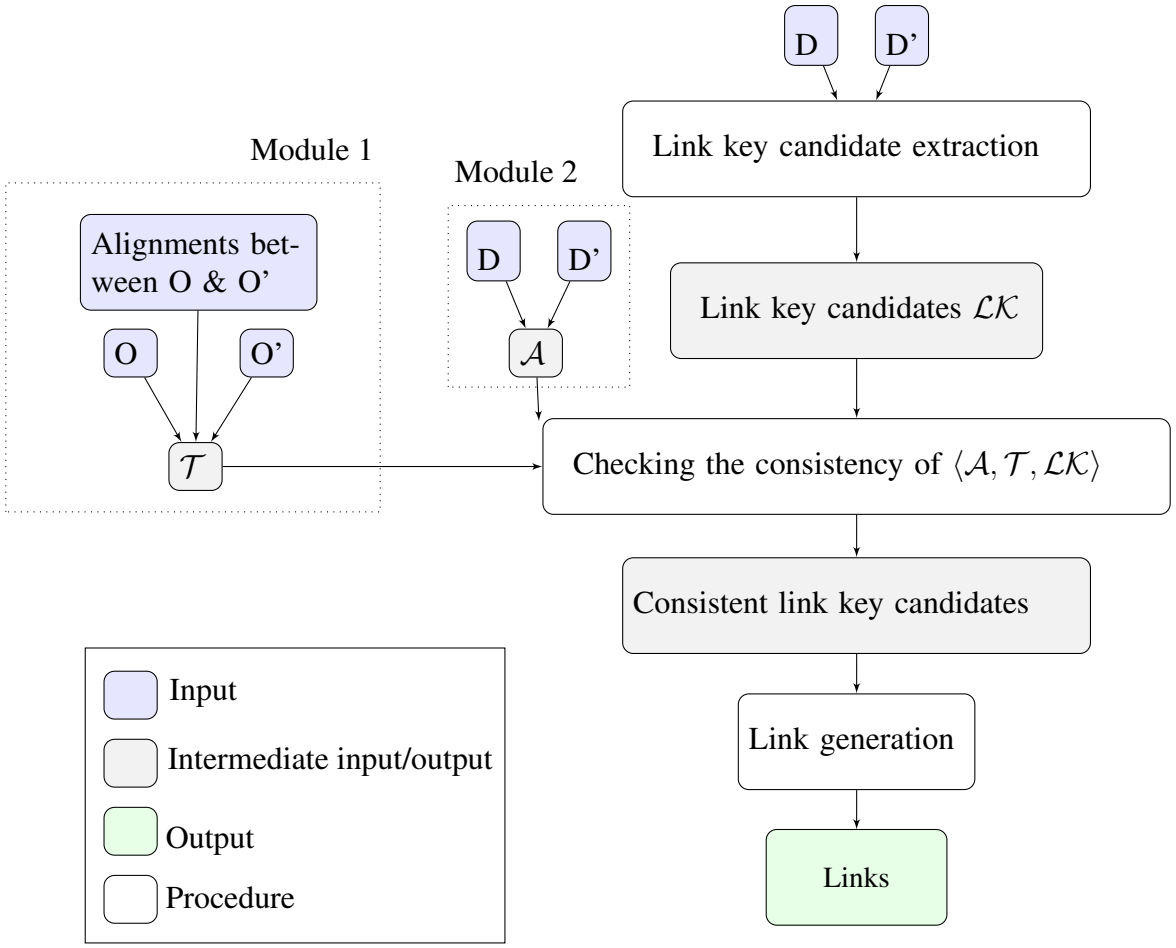


Figure 7.3 – Data interlinking pipeline based on link key inference.

7.3.2.1 Experimental goals

Some of the extracted link keys can have good coverage or discriminability, however, they might be inconsistent link keys w.r.t the ontologies of the datasets or their alignments. We check the consistency of the extracted link keys w.r.t the ontologies and alignments that governs the data. Detecting inconsistent link keys is useful for data interlinking. Since, it decreases the number of false-positive links established by inconsistent link keys.

[4] https://gitlab.inria.fr/moex/linkex

7.3.2.2 Experimental setting and results

For our aim, we follow the pipeline of Figure 7.3. First, we run the extraction algorithm on the pair of given datasets. We specify that the output of this step is a set of link keys described in EDOAL. Then we combine both datasets along with their ontologies and ontology alignments and launch the reasoner with them and the extracted link keys. For the first experiment (Experiment 3) we choose an interlinking task from the data interlinking track of the Ontology Alignment Evaluation Initiative (OAEI) [61]. In this experiment, the datasets contain pairs of properties such as age and houseNumber. These properties might agree on their values, however, they have different semantics. Since modeling data type properties exceeds the expressivity of $\mathcal{ALC}+\mathcal{LK}$ we model them as object properties. Modeling data type properties as object properties does not affect checking the satisfaction of link keys. Checking the satisfaction of link keys requires only checking the equality between property values which translates to checking the equality between individuals.

We have performed another experiment on handcrafted datasets (Experiment 4). Experiment 4 allows to find an inconsistent link key even if there is no alignment available between the input datasets ontologies. This inconsistent link key introduces an equality violating the ontology of one of the input datasets.

Experiment 3. *The datasets* D_1 *and* D_2 *contain descriptions of the instances of classes Person and Address (see Table 7.1).*

Dataset	Triples	Instances	Properties
D_1	5801	Person:500 Address:500	Person:8 Address:5
D_2	6230	Person:500 Address:500	Person:8 Address:5

Table 7.1 – Statistics on datasets D_1 and D_2.

A Person instance is described by the datatype properties givenName, state, surname, dateOfBirth, socSecurityId (ssid), phone_number, age and the object property hasAddress. An Address instance is described by the datatype properties street, houseNumber, postCode, isInSuburb6 and the object property hasAddress.

We have launched the link extraction algorithm on D_1 *and* D_2*. It returns a set of candidate link keys. We choose three candidate link keys among the extracted ones. The first link key has a poor discriminability, the second one has a poor coverage while the last is formed from properties that have different meanings. We have launched the reasoner to check the consistency of each candidate link key w.r.t* D_1*,* D_2 *and their ontologies and ontology alignments. The candidate link keys and the result of consistency checking are present in Table 7.2.*

#	candidate link key	Consistent
k_1	$(\{\langle street, street\rangle\}\langle Address, Address\rangle)$	true
k_2	$(\{\langle phone\_number, phone\_number\rangle, \langle ssid, ssid\rangle\}\langle Person, Person\rangle)$	true
k_3	$(\{\langle age, houseNumber\rangle\}\langle Person, Address\rangle)$	false

Table 7.2 – The link keys between D_1 and D_2 (Experiment 3).

The candidate link keys k_1 and k_2 are consistent w.r.t D_1, D_2 and their ontologies and ontology alignments. The third link key k_3 is inconsistent, since it states that whenever two instances of the classes Person and Address have the same values for the property pair age and houseNumber then they are the same. If this candidate link key is selected for interlinking D_1 and D_2 it will generate equality links between instances of classes Person and Address, that are disjoint classes. For instance it would have linked the entities Person3581 and Address3040 since they have the same value for the property age and house number.

Different organisations may express the data in different manners. For example, in the following experiment, the datasets use two different ways to represent entities. The first dataset uses two classes Male and Female while the second dataset uses only one class, the Person class.

Experiment 4. *The first dataset D_1 describes entities of Male and Female classes, these classes are disjoint. The second dataset D_2 is made-up of only one class, the Person class. In D_1, the instances of both Male and Female class are described using the properties firstName, name, and birthY. In D_2, the instances of both Person class are described using the properties prénom, nom, and annéeN.*

Among the instances contained in D_1 and D_2, the instances a_1 from the Female class, a_5 from the Male class and instance b_3 from share the same first name Charlie which is a common name for both male and females. They share, as well, the same values for the family name and the birth date.

s	firstName	name	birthY
a_1	Charlie	Smith	2010
a_2	Liam	Johnson	2009
a_3	Benjamin	Williams	2011
a_4	Elijah	Brown	2011

(a) Male

s	firstName	name	birthY
a_5	Charlie	Smith	2010
a_6	Amy	Davis	2008
a_7	Annafise	Lopez	2011
a_8	Mitchell	Wilson	2010

(b) Female

Table 7.3 – D_1 is made up of Male and Female classes.

s	prénom	nom	annéeN
b_1	Liam	Johnson	2007
b_2	Lucas	White	2011
b_3	Charlie	Smith	2010
b_4	Amy	Davis	2008

Table 7.4 – D_2 is made up of only the Person class.

We have launched the link key extraction algorithm on D_1 and D_2, it returns a set of candidate link keys. We have launched the reasoner to check the consistency of each candidate link key. The candidate link keys and the result of consistency checking are present in Table 7.5. The link keys k_1 and k_2 are consistent with respect to D_1 and D_2 and their ontologies.

However, both link keys are not consistent with respect to D_1 and D_2 and their ontologies. The first link key $k_1 = (\{\langle firstName, prénom\rangle, \langle name, nom\rangle, \langle birthY, annéeN\rangle\}, \langle Female, Person\rangle)$ links entity a_1 and entity b_3. The second link key $k_2 = (\{\langle firstName, prénom\rangle, \langle name,$

#	candidate	consistent
k_1	$(\{\langle firstName, prénom\rangle, \langle name, nom\rangle, \langle birthY, annéeN\rangle\}, \langle Female, Person\rangle)$	true
k_2	$(\{\langle firstName, prénom\rangle, \langle name, nom\rangle, \langle birthY, annéeN\rangle\}, \langle Male, Person\rangle)$	true

Table 7.5 – The result of experiment 4.

$nom\rangle, \langle birthY, annéeN\rangle\}, \langle Female, Person\rangle)$ links entity a_5 and entity b_3. The reasoner returns false since after it merging the entities a_1 and b_3 and a_5 and b_3, it should merge a_1 and a_5. However, the entities a_1 and a_5 belong respectively to the disjoint classes Male and Female and thus they cannot be merged.

These experiments show that reasoning with link keys is beneficial for discarding inconsistent link keys. These link keys are inconsistent with respect to the ontologies and ontology alignments of the input pair of datasets. These link keys, if used, would generate inconsistent links between different entities. We have also noticed that the reasoner was able to detect inconsistent link keys even if there is no alignment available between the ontologies of the input datasets (Experiment 4).

7.4 Conclusion

We explain the architecture of the reasoner that implements the compressed tableau algorithm given in Chapter 6. Then perform two sets of experiments. The first type of experiments tests the functionalities of the algorithm including the completion rules, matching, and blocking. These experiments reveal that the reasoner work as expected. The second kind of experiments checks the consistency of link keys extracted by a link key extraction algorithm. It shows the importance and relevance of reasoning with link keys for the data interlinking task.

Chapter 8

Conclusion and Perspectives

" You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist".
Friedrich Nietzsche

Data interlinking is a crucial task for enriching and completing RDF graphs present on the semantic web. Link keys are among the proposed methods to address the problem of interlinking RDF graphs. In this thesis, we studied the problem of reasoning with link keys. We have designed several algorithms for reasoning with link keys and implement a worst-case optimal tableau algorithm for reasoning with link keys. In what follows, we summarize the main achievements of this thesis and then we give various directions for the future work.

8.1 Summary and Conclusion

We have first introduced the description logic $\mathcal{ALC}+\mathcal{LK}$, this logic extends $\mathcal{ALC}$ by individual equalities and link keys $\mathcal{LK}$. This logic is used to perform reasoning with link keys, in particular for checking link keys consistency.

Reducing link key entailment to link key consistency checking In Chapter 4, we show that link key entailment can be reduced to link key consistency checking without introducing the negation of link keys. More precisely, we have proved that an ontology entails a link key when this ontology extended by a set of role and concept assertions presenting a witness for this link key negation is inconsistent.

Reasoning in the description logic $\mathcal{ALC}+\mathcal{LK}$ is decidable In Chapter 4, we extend that standard tableau algorithm for reasoning with the description logic $\mathcal{ALC}$ by adding completion rules for handling link keys and equalities. This algorithm uses anywhere blocking to extend the potential blockers of an individual, resulting in reduced-sized models. We show that this algorithm terminates and that it is sound and complete. The complexity of this reasoning algorithm is 2ExpTime. This shows that reasoning with link keys in the description logic $\mathcal{ALC}$ is decidable. We have also noticed that the completion rules added for handling link keys and equalities do not require additional computational power than that of $\mathcal{ALC}$.

Proving that the complexity of reasoning with link keys is EXPTIME In Chapter 5, we provide an EXPTIME algorithm for reasoning in the $\mathcal{ALC}+\mathcal{LK}$. This complexity result was achieved by adapting the compressed tableau algorithm [16] to perform reasoning in $\mathcal{ALC}+\mathcal{LK}$. This complexity result shows that reasoning in $\mathcal{ALC}+\mathcal{LK}$ has the same complexity as reasoning in $\mathcal{ALC}$ and supports the applicability of reasoning with link keys in practice.

Designing and implementing an EXPTIME tableau algorithm for reasoning in the description logic $\mathcal{ALC}+\mathcal{LK}$ In Chapter 6, we present a worst-case optimal tableau algorithm for reasoning in the description logic $\mathcal{ALC}+\mathcal{LK}$. This algorithm is directed by the application of completion rules and is implemented (Chapter 7). The application of completion rules allows for easy and practical implementation of the algorithm.

Testing the impact of reasoning with link keys on the data interlinking task In Chapter 7, we have carried out a set of experiments for evaluating the impact of reasoning with link keys on the task of data interlinking. It turns out that the reasoning with link keys is useful for the task of data interlinking. In particular, given a pair of RDF graphs, their ontologies, and ontology alignments, the reasoner was capable of detecting inconsistent link keys, generated by a link key extraction algorithm, w.r.t the input data, ontologies, and ontology alignments. These link keys should be discarded and not used for interlinking the given pair of RDF graphs.

8.2 Future Work

We summarize below the various directions for future work.

Extending the tableau algorithm for the description logic $\mathcal{ALC}$ and link keys to more expressive families of description logics Extending the description logic $\mathcal{ALC}+\mathcal{LK}$ will increase the range of the applications using reasoning with link keys. Notably, we would like to study the extension of the description logic $\mathcal{ALC}+\mathcal{LK}$ with inverse roles and role composition.

Merging link key extraction and reasoning One interesting direction would be to integrate a reasoning mechanism in the link key extraction process. This will avoid extracting link keys that are inconsistent w.r.t the RDF graphs, ontologies, or ontology alignments. However, designing a sound and complete reasoning procedure to be integrated with the link key extraction process is not straightforward. It requires to perform on the fly checking of rule application and clash-detection.

Optimising StaréLK Despite of its theoretical worst-case optimal complexity, the algorithm suffers from practical performance issues. Optimization techniques such as absorption, backjumping, and caching are used in a wide variety of standard tableau algorithms such as [49, 21] and can be adopted to improve the performance of StaréLK.

Reasoning with large ABoxes Reasoning with large ABoxes is immense. Moreover, not all the data contained in the ABox serves for checking the consistency of an input link key. One approach would be to select the part of the ABox that helps for checking the satisfaction of one particular link key. Also, we would like to stress the fact that for checking the consistency of

a link key w.r.t the knowledge available it is sufficient to artificially introduce the negation of this link key.

Finally, we would like to assess the impact of reasoning with link keys by carrying out the following experiments.

Supervised calculation of the reduction of false-positive links Detecting inconsistent link keys will decrease the number of incorrect identity links established between different entities (false-positive links). We would like to evaluate the percentage of this reduction w.r.t a reference set of identity links.

Experimenting with real datasets Last but not least, we plan to conduct a series of experiments on real datasets such as INSEE and Geonames.

Chapter 9

Appendix

As discussed before, extending $\mathcal{ALC}+\mathcal{LK}$ with inverse roles will increase the scope of applications of reasoning with link keys. In the presence of inverse roles, link keys maybe satisfied by named individuals that do not occur in the input ABox. In this case, link keys cannot be anymore expressed as DL-safe rules and the corresponding reasoning algorithms such as [35] cannot be used to do reasoning with link keys.

9.1 Extending the description logic $\mathcal{ALC}$ with inverse roles

In what follows we give the necessary definitions that allow to introduce the compressed tableau for an $\mathcal{ALCI}+\mathcal{LK}$ ontology. Defining a compressed tableau is a first step towards designing a compressed tableau algorithm for reasoning in $\mathcal{ALCI}+\mathcal{LK}$. Once such an algorithm exists, it is guaranteed to be sound. Such an algorithm is sound because have proved that an interpretation representing a model for the input ontology can be always extracted from the defined compressed tableau (Lemma 18).

The definitions of star-type, layer and saturated and clash-free star-types are remained unaltered. The definition of the matching function varies, since in the presence of inverse roles, relations between star-types are now bi-directional (Figure 9.1).

Definition 28 (Matching function over a sequence of layers). *Let $\mathcal{O}$ be an $\mathcal{ALCI}+\mathcal{LK}$ ontology. Let $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^{n}$ be a finite sequence of layers. A* matching function *over $\mathbf{\Lambda}$ is a function Ω that associates each pair (σ, τ) where σ is a star-type in $\mathbf{\Lambda}$ and τ is a non dummy triple in σ to a non-empty set $\{(\sigma_1, \tau_1), \cdots, (\sigma_m, \tau_m)\}$ where $\sigma_i \in \mathbf{\Lambda}$ and $\tau_i \in \sigma_i$ for all $1 \leq i \leq m$ and that satisfies:*

1. *for every $\sigma \in \mathbf{\Lambda}$ and $\tau \in \sigma$, if $(\sigma', \tau') \in \Omega(\sigma, \tau)$ then $\mathsf{tail}(\tau) = \mathsf{core}(\sigma')$, $\mathsf{tie}(\tau) = \mathsf{tie}^{-}(\tau')$ and $\mathsf{tail}(\tau') = \mathsf{core}(\sigma)$;*
2. *for every $\sigma \in \mathbf{\Lambda}$ and $\tau \in \sigma$, if $(\sigma', \tau') \in \Omega(\sigma, \tau)$ then $(\sigma, \tau) \in \Omega(\sigma', \tau')$;*
3. *for every $\sigma \in \mathbf{\Lambda}$ and $\tau \in \sigma$, if $(\sigma', \tau') \in \Omega(\sigma, \tau)$ then*
   (a) *if $\sigma \in \Lambda_0$ then $\sigma' \in \Lambda_0 \cup \Lambda_1$;*
   (b) *if $\sigma \in \Lambda_k$ and $0 < k < n$ then $\sigma' \in \Lambda_{k-1} \cup \Lambda_{k+1}$.*

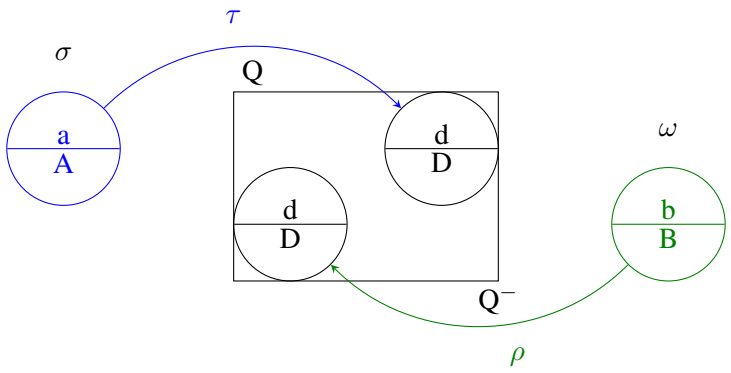


Figure 9.1 – Matching function between star-types, $(\omega, \rho) \in \Omega(\sigma, \tau)$.

As a consequence, the neighbourhood definition is modified.

Definition 29 (Neighbour)**.** *A star-type σ is called a* ρ-successor *(or successor) of ω via a triple $\rho \in \omega$ if there exits a triple $\tau \in \sigma$ such that $(\sigma, \tau) \in \Omega(\omega, \rho)$, and ω is called a* ρ-predecessor *(or predecessor) of σ. In this case, if $R \in \mathsf{tie}(\rho)$ then σ is called an R-successor of ω, and ω an R-predecessor of σ. A star-type is called a* neighbour *of another one if one is a successor of the other.*

Based on these definitions we now introduce the compressed tableau for an $\mathcal{ALCI}$ ontology. This definition uses the new definition of the matching function. Note that Item 5, related to role assertions, is modified with respect to its alternative in the compressed tableau definition for an $\mathcal{ALC}$ ontology.

Definition 30 (Compressed tableau of an $\mathcal{ALCI}$ ontology). *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology. Let $\mathbf{I}$ be the set of individuals of $\mathcal{O}$. Let $\mathbf{\Lambda}$ be a set of layers and Ω a matching function over $\mathbf{\Lambda}$. The pair $\langle \mathbf{\Lambda}, \Omega \rangle$ is said to be a* compressed tableau *of $\langle \mathcal{A}, \mathcal{T} \rangle$ if the following conditions are satisfied:*

1. *Every $\sigma \in \mathbf{\Lambda}$ is clash-free and, unless it is blocked, saturated.*
2. *For every $\sigma \in \mathbf{\Lambda}$, σ is nominal if and only if $\sigma \in \Lambda_0$.*
3. *For every $a \in \mathbf{I}$, there exists a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma_a)$.*
4. *For every $C(a) \in \mathcal{A}$, $C \in \mathsf{core}_\mathsf{C}(\sigma_a)$.*
5. *For every $R(a, b) \in \mathcal{A}$, there exist $\sigma, \sigma' \in \Lambda_0$ and $\tau \in \sigma, \tau' \in \sigma'$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma)$, $b \in \mathsf{core}_\mathsf{I}(\sigma')$, $(\sigma', \tau') \in \Omega(\sigma, \tau)$ and $R \in \mathsf{tie}(\tau)$.*

The definition of star-types satisfying a link key condition is also modified.

Definition 31 (Star-types satisfying a link key condition). *Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology, and $\langle \mathbf{\Lambda}, \Omega \rangle$ a pre-compressed tableau of $\mathcal{O}$. Let $\lambda = (\{\langle P_i, Q_i \rangle\}_{i=1}^n \mathsf{linkkey} \langle C, D \rangle) \in \mathcal{LK}$ and $\sigma, \sigma' \in \mathbf{\Lambda}$.*

1. *We say that σ, σ'* weakly satisfy the condition *of λ if there exist $\tau_1, \ldots, \tau_n \in \sigma$ and $\upsilon_1, \ldots, \upsilon_n \in \sigma'$ such that, for every $1 \leq i \leq n$, $P_i \in \mathsf{tie}(\tau_i)$, $Q_i \in \mathsf{tie}(\upsilon_i)$ and there exist $\sigma'' \in \mathbf{\Lambda}$ and $\tau_1', \upsilon_1', \ldots, \tau_n', \upsilon_n' \in \sigma''$ such that $(\sigma'', \tau_i') \in \Omega(\sigma, \tau_i)$ and $(\sigma'', \upsilon_i') \in \Omega(\sigma', \upsilon_i)$.*

2. *We say that* σ *and* σ' satisfy the condition of λ *if they weakly satisfy the condition of* λ, $C \in \mathsf{core}_\mathsf{C}(\sigma)$ *and* $D \in \mathsf{core}_\mathsf{C}(\sigma')$.

We now give the definition of a compressed tableau for an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{LK})$. A compressed tableau for $\mathcal{O}$ is a compressed tableau for the $\mathcal{ALCI}$ ontology $\langle \mathcal{A}, \mathcal{T} \rangle$ such that every link key in $\mathcal{LK}$ and every equality assertion in $\mathcal{A}$ is satisfied.

Definition 32 (Compressed tableau of an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology). *Let* $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ *be an* $\mathcal{ALCI}$+$\mathcal{LK}$ *ontology. A* compressed tableau *of* $\mathcal{O}$ *is a pair* $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ *that satisfies the following conditions:*

1. $\mathcal{CT}$ *is a compressed tableau of* $\langle \mathcal{A}, \mathcal{T} \rangle$*;*
2. *for every* $a \approx b \in \mathcal{A}$ *and every* $\sigma, \sigma' \in \mathbf{\Lambda}$*, if* $a \in \mathsf{core}_\mathsf{I}(\sigma)$ *and* $b \in \mathsf{core}_\mathsf{I}(\sigma')$ *then* $\sigma{=}\sigma'$*;*
3. *for every* $(\{\langle P_k, Q_k \rangle\}_{k=1}^n \ \mathsf{linkkey}\ \langle C, D \rangle) \in \mathcal{LK}$ *and every* $\sigma, \sigma' \in \mathbf{\Lambda}$*, if* σ *and* σ' *weakly satisfy the condition of* $(\{\langle P_k, Q_k \rangle\}_{k=1}^n \ \mathsf{linkkey}\ \langle C, D \rangle)$*, then* $\{C, \mathsf{nnf}(C)\} \cap \mathsf{core}_\mathsf{C}(\sigma) \neq \emptyset$ *and* $\{D, \mathsf{nnf}(D)\} \cap \mathsf{core}_\mathsf{C}(\sigma') \neq \emptyset$*;*
4. *for every* $(\{\langle P_k, Q_k \rangle\}_{k=1}^n \ \mathsf{linkkey}\ \langle C, D \rangle) \in \mathcal{LK}$ *and every* $\sigma, \sigma' \in \mathbf{\Lambda}$*, if* σ *and* σ' *satisfy the condition of* $(\{\langle P_k, Q_k \rangle\}_{k=1}^n \ \mathsf{linkkey}\ \langle C, D \rangle)$*, then* $\sigma{=}\sigma'$*.*

In what follows we prove that, given an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology and a compressed tableau for this ontology, a model for this ontology can be extracted its compressed tableau.

Lemma 18 (Model extraction). *Let* $\mathcal{O}$ *be an* $\mathcal{ALCI}$+$\mathcal{LK}$ *ontology. If there exists a compressed tableau of* $\mathcal{O}$ *then* $\mathcal{O}$ *is consistent.*

Proof. Let $\mathcal{O} = \langle \mathcal{A}, \mathcal{T}, \mathcal{LK} \rangle$ be an $\mathcal{ALCI}$+$\mathcal{LK}$ ontology. Let $\mathbf{I}$ and $\mathbf{R}$ be the sets of individuals and roles of $\mathcal{O}$.

Let us assume that there exists a compressed tableau $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ of $\mathcal{O}$, where $\mathbf{\Lambda} = \langle \Lambda_k \rangle_{k=0}^n$. We consider the set Path of all paths of star-types in $\mathbf{\Lambda}$, i.e. $\mathsf{Path} = \{\langle \sigma_1, \ldots, \sigma_m \rangle \mid m \geq 1$ and $\sigma_k \in \mathbf{\Lambda}$ for $1 \leq k \leq m\}$. If $\mathsf{p} \in \mathsf{Path}$, $\mathsf{last}(\mathsf{p})$ denotes the last element of p and $\mathsf{length}(\mathsf{p})$ denotes its length; if $\mathsf{p}, \mathsf{p}' \in \mathsf{Path}$, $\mathsf{p} \cdot \mathsf{p}'$ denotes the concatenation of p and p'. If $a \in \mathbf{I}$ then, by (3) of Definition 30, there exists a unique star-type $\sigma_a \in \Lambda_0$ such that $a \in \mathsf{core}_\mathsf{I}(\sigma)$. We denote $\langle \sigma_a \rangle \in \mathsf{Path}$ by p_a.

We define an interpretation $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ from $\mathcal{CT}$ in the following way:

1. $\Delta^\mathcal{I}$ is inductively defined as follows:
   (a) $\mathsf{p}_a \in \Delta^\mathcal{I}$ for every $a \in \mathbf{I}$;
   (b) if $\mathsf{p} \in \Delta^\mathcal{I}$ and there exist $\tau \in \mathsf{last}(\mathsf{p})$, $\sigma \in \mathbf{\Lambda}$ and $\tau' \in \sigma$ such that $(\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$, then, if σ is not blocked then $\mathsf{p} \cdot \langle \sigma \rangle \in \Delta^\mathcal{I}$, otherwise, $\mathsf{p} \cdot \langle \mathsf{b}(\sigma) \rangle \in \Delta^\mathcal{I}$.
2. For every $a \in \mathbf{I}$, $a^\mathcal{I} = \mathsf{p}_a$.
3. For every concept name A in $\mathcal{O}$, $A^\mathcal{I} = \{\mathsf{p} \in \Delta^\mathcal{I} \mid A \in \mathsf{core}_\mathsf{C}(\mathsf{last}(\mathsf{p}))\}$.

4. For every role name $R \in \mathbf{R}$, $R^{\mathcal{I}} = \Delta_1^R \cup \Delta_2^R \cup \Delta_3^R$, where

$$\Delta_1^R = \{(\mathsf{p}_a, \mathsf{p}_b) : a, b \in \mathbf{I} \text{ and there exist } \tau \in \sigma_a, \tau' \in \sigma_b \text{ such that } (\sigma_b, \tau') \in \Omega(\sigma_a, \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

$$\Delta_2^R = \{(\mathsf{p}, \mathsf{p} \cdot \langle\sigma\rangle) : \mathsf{p} \in \Delta^{\mathcal{I}}, \sigma \in \mathbf{\Lambda}, \sigma \text{ is not blocked and there exist } \tau \in \mathsf{last}(\mathsf{p}), \tau' \in \sigma \text{ such that } (\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

$$\Delta_3^R = \{(\mathsf{p}, \mathsf{p} \cdot \langle\mathsf{b}(\sigma)\rangle) : \mathsf{p} \in \Delta^{\mathcal{I}}, \sigma \in \mathbf{\Lambda}, \sigma \text{ is blocked and there exist } \tau \in \mathsf{last}(\mathsf{p}), \tau' \in \sigma \text{ such that } (\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau) \text{ and } R \in \mathsf{tie}(\tau)\}$$

Let us prove that $\mathcal{I}$ is indeed an interpretation. We only need to prove that $\Delta^{\mathcal{I}} \neq \emptyset$ and that, for every name $R \in \mathbf{R}$, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Since $\mathbf{I} \neq \emptyset$, then, there exists at least one individual $a \in \mathbf{I}$. Thus, by (1.a), $\mathsf{p}_a \in \Delta^{\mathcal{I}}$. Therefore, $\Delta^{\mathcal{I}} \neq \emptyset$. Now, let $R \in \mathbf{R}$. By (1.a), $\Delta_1^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. By (1.b), $\Delta_2^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $\Delta_3^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. This implies $\Delta_1^R \cup \Delta_2^R \cup \Delta_3^R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Therefore, $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Let us prove that $\mathcal{I}$ is a model of $\mathcal{O}$, i.e. $\mathcal{I}$ satisfies all the assertions of $\mathcal{A}$, all the GCIs of $\mathcal{T}$ and all the link keys of $\mathcal{LK}$. We start by proving that $\mathcal{I}$ satisfies all the assertions of $\mathcal{A}$.

Assume that $a \approx b \in \mathcal{A}$. By the definition of $\mathcal{I}$, $a^{\mathcal{I}} = \mathsf{p}_a$ and $b^{\mathcal{I}} = \mathsf{p}_b$, where $\mathsf{p}_a = \langle\sigma_a\rangle$, $\mathsf{p}_b = \langle\sigma_b\rangle$ and σ_a and σ_b are the unique star-types of Λ_0 such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_b)$. Since $\mathcal{CT}$ is a compressed tableau, by (2) of Definition 32, we have $\sigma_a{=}\sigma_b$. This implies $\mathsf{p}_a{=}\mathsf{p}_b$. Therefore, $a^{\mathcal{I}} = b^{\mathcal{I}}$, i.e. $\mathcal{I} \models a \approx b$.

Assume that $a \not\approx b \in \mathcal{A}$. We proceed by contradiction. Assume $\mathcal{I} \not\models a \not\approx b$. This means $a^{\mathcal{I}}{=}b^{\mathcal{I}}$. Then, by the definition of $\mathcal{I}$, $\mathsf{p}_a = \mathsf{p}_b$. Thus, $\sigma_a = \sigma_b$, where σ_a and σ_b are the unique star-types of Λ_0 such that $a \in \mathsf{core}_{\mathsf{I}}(\sigma_a)$ and $b \in \mathsf{core}_{\mathsf{I}}(\sigma_b)$. Let $\sigma{=}\sigma_a{=}\sigma_b$. Therefore, $\sigma \in \mathbf{\Lambda}$ and $a, b \in \mathsf{core}_{\mathsf{I}}(\sigma)$. By (2) of Definition 25, σ is not clash-free. This contradicts the fact that $\mathcal{CT}$ is a compressed tableau, so it must be $\mathcal{I} \models a \not\approx b$.

Assume that $R(a, b) \in \mathcal{A}$. Since $\mathcal{CT}$ is a compressed tableau, by (5) of Definition 30, there exist $\tau \in \sigma_a, \tau' \in \sigma_b$ such that $(\sigma_b, \tau') \in \Omega(\sigma_a, \tau)$ and $R \in \mathsf{tie}(\tau)$. Then, by the definition of $\mathcal{I}$, we have $(\mathsf{p}_a, \mathsf{p}_b) \in R^{\mathcal{I}}$. Also, $a^{\mathcal{I}} = \mathsf{p}_a$ and $b^{\mathcal{I}} = \mathsf{p}_b$. Therefore, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, which is the same as $\mathcal{I} \models R(a, b)$.

Assume $C(a) \in \mathcal{A}$. To prove $\mathcal{I} \models C(a)$, we use the following:

$$\text{For every } \mathsf{p} \in \Delta^{\mathcal{I}}, \text{ if } E \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p})) \text{ then } \mathsf{p} \in E^{\mathcal{I}} \qquad (\dagger)$$

Assume that (†) is true. Since $C(a) \in \mathcal{A}$, by (4) of Definition 30, we have that $C \in \mathsf{core}_{\mathsf{C}}(\sigma_a)$. Then, $C \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}_a))$. By (†), $\mathsf{p}_a \in C^{\mathcal{I}}$. Since $a^{\mathcal{I}} = \mathsf{p}_a$, we have $a^{\mathcal{I}} \in C^{\mathcal{I}}$, i.e. $\mathcal{I} \models C(a)$.

We now show that $\mathcal{I}$ satisfies all GCIs in $\mathcal{T}$. Let $C \sqsubseteq D \in \mathcal{T}$. Let $\mathsf{p} \in C^{\mathcal{I}}$. Since $\mathcal{CT} = \langle\mathbf{\Lambda}, \Omega\rangle$ is a compressed tableau and $\mathsf{last}(\mathsf{p}) \in \mathbf{\Lambda}$, by (1) of Definition 30, $\mathsf{last}(\mathsf{p})$ is saturated. Since $C \sqsubseteq D \in \mathcal{T}$, by (5) of Definition 24, then $\mathsf{nnf}(C)$ or D belongs to $\mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. If $\mathsf{nnf}(C) \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$, by (†), $\mathsf{p} \in (\mathsf{nnf}(C))^{\mathcal{I}}$. This contradicts $\mathsf{p} \in C^{\mathcal{I}}$ because $(\mathsf{nnf}(C))^{\mathcal{I}} = (\neg C)^{\mathcal{I}}$ and $(\neg C)^{\mathcal{I}} \cap C^{\mathcal{I}} = \emptyset$. Therefore, it must be $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Then, by (†), $\mathsf{p} \in D^{\mathcal{I}}$. This proves $\mathcal{I} \models C \sqsubseteq D$.

Finally, we show that $\mathcal{I}$ satisfies all link keys in $\mathcal{LK}$. Let $\lambda = (\{\langle P_i, Q_i\rangle\}_{i=1}^n \text{ linkkey } \langle C, D\rangle) \in \mathcal{LK}$. Let $\mathsf{p}, \mathsf{q}, \mathsf{r}_1, \ldots, \mathsf{r}_n \in \Delta^{\mathcal{I}}$ such that $\mathsf{p} \in C^{\mathcal{I}}$, $\mathsf{q} \in D^{\mathcal{I}}$, and $(\mathsf{p}, \mathsf{r}_i) \in P_i^{\mathcal{I}}$ and $(\mathsf{q}, \mathsf{r}_i) \in Q_i^{\mathcal{I}}$ for $1 \leq i \leq n$. We have to prove that $\mathsf{p}{=}\mathsf{q}$.

Assume that there exists $i_0 \in \{1, \ldots, n\}$ such that $\mathsf{length}(\mathsf{r}_{i_0}) \geq 2$. Then, $(\mathsf{p}, \mathsf{r}_{i_0}) \in \Delta_2^{P_{i_0}} \cup \Delta_3^{P_{i_0}}$ and $(\mathsf{q}, \mathsf{r}_{i_0}) \in \Delta_2^{Q_{i_0}} \cup \Delta_3^{Q_{i_0}}$. Therefore, there exist $\sigma, \sigma' \in \mathbf{\Lambda}$ such that $\mathsf{r}_{i_0} = \mathsf{p} \cdot \langle\sigma\rangle$ and $\mathsf{r}_{i_0} = \mathsf{q} \cdot \langle\sigma'\rangle$. Then $\mathsf{p} \cdot \langle\sigma\rangle = \mathsf{q} \cdot \langle\sigma'\rangle$, and, thus, $\mathsf{p} = \mathsf{q}$ (and $\sigma = \sigma'$).

Assume that $\mathsf{length}(\mathsf{r}_i) = 1$ for every $1 \leq i \leq n$. Then, $(\mathsf{p}, \mathsf{r}_i) \in \Delta_1^{P_i}$ and $(\mathsf{q}, \mathsf{r}_i) \in \Delta_1^{Q_i}$ for every $1 \leq i \leq n$. Therefore, there exist $a, b_1, \ldots, b_n \in \mathbf{I}$, $\tau_1, \ldots, \tau_n \in \sigma_a$ and $\tau_1' \in \sigma_{b_1}, \ldots, \tau_n' \in \sigma_{b_n}$ such that $\mathsf{p} = \mathsf{p}_a$, $\mathsf{r}_i = \mathsf{p}_{b_i}$, $(\sigma_{b_i}, \tau_i') \in \Omega(\sigma_a, \tau_i)$ and $P_i \in \mathsf{tie}(\tau_i)$ for $1 \leq i \leq n$, and there exist $c, d_1, \ldots, d_n \in \mathbf{I}$, $\upsilon_1, \ldots, \upsilon_n \in \sigma_c$ and $\upsilon_1' \in \sigma_{d_1}, \ldots, \upsilon_n' \in \sigma_{d_n}$ such that $\mathsf{q} = \mathsf{p}_c$, $\mathsf{r}_i = \mathsf{p}_{d_i}$, $(\sigma_{d_i}, \upsilon_i') \in \Omega(\sigma_c, \upsilon_i)$ and $Q_i \in \mathsf{tie}(\upsilon_i)$ for $1 \leq i \leq n$. Therefore, $\mathsf{p}_{b_i} = \mathsf{p}_{d_i}$ for $1 \leq i \leq n$. Since $\mathsf{p}_x = \langle \sigma_x \rangle$ for any $x \in \mathbf{I}$, then $\sigma_{b_i} = \sigma_{d_i}$ for $1 \leq i \leq n$.

Let $\sigma_i = \sigma_{b_i} = \sigma_{d_i}$ for $1 \leq i \leq n$. Then, $(\sigma_i, \tau_i') \in \Omega(\sigma_a, \tau_i)$ and $(\sigma_i, \upsilon_i') \in \Omega(\sigma_c, \upsilon_i)$.

Then, σ_a and σ_c weakly satisfy the condition of λ. Since $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is a compressed tableau, by (3) of Definition 32, we have $\{C, \mathsf{nnf}(C)\} \cap \mathsf{core}_{\mathsf{C}}(\sigma_a) \neq \emptyset$ and $\{D, \mathsf{nnf}(D)\} \cap \mathsf{core}_{\mathsf{C}}(\sigma_b) \neq \emptyset$. Assume that $\mathsf{nnf}(C) \in \mathsf{core}_{\mathsf{C}}(\sigma_c)$.

Then, by (†), we have $\mathsf{p} \in (\mathsf{nnf}(C))^{\mathcal{I}}$, which contradicts $\mathsf{p} \in C^{\mathcal{I}}$. Thus, it must be $C \in \mathsf{core}_{\mathsf{C}}(\sigma_a)$. In a similar manner, $D \in \mathsf{core}_{\mathsf{C}}(\sigma_c)$. Therefore, σ_a and σ_c satisfy the condition of λ. By (4) of Definition 32, $\sigma_a = \sigma_c$. This implies $\mathsf{p}_a = \mathsf{p}_c$, thus $\mathsf{p} = \mathsf{q}$.

It remains to prove (†). Let $\mathsf{p} \in \Delta^{\mathcal{I}}$ and assume $E \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. We have to prove that $\mathsf{p} \in E^{\mathcal{I}}$. We will proceed by induction on the length of E, but, before anything else, note that, since $\mathcal{CT} = \langle \mathbf{\Lambda}, \Omega \rangle$ is a compressed tableau, then $\mathsf{last}(\mathsf{p})$ is saturated and clash-free.

1. Assume $E = A$, where A is a concept name. We have $A \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. By the definition of $\mathcal{I}$, $A^{\mathcal{I}} = \{\mathsf{p} \in \Delta^{\mathcal{I}} \mid A \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))\}$. Therefore, $\mathsf{p} \in A^{\mathcal{I}}$.

2. Assume $E = E_1 \sqcap E_2$. We have that $E_1 \sqcap E_2 \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is saturated, then $E_1, E_2 \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. By induction hypothesis, $\mathsf{p} \in E_1^{\mathcal{I}}$ and $\mathsf{p} \in E_2^{\mathcal{I}}$, which implies $\mathsf{p} \in (E_1 \sqcap E_2)^{\mathcal{I}}$, thus $\mathsf{p} \in E^{\mathcal{I}}$.

3. Assume $E = E_1 \sqcup E_2$. We have that $E_1 \sqcup E_2 \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Since $\mathsf{last}(\mathsf{p})$ is saturated, then $\{E_1, E_2\} \cap \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p})) \neq \emptyset$. Therefore, $E_1 \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$ or $E_2 \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. By induction hypothesis, $\mathsf{p} \in E_1^{\mathcal{I}}$ or $\mathsf{p} \in E_2^{\mathcal{I}}$, which implies $\mathsf{p} \in (E_1 \sqcup E_2)^{\mathcal{I}}$, thus $\mathsf{p} \in E^{\mathcal{I}}$.

4. Assume $E = \forall R.D$. We have $\forall R.D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Let $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. We have to prove that $\mathsf{p}' \in D^{\mathcal{I}}$.

   Since $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$, by the definition of $\mathcal{I}$, there are three possibilities :

   (a) There exist $a, b \in \mathbf{I}$, $\tau \in \sigma_a$ and $\tau' \in \sigma_b$ such that $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}_a, \mathsf{p}_b)$, $(\sigma_b, \tau') \in \Omega(\sigma_a, \tau)$ and $R \in \mathsf{tie}(\tau)$.

   (b) There exist $\sigma \in \mathbf{\Lambda}$, $\tau \in \mathsf{last}(\mathsf{p})$ and $\tau' \in \sigma$ such that σ is non blocked, $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}, \mathsf{p} \cdot \langle \sigma \rangle)$, $(\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$ and $R \in \mathsf{tie}(\tau)$.

   (c) There exist $\sigma \in \mathbf{\Lambda}$, $\tau \in \mathsf{last}(\mathsf{p})$ and $\tau' \in \sigma$ such that σ is blocked, $(\mathsf{p}, \mathsf{p}') = (\mathsf{p}, \mathsf{p} \cdot \langle \mathsf{b}(\sigma) \rangle)$, $(\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$ and $R \in \mathsf{tie}(\tau)$.

   Consider the case (b). Since $\mathsf{last}(\mathsf{p})$ is saturated, $\forall R.D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$ and $R \in \mathsf{tie}(\tau)$, then $D \in \mathsf{tail}_{\mathsf{C}}(\tau)$. Now, since $D \in \mathsf{tail}_{\mathsf{C}}(\tau)$ and $(\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$, by (1) of Definition 28, $D \in \mathsf{core}_{\mathsf{C}}(\sigma)$. Therefore, $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. This ends the proof in this case. The proof in the case (a) is similar. Consider (c). We can proceed like in (b) to obtain $D \in \mathsf{core}_{\mathsf{C}}(\sigma)$. Since σ is blocked, then $\mathsf{core}_{\mathsf{C}}(\sigma) = \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$. Therefore, $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$, thus $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$.

5. Assume $E = \exists R.D$. We have $\exists R.D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. We have to prove that $\mathsf{p} \in (\exists R.D)^{\mathcal{I}}$, i.e. there exists $\mathsf{p}' \in \Delta^{\mathcal{I}}$ such that $\mathsf{p}' \in D^{\mathcal{I}}$ and $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$.

   Since $\mathsf{last}(\mathsf{p})$ is saturated, there exists a triple $\tau \in \mathsf{last}(\mathsf{p})$ such that $R \in \mathsf{tie}(\tau)$ and $D \in \mathsf{tail}_{\mathsf{C}}(\tau)$. Then, τ is not dummy, and, thus, $\Omega(\mathsf{last}(\mathsf{p}), \tau)$ is not empty, i.e. there exists $\sigma \in \mathbf{\Lambda}$ and a triple $\tau' \in \sigma$ such that $(\sigma, \tau') \in \Omega(\mathsf{last}(\mathsf{p}), \tau)$.

   By the definition of Ω, $\mathsf{tail}_{\mathsf{C}}(\tau) = \mathsf{core}_{\mathsf{C}}(\sigma)$. Thus, $D \in \mathsf{core}_{\mathsf{C}}(\sigma)$. There are two cases: σ is blocked or not. Assume that σ is not blocked and let $\mathsf{p}' = \mathsf{p} \cdot \langle \sigma \rangle$. By the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. Also, $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, it follows $\mathsf{p}' \in D^{\mathcal{I}}$. Therefore, $\mathsf{p} \in (\exists R.D)^{\mathcal{I}}$. Assume that σ is blocked and let $\mathsf{p}' = \mathsf{p} \cdot \langle \mathsf{b}(\sigma) \rangle$. By the definition of $\mathcal{I}$, $(\mathsf{p}, \mathsf{p}') \in R^{\mathcal{I}}$. Since $\mathsf{core}_{\mathsf{C}}(\sigma) = \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$, it follows $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{b}(\sigma))$, and, thus, $D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}'))$. By the induction hypothesis, $\mathsf{p}' \in D^{\mathcal{I}}$. Therefore, $\mathsf{p} \in (\exists R.D)^{\mathcal{I}}$.

6. Assume $E = \neg D$. We have $\neg D \in \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$. Assume that D is a concept name. Since $\mathsf{last}(\mathsf{p})$ is clash-free, then $D \notin \mathsf{core}_{\mathsf{C}}(\mathsf{last}(\mathsf{p}))$, and, by the definition of $\mathcal{I}$, it follows $\mathsf{p} \notin D^{\mathcal{I}}$, i.e. $\mathsf{p} \in (\neg D)^{\mathcal{I}}$. If D is not a concept name, we can proceed by induction on the length of D to prove that $\mathsf{p} \in (\neg D)^{\mathcal{I}}$.

□

This shows that an interpretation representing a model for the input ontology can be always constructed from the compressed tableau. The next step is to design a compressed tableau algorithm that builds this compressed tableau. We believe that the compressed tableau algorithm given in Chapter 6 can be adapted for reasoning in $\mathcal{ALCI}+\mathcal{LK}$. Apparently, the matching algorithms have to be changed to perform matching according to the new definition of matching (Definition 28).

Bibliography

[1] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk - a link discovery framework for the web of data. In *LDOW*, 2009.

[2] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes: A time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, page 2312–2317. AAAI Press, 2011.

[3] Fatiha Saïs, Nathalie Pernelle, and Marie-Christine Rousset. L2R: A Logical Method for Reference Reconciliation. In *Twenty-Second AAAI Conference on Artificial Intelligence*, page 2007, Vancouver, British Columbia, Canada, July 2007.

[4] Mustafa Al-Bakri, Manuel Atencia, Jérôme David, Steffen Lalande, and Marie-Christine Rousset. Uncertainty-sensitive reasoning for inferring sameas facts in linked data. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 698–706. IOS Press, 2016.

[5] Norbert Fuhr. Probabilistic datalog: Implementing logical information retrieval for advanced applications. *J. Am. Soc. Inf. Sci.*, 51:95–110, 2000.

[6] Dezhao Song and Jeff Heflin. Automatically generating data linkages using a domain-independent candidate selection approach. In *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, ISWC'11, page 649–664, Berlin, Heidelberg, 2011. Springer-Verlag.

[7] Houssameddine Farah, Danai Symeonidou, and Konstantin Todorov. Keyranker: Automatic rdf key ranking for data linking. In *Proceedings of the Knowledge Capture Conference*, K-CAP 2017, New York, NY, USA, 2017. Association for Computing Machinery.

[8] Manel Achichi, Mohamed Ben Ellefi, Danai Symeonidou, and Konstantin Todorov. Automatic Key Selection for Data Linking. In *EKAW: Knowledge Engineering and Knowledge Management*, volume LNCS of *Knowledge Engineering and Knowledge Management*, pages 3–18, Bologne, Italy, November 2016. Springer International Publishing.

[9] Jérôme Euzenat and Pavel Shvaiko. Ontology matching. In *Springer Berlin Heidelberg*, 2013.

[10] Manuel Atencia, Jérôme David, and Jérôme Euzenat. Data interlinking through robust linkkey extraction. In *Proceedings of the Twenty-First European Conference on Artificial Intelligence*, ECAI'14, page 15–20, NLD, 2014. IOS Press.

[11] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1995.

[12] Jérôme Euzenat Manuel Atencia, Jérôme David. On the relation between keys and link keys for data interlinking. 2012.

[13] Manuel Atencia, Jérôme David, Jérôme Euzenat, Amedeo Napoli, and Jérémy Vizzini. Link key candidate extraction with relational concept analysis. *Discrete Applied Mathematics*, 273:2–20, 2020.

[14] Nacira Abbas, Jérôme David, and Amedeo. Napoli. Discovery of link keys in rdf data based on pattern structures: Preliminary steps.

[15] Franz Baader, Ian Horrocks, Carsten Lutz, and Uli Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[16] Chan Le Duc, Thinh Dong, Myriam Lamolle, and Aurélien Bossard. Raisonnement fondé sur un tableau compressé pour les logiques de description. In *Acte des journées d'intelligence artificielle fondamentale*, 2016. `https://www.supagro.fr/jfpc_jiaf_2016/Articles.IAF.2016/LeDuc_IAF_2016.pdf`.

[17] Database applications semantics.

[18] David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. *Consequence-based Reasoning for Description Logics with Disjunction, Inverse Roles, Number Restrictions, and Nominals*. 2018.

[19] Diego Calvanese, Domenico Carbotta, and Magdalena Ortiz. A practical automata-based technique for reasoning in expressive description logics. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, page 798–804. AAAI Press, 2011.

[20] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, jun 2007.

[21] B. Motik, R. Shearer, and I. Horrocks. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228, Oct 2009.

[22] Volker Haarslev and Ralf Möller. Racer: An owl reasoning agent for the semantic web. *Proceedings of the International Workshop on Applications, Products, and Services of Web-based Support Systems*, 18, 01 2003.

[23] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. Artif. Int. Res.*, 1(1):109–138, dec 1993.

[24] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *KR*, 1998.

[25] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Keys for free in description logics. In Franz Baader and Ulrike Sattler, editors, *Proceedings of the 2000 International Workshop on Description Logics (DL2000), Aachen, Germany, August 17-19, 2000*, volume 33 of *CEUR Workshop Proceedings*, pages 79–88. CEUR-WS.org, 2000.

[26] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, page 155–160, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[27] Carsten Lutz, Carlos Areces, Ian Horrocks, and Ulrike Sattler. Keys, nominals, and concrete domains. *J. Artif. Int. Res.*, 23(1):667–726, jun 2005.

[28] David Toman and Grant E. Weddell. On keys and functional dependencies as first-class citizens in description logics. In *IJCAR*, 2006.

[29] Alexander Borgida and Grant E. Weddell. Adding uniqueness constraints to description logics (preliminary report). In *DOOD*, 1997.

[30] Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, KR'96, page 316–327, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[31] D.S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28(1):167–189, 1984.

[32] Phokion G. Kolaitis, Jonathan Panttaja, and Wang-Chiew Tan. The complexity of data exchange. In *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '06, page 30–39, New York, NY, USA, 2006. Association for Computing Machinery.

[33] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, page 723–731, New York, NY, USA, 2004. Association for Computing Machinery.

[34] Jarkko Kari. On the undecidability of the tiling problem. In *SOFSEM*, 2008.

[35] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *Web Semant.*, 3(1):41–60, jul 2005.

[36] Ullrich Hustadt. Reducing shiq- description logic to disjunctive datalog programs. pages 152–162. AAAI Press, 2004.

[37] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning for description logics around-shiq in a resolution framework. 2004.

[38] Link key extraction under ontological constraints.

[39] Alexander Borgida and Luciano Serafini. Distributed description logics: Directed domain correspondences in federated information sources. In *OTM*, 2002.

[40] Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In *ESWC*, 2005.

[41] Antoine Zimmermann and Chan Le Duc. Reasoning with a network of aligned ontologies. In *RR*, pages 43–57, 2008.

[42] Jérémy Lhez, Chan Le Duc, Thinh Dong, and Myriam Lamolle. Decentralized Reasoning on a Network of Aligned Ontologies with Link Keys. In *JIAF 2019 - 13èmes Journées d'Intelligence Artificielle Fondamentale*, pages 1–10, Toulouse, France, July 2019.

[43] Christian Meilicke, Heiner Stuckenschmidt, and Andrei Tamilin. Improving automatically created mappings using logical reasoning. In *Proceedings of the 1st International Conference on Ontology Matching - Volume 225*, OM'06, page 61–72, Aachen, DEU, 2006. CEUR-WS.org.

[44] C. Meilicke, H. Stuckenschmidt, and Andrei Tamilin. Repairing ontology mappings. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, page 1408–1413. AAAI Press, 2007.

[45] Antoine Zimmermann and Chan Le Duc. Reasoning on a network of aligned ontologies. In Georg Lausen Diego Calvanese, editor, *Proc. 2nd International conference on web reasoning and rule systems (RR)*, volume 5341 of *Lecture notes in computer science*, pages 43–57, Karlsruhe, Germany, November 2008. Springer Verlag. zimmermann2008b.

[46] Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient owl reasoner. In *in Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED*, pages 26–27.

[47] Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. Owllink. *Semant. Web*, 2(1):23–32, jan 2011.

[48] Chan Le Duc, Myriam Lamolle, Antoine Zimmermann, and Olivier Curé. Draon: A distributed reasoner for aligned ontologies. In *Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013), Ulm, Germany, July 22, 2013*, pages 81–86, 2013.

[49] Giuseppe De Giacomo, Francesco Donini, and Fabio Massacci. Exptime tableaux for alc. pages 107–110, 01 1996.

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

[51] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.

[52] Maroua Gmati, Manuel Atencia, and Jérôme Euzenat. Tableau extensions for reasoning with link keys. In *11th ISWC workshop on ontology matching (OM)*, pages 37–48, Kobe, Japan, October 2016. No commercial editor. gmati2016a.

[53] Manfred Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In *Proc. 1st conference on the Principles of Knowledge Representation and Reasoning (KR)*, pages 421–431. Morgan Kaufmann, 1989.

[54] Franz Baader, Martin Buchheit, and Bernhard Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1-2):195–213, 1996.

[55] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In *Proceedings of the International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 1999)*. Springer, 1999.

[56] Vaughan R. Pratt. A practical decision method for propositional dynamic logic. In *Proceedings of the tenth annual ACM symposium on Theory of Computing*, pages 326–337, 1978.

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

[58] Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. In *Proceedings of the International Workshop on Description Logics*, pages 37–48. CEUR Workshop, 1996.

[59] Matthew Horridge and Sean Bechhofer. The owl api: A java api for working with owl 2 ontologies. In *Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529*, OWLED'09, page 49–58, Aachen, DEU, 2009. CEUR-WS.org.

[60] Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. The alignment api 4.0. *Semant. Web*, 2(1):3–10, jan 2011.

[61] Ontology alignment evaluation initiative.