QUESTIONS AND ANSWERS: REASONING AND QUERYING IN DESCRIPTION LOGIC

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

April 2001

By
Sergio Tessaris
Department of Computer Science

Contents

Abstract

Description Logics (DLs) are a family of formal languages for describing complex structured classes. These languages contain boolean operators and quantification over class attributes, as well as the specification of elements of the classes and their properties. DL knowledge bases (KBs) consist of a terminological part which describes the general organisation of the classes, and an assertional part for describing the properties of individuals. In spite of its inherent intractability, practical algorithms have been recently developed for purely terminological reasoning; when individuals are introduced there is still a lack of results.

This thesis constitutes an advance in the direction of the development of DL systems providing efficient and powerful reasoning in presence of individuals. We choose an expressive DL ($\mathcal{SH}f$), which has been previously studied from the terminological perspective (i.e. without individuals), and we investigate the problem of reasoning with individuals.

$\mathcal{SH}f$ extends the standard DL $\mathcal{ALC}$ with transitive roles, role hierarchy, and attributes. This extension enables the representation of general inclusion axioms using a technique called *internalisation*. This thesis investigates two automated reasoning problems: KB satisfiability and query answering.

KB satisfiability is the fundamental problem of deciding whether a given KB does not contain any contradiction. This problem is not only important for delivering consistent KBs, but most of the reasoning tasks for DL systems can be reduced to KB satisfiability. We show how the technique of *precompletion*, which has previously only be used in less expressive DLs, can be used in our expressive DL.

A serious shortcoming of many Description Logic based knowledge representation systems is the inadequacy of their query languages. We present a novel technique that can be used to provide an expressive query language for such systems. One of the main advantages of this approach is that, being based on a reduction to knowledge base satisfiability, it can easily be adapted to most existing (and future) Description Logic implementations. We believe that providing Description Logic systems with an expressive query language for interrogating the knowledge base will significantly increase their utility.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Acknowledgements

I would like to thank all the people who helped and inspired me during the period of my PhD: my supervisors (in order of appearance) Carole Goble, Graham Gough, and Ian Horrocks, the present and past members of Information Management Group, and several other people in the Department of Computer Science (the list would be too long).

I am particularly grateful to Ian Horrocks and Graham Gough for their support, both scientific and human. Moreover, without their tireless proofreading this thesis would have been hardly readable.

I would like to mention Enrico Franconi as the principal cause for my involvement in the DL community, and to thank him for his advice.

This work has been supported by a grant from the Engineering and Physical Sciences Research Council.

Chapter 1

Introduction

Description logics are knowledge representation systems evolved from early semantic networks and frame systems. They have been developed as a logical reconstruction of the KL-ONE–like knowledge representation systems (see Woods and Schmolze [1992]). They differ from their ancestors in that they place a strong emphasis on a precise semantic characterisation of the modelling language. The need for a unequivocally defined semantics stems from the fact that having a clear semantics enables the description and justification of the automated deduction processes.

Description logics have been proved effective in several application fields including automated language understanding and generation, configuration and data modelling (see Franconi [1994], Calvanese et al. [1998b]). In addition, their formal semantics enables the exact characterisation of the expressiveness of a DL system against its computational properties.

Although good results have recently been obtained in the development of optimised algorithms for DL knowledge bases without individuals, little progress has been made with reasoning involving individuals. This research is an attempt to fill this gap, by exploring the possibility of developing optimised algorithms.

A serious shortcoming of many Description Logic based knowledge representation systems is the inadequacy of their query language. In this thesis we present a novel technique that can be used to provide an expressive query language for such systems.

This chapter presents an overview of background of the thesis, together with the overall description of the document.

1.1 Background

1.1.1 Logic based knowledge representation

This thesis concerns knowledge representation systems that represent the domain using a formal language. In these systems, a precise semantics unambiguously defines the meaning of a knowledge base written using the language. In addition, the knowledge representation system provides a set of automated reasoning services to manipulate and/or interrogate knowledge bases.

In particular it is assumed that the knowledge can be partitioned into *terminological* and *assertional* parts. The terminological part (or simply terminology) describes the structural properties of the terms of the domain, while the assertional part depicts a particular configuration of the domain by introducing individuals and asserting their properties using the definitions in the terminology.

Example 1.1

Terminology	Birds	**are**	Animals.
	Penguins	**are**	Birds **and not** Flyers.
Assertions	tweety	**is-a**	Bird **and not** Flyer **and**
			has FRIEND **which-is** Flyer.

This example shows a simple knowledge base written using a natural language–like syntax. The terminological part defines the properties of Birds and Penguins, while the assertional part states some properties of a particular individual (`tweety`). This individual is described as a bird which cannot fly, having at least one friend which is a flyer.

Partitioning knowledge in this way is quite a common methodology in computer science. For instance, in the database setting the distinction between schemata and an actual database could be seen as the dichotomy between the terminology and the assertional part. Generally speaking there are two main reasons to adopt the partitioning technique; firstly, the same terminology could be used for several knowledge bases, and secondly, different algorithms can be used on each part, exploiting the different characteristics of the two components.

Two other important features of knowledge representation systems are the possibility of adopting an *open world* semantics for the knowledge base (see Reiter [1977]), as well as the ability to represent incomplete information.

The adoption of an open world semantics leaves a degree of uncertainty regarding facts that are not explicitly asserted. For example, individuals not mentioned in the knowledge base could be part of the domain and each individual mentioned could have more properties than the asserted ones. Referring to Example 1.1, `tweety` could either be a penguin or not, and there could also be additional birds or animals in that little world.

Incomplete information can appear in different forms. Firstly, the implied existence of individuals without explicitly mentioning them: the knowledge base in Example 1.1 implies the existence of at least one flyer which is friend of `tweety`. Secondly, in the form of disjunctive information, for example it could be specified that `tweety`'s friend should be either an auk or a puffin.

Description logics knowledge representation systems take advantage of the above properties. The next section gives an overview of the main features of this Description Logic formalism, while a more formal description is provided in Chapter 2.

1.1.2 Description logics

There are several ways to give an intuition about what a description logic is. For a knowledge representation audience, DLs could be viewed as a logical reconstruction of frame systems and semantic networks. An object oriented database audience can look at DLs as the static (declarative) part of a data definition language, while a logician could look at them as a syntactic variant of a multi-modal modal logic. Given this intuition, dipping into the literature reveals a strong emphasis on the formal logic nature of DLs (see KRSS). In fact a DL consists of a formal language, described by a clear set theoretical semantics, together with a calculus (see Chapter 2).

Language expressions of a DL describe classes (concepts) of individuals that share some properties. Properties can also be specified by means of relations (roles) between individuals. The language is compositional, i.e. the concept descriptions are built by combining different subexpressions using constructors. For instance, if the concepts `STAFF` and `TECHNICIAN` are defined, the notion of technical staff can be represented by putting together the two concepts in the expression $\mathtt{STAFF} \sqcap \mathtt{TECHNICIAN}$. The kind of expressions that can be built using a DL language are

$$\mathtt{Bird} \sqcap \neg\mathtt{Flyer} \sqcap (\exists \mathtt{FRIEND}.\mathtt{Flyer})$$
$$\mathtt{Vehicle} \sqcap (\exists \mathtt{PART\text{-}OF}.\mathtt{Engine}) \sqcap (\geq 2\, \mathtt{PART\text{-}OF}.\mathtt{Wheel})$$

where elements in typewriter style are terms of the language (concepts in small letters and roles in capital letters), while $\sqcap$, $\neg$, $\exists$ are language constructors.[1]

The semantics of the language is given in a set theoretical way, over a *domain* which is a set of elements. A concept expression corresponds to a subset of the domain, while a role expression corresponds to a binary relation over the domain. As their shape suggests, some of these constructors have a strong relationship with boolean operators and logical quantifiers.

A DL knowledge base consists of a terminology, traditionally called the Tbox, and an assertional part, called Abox. The Tbox contains the definitions of the terms (concept definitions), while the Abox contains a set of membership and role assertions. Membership assertions relate an individual to a concept, stating that the individual involved is an instance of the concept. Role assertions state that two individuals are linked by a given role. Example 1.1 could be expressed by the DL knowledge base in Example 1.2.

Example 1.2

Tbox
$$\begin{array}{lcl}\texttt{Bird} & \sqsubseteq & \texttt{Animal}\\ \texttt{Penguin} & \sqsubseteq & \texttt{Bird} \sqcap \neg\texttt{Flyer}\end{array}$$

Abox $\texttt{tweety}{:}(\texttt{Bird} \sqcap \neg\texttt{Flyer} \sqcap (\exists\texttt{FRIEND}.\texttt{Flyer}))$

Reasoning with DL knowledge bases is a deduction process which extracts not only the facts explicitly asserted in a knowledge base, but also their logical consequences. For instance, from the knowledge base of Example 1.2 can be derived the fact that `tweety` is an `Animal`. When only the terminology is involved in a deduction, the reasoning is said to be *terminological*; otherwise it is said to be *hybrid*.

DL systems provide automated reasoning services for interacting with the knowledge base. Typical reasoning services provided are: verifying the non–contradiction of assertions in the knowledge base, checking that an individual satisfies a concept expression, retrieval of all the individuals satisfying a concept expressions and subsumption checking, which verifies whether one expression is more general than another.[2] Nevertheless, it has been proved that all services can be reduced in polynomial time to the problem of knowledge base satisfiability (see Schaerf [1994]), that is the checking for the absence of contradicting assertions in a given knowledge base.

[1]The meaning of the different elements of the language will be clarified in Chapter 2.

[2]An expression is more general than another if its corresponding set always includes that of the second concept.

The term *complexity of reasoning* in this context refers to the complexity of the knowledge base satisfiability problem, and it characterises the computational properties of a DL system. The complexity actually depends on the expressiveness of the underlying DL language. Moreover, the language (and its expressiveness) is defined by the kinds of constructors it provides. Therefore the complexity is determined by the constructors provided by the language.

One example of the influence of the constructors on the computational properties is the interaction of the role conjunction constructor with the number restriction constructor. The role conjunction constructor builds a new role expression from the intersection of atomic roles; while number restriction constructor constrains the number of individuals related by a given role. These two kinds of constructor do not cause any problem separately, but when they are taken together (i.e. allowing role conjunction in the role of the number restriction) they cause the undecidability of the KB satisfiability problem (see Baader and Sattler [1996]).

Different applications need different language constructors, and the modularity of the language facilitates the addition of new constructors. The result is that there is not a single DL language, but a family of different languages. A big effort has been spent in extending the expressiveness of DL languages, while maintaining the soundness and decidability of the reasoning services (see for example Buchheit et al. [1993]).

Nowadays the undecidability boundaries and the complexity of the reasoning are well understood on the theoretical side (see Donini et al. [1997]). However there is still a lot work to do on the definition of “practical” algorithms, and the analysis of the interaction between the different constructors is still an active topic in the DL community (see Franconi et al. [1998a]).

1.2 Reasoning with DL knowledge bases

Description logics have proved to be effective in various applications, both those that require an assertional part and those that do not. In fact, while DLs were initially developed to be the knowledge representation part of an AI system, recent work has showed that their ability to reason in an abstract way about a domain[3] could be extremely useful for knowledge modelling tasks. For example, DLs could be used as tools for checking the consistency of entity–relationship or object oriented schemata (see Calvanese et al. [1998b]).

[3]Without the necessity to populate the domain with individuals.

Recent years have seen significant advances in the design of sound and complete reasoning algorithms for DLs with both expressive logical languages and unrestricted Tboxes, i.e., those allowing arbitrary concept inclusion axioms (see Baader [1991], Horrocks and Sattler [1999], De Giacomo and Massacci [1998]). Moreover, systems using highly optimised implementations of (some of) these algorithms have also been developed, and have been show to work well in realistic applications (see Horrocks [1998], Patel-Schneider [1998]). While most of these have been restricted to terminological reasoning (i.e., the Abox is assumed to be empty), attention is now turning to the development of both algorithms and (optimised) implementations that also support Abox reasoning (see Haarslev and Möller [1999], Tessaris and Gough [1999]).

This research starts with the premise that there are still several key applications for a DL hybrid system. Generally speaking, applications that require the ability to state incomplete knowledge about the individuals in the domain are good candidates for the use of a hybrid DL; one example is natural language processing (see Franconi [1994]). The optimisations recently applied to Tbox algorithms (see Horrocks and Patel-Schneider [1998a]) give pointers towards the improvement or redesign of current Abox algorithms in order to enhance performance in realistic applications.

In this thesis we discuss techniques and algorithms used to deal with two essential reasoning tasks: knowledge base satisfiability and query answering. We focus on a particular DL language, which is implemented in the terminological system[4] FaCT (see Horrocks [1998]). This DL, called $\mathcal{SH}f$,[5] possesses most of the relevant features of an expressive DL language. In addition, experience with the FaCT system has shown that $\mathcal{SH}f$ is well suited for knowledge representation modelling.

Several techniques have been proposed for the verification of the satisfiability of a KB, and many of them have lead to the development of DL systems. However, some of these techniques have been developed for DLs less expressive than is required for a modern system. We investigate the extension of one of these techniques, the so called *precompletion technique* (see Hollunder [1996]), for reasoning with $\mathcal{SH}f$.

Although modern DL systems provide sound and complete Abox reasoning for very expressive logics, their utility is limited compared to earlier DL systems by their very weak Abox query languages. Typically, these only support instantiation (is an

[4]DL system without the Abox.

[5]$\mathcal{SH}f$ has been previously called $\mathcal{ALC}f\mathcal{H}_{\mathcal{R}^+}$, but the new name follows a more compact naming scheme recently introduced in Horrocks et al. [1999b].

individual i an instance of a concept C), realisation (what are the most specific concepts i is an instance of) and retrieval (which individuals are instances of C). This is in contrast to a system such as Loom where a full first order query language is provided, although based on incomplete reasoning algorithms (see MacGregor and Brill [1992]).

We present a technique for answering conjunctive queries over DL KBs. This work has been inspired by the use of Abox reasoning to decide conjunctive query containment (see Horrocks et al. [1999a], Calvanese et al. [1998a]). However, the characteristics of $\mathcal{SH}f$ forces the use of an algorithm which is much more complicated of those presented in these previous works.

Our main contributions can be summarised in the following points.

- The extension of the already known *precompletion technique* for KB satisfiability to the DL language $\mathcal{SH}f$. Previously the technique was applied to DLs without general inclusion axioms, transitive roles, or role hierarchies.

- A query answering algorithm for disjunctions of conjunctive queries for $\mathcal{SH}f$ KBs with Aboxes.

- A prototype software system performing KB satisfiability for $\mathcal{SH}f$ has been developed. The system has been used for verifying the practical feasibility of the precompletion technique, and the impact of different optimisations on the performance of the system.

- A characterisation of the class of models of the logic $\mathcal{SH}f$. This was previously done (implicitly) for the problem of KB satisfiability, but to our knowledge, it is the first result for the query answering.

The next section gives an overview of the structure of this thesis.

1.3 Thesis outline

The thesis contains both introductory and technical chapters, in particular the reader can get an overview of the work presented by reading the even numbered chapters only.

Chapter 2 gives a formal description of the Description Logics in general, and in particular the DL $\mathcal{SH}f$ which is used in the following chapters.

Chapter 3 describes the class of interpretations which characterise the logic $\mathcal{SH}f$. The results from this chapter are used in the proofs presented in Chapter 5 and Chapter 7. This model–theoretic characterisation of $\mathcal{SH}f$ is not only essential for most of the proofs, but it provides an insight on the expressiveness of $\mathcal{SH}f$.

Chapter 4 and Chapter 5 investigate the problem of Knowledge Base satisfiability for $\mathcal{SH}f$. The first gives an overview of the methods which have been used for tackling the problem, and provides an introduction to the technique we investigated. Chapter 5 provides a formal description of the precompletion algorithm, and shows the proofs for its correctness and completeness.

Chapter 6 and Chapter 7 introduce and provide a solution to the problem of answering conjunctive queries over $\mathcal{SH}f$ knowledge bases. The first chapter gives an intuitive presentation of the problem and our solution; while Chapter 7 provides the full details.

Chapter 8 describes the experiments we performed with an implementation of the algorithm for KB satisfiability. Finally, Chapter 9 draws conclusions from this work and presents further lines of research suggested by the results achieved so far.

Chapter 2

Preliminaries

In this chapter *Description Logics* (DLs) are formally introduced, together with the description of what we intend by the term *DL based system*. In Section 2.1 we present the syntax and semantics of the logics belonging to the class of DLs, followed by the description of DL based systems. Section 2.2 introduces common reasoning problems over DL knowledge bases (KBs), and the automation of the reasoning itself. Section 2.3 describes an extension of DL KBs with the introduction of axioms on roles. Finally, we describe the DL language we investigate, and which will be used towards this thesis.

2.1 Description Logics

The core of Description Logics is the concept language, a formal language designed to describe classes and relationships between elements of the classes. DL based knowledge bases are built using concept languages expressions, and they are usually divided in two distinct parts: intensional and extensional. The intensional part describes the general schema of the classes and relationships, while the extensional part constitutes a (partial) instantiation of the schema, since it contains assertions about a set of *individuals*. Historically the intensional part takes the name of terminology or Tbox, and the extensional part the name of assertional part or Abox. Queries on a DL knowledge base are answered by an inferential process involving both the Tbox and Abox parts. The answer to a query is deduced as logical consequences of the content of the knowledge base according to the formal semantics.

2.1.1 Syntax and Semantics

All description logic systems are based on a common family of languages, called *concept languages*, for describing structured classes of objects. The foundations of concept languages are *concepts* and *roles*: a concept represents a class of objects sharing some common characteristics, while a *role* represents a binary relation between objects or, in other words, attributes attached to objects.

The language is completely described by a formal syntax and a Tarsky-like semantics. A formal definition of the language is essential for knowledge bases characterisation, and for the definition of reasoning services.

A concept language is composed of an alphabet of distinct concept names ($\mathcal{CN}$), role names ($\mathcal{RN}$) and individual names ($\mathcal{O}$); together with a set of constructors for building concept and role expressions. Concept expressions (or simply concepts) describe subsets of the domain; for example $\texttt{Motorcycles} \sqcup \texttt{Cars}$ denotes the union of the elements in `Motorcycles` and in `Cars`. Describing the abstract syntax, concept names are indicated by letter A or B, role names by P, and individual names by lowercase letters a, b. Concept expressions are indicated by letters C or D, and role expressions by R or Q.

Description logics form a family of different logics, distinguished by the set of constructors they provide. Each language is named according to a convention introduced in Schmidt-Schauss and Smolka [1991]: each constructor is associated to a different capital letter (e.g. $\mathcal{U}$ to disjunction and $\mathcal{C}$ to negation) and the name of a language is composed by the prefix $\mathcal{AL}$ (acronym for attributive language) followed by the letters correspondent to the constructors in the language (e.g. $\mathcal{ALU}$ or $\mathcal{ALC}$). One of the reasons for having such a fine granularity on the classification of concept languages is that the computational properties of the reasoning changes dramatically with the presence or absence of a particular constructor; therefore the ability to indicate precisely the constructors which are in a DL is very useful.

The very basic language denoted by the prefix $\mathcal{AL}$ is close to the expressivity of frame based representation systems. It enables the specification of hierarchies of concepts by means of the conjunction of two concepts ($\sqcap$). The expression $C \sqcap D$ denotes the intersection of the elements in C and in D. The hierarchy comes from the fact that $C \sqcap D$ is more specific than both C and D, because it denotes a smaller set of elements. A second group of constructors in $\mathcal{AL}$ enables the specification of attributes by the unqualified existential ($\exists.R$) and qualified universal ($\forall R.C$) quantifiers. These expressions build new concepts in terms of the roles: expression $\exists.R$ denotes the set of elements related to some other element by the role R, while $\forall R.C$ denotes the set of elements which are related by the role R exclusively to elements of the concept C. Using frame systems nomenclature, the existential quantifier represents the fact of having attribute R, while the universal quantifier specifies the type restriction of that attribute. In addition, the concept language $\mathcal{AL}$ has the ability to represent the complement of concept names by the expression $\neg A$,[1] and it provides the symbols $\top$ and $\bot$ for the full domain and the empty set respectively.

The semantics of DL constructors is defined in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of a nonempty domain $\Delta^{\mathcal{I}}$ and a interpretation function $\cdot^{\mathcal{I}}$. The interpretation function maps concept names into subsets of the domain, role names into subsets of the cartesian product of the domain ($\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$), and individual names into elements of the domain. The only restriction on the interpretations is the so called unique name assumption, which imposes that different individual names must be mapped into distinct elements of the domain. Given a concept name A (role name P) the set denoted by $A^{\mathcal{I}}$ ($P^{\mathcal{I}}$) is called the interpretation, or extension, of A (R) w.r.t. $\mathcal{I}$.

Example 2.1
For example, given the concept names `Even`, `Odd`, the role `SUCC`, and the individuals `one`,`two`,`three` and `four`. A possible interpretation is composed by the natural numbers $\Delta^{\mathcal{I}} = \{1, 2, 3, 4, 5\}$ and the interpretation function defined by:

$$\begin{aligned}
\texttt{Odd}^{\mathcal{I}} &= \{1, 3, 5\} \\
\texttt{Even}^{\mathcal{I}} &= \{2, 4\} \\
\texttt{SUCC}^{\mathcal{I}} &= \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\} \\
\texttt{one}^{\mathcal{I}} &= 1 \\
\texttt{two}^{\mathcal{I}} &= 2 \\
\texttt{three}^{\mathcal{I}} &= 3 \\
\texttt{four}^{\mathcal{I}} &= 4
\end{aligned}$$

Note that at this stage there is no way to say that an interpretation is correct or wrong. In this example the interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ "looks correct" because of the natural language semantics we associate to the names, but a different interpretation where the role `SUCC` is mapped to $\{(1, 1), (3, 3)\}$ would be a valid interpretation as well.

[1] But not the negation of a general concept expression.

Syntax	Semantics	Description
A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	concept name
$\top$	$\Delta^{\mathcal{I}}$	top
$\bot$	$\emptyset$	bottom
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	conjunction
$\forall R.C$	$\{x \mid \forall y (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$	universal quantification
$\exists R.C$	$\{x \mid \exists y (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	existential quantification ($\mathcal{E}$)
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	general negation ($\mathcal{C}$)
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	disjunction ($\mathcal{U}$)
$\leq n\, R$	$\{x \mid \sharp \{y \mid (x,y) \in R^{\mathcal{I}}\} \leq n\}$	number restriction ($\mathcal{N}$)
$\geq n\, R$	$\{x \mid \sharp \{y \mid (x,y) \in R^{\mathcal{I}}\} \geq n\}$	
$\leq n\, R.C$	$\{x \mid \sharp \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \leq n\}$	qualified number restriction ($\mathcal{Q}$)
$\geq n\, R.C$	$\{x \mid \sharp \{y \mid (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \geq n\}$	
$\{\, a_1, \ldots, a_n \,\}$	$\{{a_1}^{\mathcal{I}}, \ldots, {a_n}^{\mathcal{I}}\}$	`one-of` ($\mathcal{O}$)

Table 2.1: Syntax and Semantics of concept expression constructors.

Syntax	Semantics	Description
P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	role name
$R \sqcap Q$	$R^{\mathcal{I}} \cap Q^{\mathcal{I}}$	role conjunction ($\mathcal{R}$)
$R \sqcup Q$	$R^{\mathcal{I}} \cup Q^{\mathcal{I}}$	role disjunction
R^{-1}	$\{(y,x) \mid (x,y) \in R^{\mathcal{I}}\}$	converse role ($\mathcal{I}$)
$R \circ Q$	$R^{\mathcal{I}} \circ Q^{\mathcal{I}}$	role composition
$C?$	$\{(x,x) \mid x \in C^{\mathcal{I}}\}$	test
R^*	$\bigcup_{i \geq 0} (R^{\mathcal{I}})^i$	reflexive transitive closure: $(R^{\mathcal{I}})^0 = ((\exists R.\top)?)^{\mathcal{I}}$, and $(R^{\mathcal{I}})^{i+1} = (R^{\mathcal{I}})^i \circ R^{\mathcal{I}}$

Table 2.2: Syntax and Semantics of role expression constructors.

Interpretations can be naturally seen as directed labeled graphs; the nodes are the elements of the interpretation domain, while the interpretation function provides both the labeled edges and the set of node labels. For example, the interpretation $\mathcal{I}$ of Example 2.1 is represented by the graph

$$\begin{tikzcd}
\overset{\{\mathtt{Odd}\}}{1} \arrow[r, "\{\mathtt{SUCC}\}"'] \arrow[d, dotted, no head] & \overset{\{\mathtt{Even}\}}{2} \arrow[r, "\{\mathtt{SUCC}\}"'] \arrow[d, dotted, no head] & \overset{\{\mathtt{Odd}\}}{3} \arrow[r, "\{\mathtt{SUCC}\}"'] \arrow[d, dotted, no head] & \overset{\{\mathtt{Even}\}}{4} \arrow[r, "\{\mathtt{SUCC}\}"'] \arrow[d, dotted, no head] & \overset{\{\mathtt{Odd}\}}{5} \arrow[llll, bend right, "\{\mathtt{SUCC}\}"'] \\
\mathtt{one} & \mathtt{two} & \mathtt{three} & \mathtt{four} &
\end{tikzcd}$$

Note that the node and edge labels are sets of concept and role names respectively, because a single element of the domain can be in the extension of more than one concept name (and analogously a pair of elements in the extension of several roles). In the graph we indicated the mapping for the individual names with the dotted lines, but in general this will be omitted when we are interested in the structure of the interpretation only.

Expressions are interpreted according the semantics given in Table 2.1 and Table 2.2; for instance, with respect to the Example 2.1 the expression $(\exists\mathtt{SUCC}.\mathtt{Even})^{\mathcal{I}}$ is interpreted as the set $\{1,3\}$. The interpretation of a concept expression can be the empty set as well; for instance, with respect to the interpretation of Example 2.1 the set $(\mathtt{Even} \sqcap (\exists\mathtt{SUCC}.\mathtt{Even}))^{\mathcal{I}}$ is empty.

An interpretation is said to be a model for a concept expression if the extension of that expression is nonempty. The existence of a model for a concept expression defines the satisfiability of the concept itself; i.e. a concept expression C is satisfiable if and only if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}}$ is nonempty.

Different concept expressions can be compared with respect to their models. Two concepts are equivalent if and only if their extensions are equal with respect to every interpretation. In a similar way, a concept is subsumed by another if the extension of the first one is included in that of the second with respect to every interpretation. Formally, concept C is subsumed by concept D (written as $C \sqsubseteq D$) if and only if for any interpretation $\mathcal{I}$ the inclusion $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds.

2.1.2 DL based systems

Traditionally, the knowledge base of a DL-based system is composed of two distinct parts: the intensional (Tbox) and the extensional (Abox) components.

Terminology

The intensional part describes the relation between concepts and roles expressions. It can be seen under two different lights: as a collection of definitions for role and concept names, or as a set of axioms that restrict the models for the knowledge base. The Tbox is composed of a set of statements of the forms:

$$C \doteq D \tag{2.1}$$
$$C \sqsubseteq D \tag{2.2}$$

The first statement asserts that the concept expressions C and D are equivalent, while the second that concept expression C is more specific that (or included in) expression D. When the left hand side of the statement is a concept name ($A \doteq D$ or $A \sqsubseteq D$), the statement is said to be definitional, because it defines the characteristics of the concept name. Examples of general statements are $(\texttt{Odd} \sqcap \texttt{Even}) \sqsubseteq \bot$ and $\top \sqsubseteq (= 1\, \texttt{SUCC})$, while $\texttt{Integer} \doteq (\texttt{Odd} \sqcup \texttt{Even}) \sqcap (= 1\, \texttt{SUCC})$ is a definition.

The semantics of Tbox statements is given in terms of interpretations analogously to the subsumption and equivalence seen in Section 2.1.1. An interpretation $\mathcal{I}$ satisfies (is a model of) the statement $C \doteq D$ if and only if $C^{\mathcal{I}} = D^{\mathcal{I}}$, and it satisfies $C \sqsubseteq D$ if and only if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. The generalisation to a full Tbox is straightforward, an interpretation is a model for a Tbox if it satisfies all the statements contained in the Tbox.

DL systems are also characterised by the kind of statements they allow on the Tbox. Systems with free Tboxes allow any kind of assertions, others restrict to the definitional statements or to acyclic assertions only. A terminological cycle in a Tbox is a recursive definition, or one or more mutually recursive definitions. Examples of recursive definitions are $\texttt{Human} \sqsubseteq \forall \texttt{OFFSPRINGS}.\texttt{Human}$ or the pair of definitions

$$\left\{ \begin{array}{rcl} \texttt{Odd} & \doteq & (\neg\texttt{Even} \sqcap \forall\texttt{SUCC}.\texttt{Even}), \\ \texttt{Even} & \sqsubseteq & \forall\texttt{SUCC}.\texttt{Odd} \end{array} \right\}.$$

Allowing terminological cycles in the Tbox rises several problems, on both the semantic and computational aspects (see Nebel [1991]). For this reason, most of the early DL systems do not allow cycles in the terminology (see Borgida and Patel-Schneider [1994], Baader and Hollunder [1991b]). On the other hand, recursive definitions are ubiquitous in computer science (e.g. data structures), and a very natural way to model application domains (e.g. the `Human` definition above). For these reasons recursive definitions have been widely studied (see Nebel [1991], De Giacomo and Lenzerini [1997]), and all modern systems provide unrestricted concept inclusion assertions.

The first problem for cyclic definitions is providing a semantics; the different proposed semantics are either *fixed point based* or *descriptive*. Using the descriptive semantics, all the interpretations satisfying the Tbox statements are models for the Tbox itself. While under the fixed point semantics, interpretations are filtered according a fixed point operator (see Nebel [1991], De Giacomo and Lenzerini [1997] for more details). The descriptive semantics is adopted here because of its wide acceptance as the most appropriate one (see Donini et al. [1996b]).

We distinguish three different types of terminologies: definitional acyclic, definitional, and free. The latter (free) type is the most general one: every kind of terminological axiom of the form $C \sqsubseteq D$ can be put in the terminology[2]. The definitional approach restricts the form of the axioms in such a way that only concept names are allowed on the left hand side of an axiom, and each name can appear only once in a left hand side. Under the definitional restriction the axioms are called definitions.[3] The most restrictive approach imposes that the definitions must be acyclic.

Many systems take the acyclic definitional approach because it is the simplest case, and it exhibits good computational properties. In addition, it can be shown that a knowledge base containing only acyclic definitions can be transformed in an equivalent one with an empty terminology.

The transformation proceeds in two phases. Firstly, all the inclusion definitions of the form $A \sqsubseteq C$ are transformed into equivalent equality definitions $A \doteq C \sqcap A^*$, introducing a new concept name for each inclusion. Finally, all the defined concept names in concept expressions are substituted with their definition. After the transformation the Tbox can be ignored and all the reasoning can be carried out with the fully expanded expressions.

[2]The inclusion axiom is general enough for representing definitions ($C \doteq D$) as well. In fact, a single definition $C \doteq D$ can be expressed by the pair of inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

[3]Actually, in DLs which provide the disjunctive constructor, free terminologies can always be transformed in (possibly cyclical) definitional terminologies (see Buchheit et al. [1993]).

Abox assertions

The extensional part describes a particular configuration of a domain, introducing individual names and their properties. Aboxes are composed of statements of the form:

$$a{:}C \tag{2.3}$$

$$\langle a, b\rangle{:}R \tag{2.4}$$

The assertion $a{:}C$ expresses the fact that the interpretation of individual a is in the extension of a concept expression C. The role assertion $\langle a, b\rangle{:}R$ states that the pair composed by the interpretations of individuals a and b is in the interpretation of role expression R. The semantics of an assertion is given with respect to interpretations. An interpretation $\mathcal{I}$ satisfies $a{:}C$ ($\langle a, b\rangle{:}R$) if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ ($(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). Analogously to the Tbox, an interpretation $\mathcal{I}$ is a model for an Abox if it satisfies every assertion in the Abox.

DL knowledge bases

A DL knowledge base is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A}\rangle$, where $\mathcal{T}$ is a Tbox and $\mathcal{A}$ an Abox. Given the definition of models for a Tbox and an Abox, a model for a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A}\rangle$ is a model for both $\mathcal{T}$ and $\mathcal{A}$. The definition of models of a knowledge base naturally introduces the concept of logical implication: a statement α is a logically implied (or entailed) by the knowledge base Σ (written as $\Sigma \models \alpha$) if and only if α is true in every model of Σ. For example the statement $b{:}B$ is a logical implication of the knowledge base $\langle \emptyset, \{a{:}\forall R.B, \langle a, b\rangle{:}R\}\rangle$.

Two knowledge bases are equivalent if the models for one are models for the other and vice versa. This equivalence is a powerful tool for manipulating knowledge bases: as long as the equivalence is guaranteed, a knowledge base can be modified without affecting its logical implications. For example the knowledge base $\Sigma = \langle \{A \doteq B \sqcap \exists R.C\}, \{a{:}A\}\rangle$ is equivalent to $\Sigma' = \langle \{A \doteq B \sqcap \exists R.C\}, \{a{:}B \sqcap \exists R.C\}\rangle$, and the statement $a{:}B$ is logically implied by both Σ and Σ'.

2.2 Reasoning with DLs

Reasoning with DL knowledge bases is the fundamental process of discovering the facts entailed by the knowledge base. The interaction with a DL system is performed by a collection of *reasoning services* implemented employing automated reasoning techniques. The reasoning services may differ according to the purpose of the DL system, for example checking the truth value of boolean queries, or verifying the consistency of the knowledge base.

The reasoning services provided by a DL system are formally defined in terms of the logical consequence. Reasoning services can be classified as *basic* services, which involve the checking of the truth value for a statement, and *complex* services. Among complex reasoning services are tasks such as finding all the individuals being in a concept expression, or organising the concept names appearing in the terminology in a taxonomy. Basic services are for instance the verification of the subsumption between two concepts or the satisfiability of a concept. The principal basic services are:

Knowledge base satisfiability written as $\Sigma \models \top \not\equiv \bot$, is the problem of checking whether there is at least a nonempty model for Σ.

Concept satisfiability written as $\Sigma \models C \not\equiv \bot$, is the problem of checking whether there exists a model of Σ in which the extension of the concept C is not empty.

Subsumption written as $\Sigma \models C \sqsubseteq D$, is the problem of verifying that in every model of Σ the extension of C is included in that of D.

Instance Checking written as $\Sigma \models a{:}C$, is the problem of verifying that in every model of Σ the interpretation of a is in the extension of C.

Reasoning services are not independent of one another, on the contrary each can be often reduced to another. The reducibility actually depends on the underlying concept language: using a language closed under negation, all the basic reasoning services can be reduced to knowledge base satisfiability (see Schaerf [1994] for the details).

The complex reasoning tasks provided vary from system to system, and are defined on top of the basic services. The most common are classification and retrieval. Classification consists of explicitly representing the concept taxonomy entailed by the knowledge bases. The concept taxonomy is a graph whose nodes are the concept names that appear in the knowledge base, and the edges represent the subsumption relation between them. This graph can be constructed by checking the subsumption between every pair of concept names. Retrieval (or query answering) is the collecting of all individuals in the knowledge base that are instance of a given concept in every model of the knowledge base. It is formally defined by the set $\{a \in \mathcal{O} \mid \Sigma \models a{:}C\}$, where C is the querying concept.

2.2.1 Terminological reasoning

Terminological reasoning involves only the terminology (i.e. without considering Abox assertions). In spite of the fact that all the reasoning services can be reduced to knowledge base satisfiability, terminological reasoning is important because of the fact that for most of the concept languages the terminology does not depend on the assertional part. This independence relies on the fact that some services which require dealing with concept expressions only (e.g. subsumption) can be performed ignoring the assertions. This assumption is extremely useful; for example, it enables the verification that an `Elephant` is a `Mammal` without considering the complete knowledge base about wild life.

Unfortunately, not all concept languages enjoy this property; among these are the concept languages in which individuals can be used in concept expressions (e.g. the ones with `one-of`). In fact, while in general the expression $\forall R.\{\ a, b\ \}$ is not subsumed by $\forall R.A$, with respect to a knowledge base containing the assertions $\{a{:}A, b{:}A\}$ the subsumption actually holds:

$$\langle \emptyset, \{a{:}A, b{:}A\}\rangle \models \forall R.\{\ a, b\ \} \sqsubseteq \forall R.A$$

Since most of the languages enjoy the property of terminology independence, and terminological reasoning services are fundamental for knowledge representation systems (classification, for example), a big effort has been spent on the development of efficient concept satisfiability algorithms (see Horrocks and Patel-Schneider [1998a]). The concept satisfiability problem is general enough, because concept subsumption can be reduced to the unsatisfiability of a concept expressions: $C \sqsubseteq D$ if and only if $C \sqcap \neg D$ is unsatisfiable.[4]

The main idea of the concept satisfiability algorithm, presented in Baader and Hollunder [1991a], is based on a notational variant of the first–order tableaux calculus. It is based on the general idea that, since presenting a model is a way of showing that the

[4]If $C \sqcap \neg D$ is unsatisfiable C intersected with $\neg D$ it is empty in each model, so C has to be included in D. The other way around is analogous.

concept is satisfiable, a tableau can be used to build a model of the given concept.

Example 2.2
Looking for a model of expression $A \sqcap \exists R.B$ means building an interpretation $\mathcal{I}$ in which the expression's extension is not empty. The process starts "creating" an object x in the domain $\Delta^{\mathcal{I}}$, which belong to the extension of the concept $A \sqcap \exists R.B$:

$$x \in (A \sqcap \exists R.B)^{\mathcal{I}},$$

x should be in both $A^{\mathcal{I}}$ and $(\exists R.B)^{\mathcal{I}}$ because of the semantics of $\sqcap$, so two more constraints are induced:

$$x \in A^{\mathcal{I}}, x \in (\exists R.B)^{\mathcal{I}}$$

The semantics of the existential constructor guides the generation of a new individual $y \in \Delta^{\mathcal{I}}$ (potentially different from x) such that:

$$(x, y) \in R^{\mathcal{I}}, y \in B^{\mathcal{I}}$$

From the set of constraints above an interpretation can be derived:

$$\begin{array}{rcl} \Delta^{\mathcal{I}} & = & \{x, y\} \\ A^{\mathcal{I}} & = & \{x\} \\ B^{\mathcal{I}} & = & \{y\} \\ R^{\mathcal{I}} & = & \{(x, y)\} \end{array}$$

which can be easily verified as being a model for the initial concept expression $A \sqcap \exists R.B$.

Example 2.2 can be generalised by the notion of *constraint system* as defined in Schmidt-Schauss and Smolka [1991]. A constraint system is a set of constraints, which are syntactic elements of the forms:

$$\begin{array}{rl} x{:}C & \text{concept constraint} \\ \langle x, y\rangle{:}R & \text{role constraint} \\ x \neq y & \text{inequality constraint} \end{array} \tag{2.5}$$

Although the syntax is similar to the one for the Abox assertions, there is a difference in the meaning of the x and y items. These are not individuals, but variables, so the unique name assumption does not apply to them unless stated by an inequality constraint. The similarity with Abox assertions is that both kind of expressions restrict their arguments to be in a concept or role expression.

When in a constraint system between two variables the inequality constraint holds, these two variables are said to be separated. A constraint system can be seen as a graph where the edges are labelled with the roles, and the nodes are the variable names; when a variable x is connected by a role R to a variable y, y is said to be an R-successor of x. Given a constraint system S the notation $S[y/z]$ represents the constraint system obtained by substituting each occurrence of variable y with variable z in the constraint system S.

The process of looking for a model proceeds by extending (or completing) a constraint system using a set of completion rules. These rules modify a constraint system, adding or rewriting the constraints contained on it. For example the rule for the existential quantification (see Example 2.2) is defined as:

$$
\begin{array}{lll}
S & \rightarrow_{\exists} & \{\langle x,y\rangle{:}R,\ y{:}C\} \cup S \\
 & & \text{if } x{:}\exists R.C \text{ in } S,\ y \text{ is a new variable} \\
 & & \text{and there is no } z \text{ s.t. both } \langle x,z\rangle{:}R \text{ and } z{:}C \text{ in } S.
\end{array}
$$

The meaning of the rule is that the constraint system S should be expanded by adding the constraints $\langle x,y\rangle{:}R$ and $y{:}C$, if the conditions in the following lines are verified. The set of completion rules varies according to the different concept languages, for example the rules for $\mathcal{ALCN}$ are shown in Figure 2.1.

The concept satisfiability algorithm starts with the constraint system $\{x{:}C\}$, where C is the concept to check, then it applies the completion rules as long as the preconditions are satisfied. When no rule is applicable the constraint system is said to be completed, and if there are not contradictory constraints, a model for the concept C can be derived (see Donini et al. [1996b]). Contradictions in a constraint system are detected by the so–called clashes, which are constraint systems containing constraints of one of these forms:

1. $\{x{:}\bot\}$

2. $\{x{:}A, x{:}\neg A\}$

3. $\{x{:}\leq n\, R\} \cup \{\langle x,y_i\rangle{:}R \mid i \in 1 \ldots n+1\}$
   $\cup \{y_i \neq y_j \mid i,j \in 1 \ldots n+1, i \neq j\}$

$S \rightarrow_{\sqcap} \{x{:}C_1,\ x{:}C_2\} \cup S$
if $x{:}C_1 \sqcap C_2$ in S,
and both $x{:}C_1$ and $x{:}C_2$ are not in S.

$S \rightarrow_{\sqcup} \{x{:}D\} \cup S$
if $x{:}C_1 \sqcup C_2$ in S,
neither $x{:}C_1$ or $x{:}C_2$ is in S
and $D = C_1$ or $D = C_2$.

$S \rightarrow_{\forall} \{y{:}C\} \cup S$
if $x{:}\forall R.C$ in S, and $\langle x, y\rangle{:}R$ is in S,
and $y{:}C$ not in S.

$S \rightarrow_{\exists} \{\langle x, y\rangle{:}R,\ y{:}C\} \cup S$
if $x{:}\exists R.C$ in S, y is a new variable
and there is no z s.t. both $\langle x, z\rangle{:}R$ and $z{:}C$ in S.

$S \rightarrow_{\geq} \{\langle x, y_i\rangle{:}R \mid i \in 1 \ldots n\} \cup \{y_i \neq y_j \mid i, j \in 1 \ldots n, i \neq j\} \cup S$
if $x{:}\geq n\, R$ in S, $y_1 \ldots y_n$ are new variables
and x has not n pairwise disjoint R-successors in S.

$S \rightarrow_{\leq} S[y/z]$
if $x{:}\leq n\, R$ in S,
x has no more than n R-successors in S,
$\langle x, y\rangle{:}R$, $\langle x, z\rangle{:}R$ are in S,
and $y \neq z$ is not in S.

Figure 2.1: Completion rules for $\mathcal{ALCN}$

Not all the completion rules are deterministically applicable (i.e. only in one way). For example in the disjunction rule $\rightarrow_{\sqcup}$ one of the two concepts can be nondeterministically chosen to be inserted in the augmented constraint system; the choice can generate results in two different constraint systems. One is chosen and if it leads to a clash, the algorithm backtracks and tries the second one. In the case of $\mathcal{ALCN}$ the nondeterministic rules are those associated with disjunction, and with the upper number restriction (the variables to be merged are nondeterministically chosen).

The algorithm based on the completion rules has been proven to solve the concept consistency checking problem; i.e. it produce a clash free completed constraint system if and only if the concept is satisfiable (see Donini et al. [1996b]). Presenting the formal proof of that result is not in the scope of this work, but an overview of how it is structured can be useful because of it can be generalised to other concept languages with different set of rules. The first step consists in showing that the deterministic rules application preserves the satisfiability of the constraint system, and that the nondeterministic rules can be applied in such a way that the satisfiability is preserved. The second step consists in showing that the rule application always terminates; i.e. all the possible completed constraint systems which can be generated are finite. Finally, the last step shows that from a completed clash-free constraint system a model can be constructed.

2.2.2 Hybrid reasoning

Hybrid reasoning takes account of both the parts of a knowledge base. We consider algorithms for solving the problem of knowledge base satisfiability. In principle, this approach is general enough because all reasoning services can be reduced to knowledge base satisfiability (see Section 2.2). Indeed, most of the DL systems available provide different reasoning services by reducing them to KB satisfiability, and although alternative approaches have been suggested (see for example Rousset [1999]), no other satisfactory alternative has yet been found.

For a large class of concept languages, including $\mathcal{ALCN}$, the algorithm can be a generalisation of that used for terminological reasoning. The notion of constraint system is extended, allowing the presence of constraints for individuals as well as variables. In addition, due to the unique name assumption, each pair of different individuals implicitly introduces an inequality constraint which forces them to be separated.

The initial constraint system is not composed by a single constraint as the terminological case, but the full Abox is included in the constraint system. Each role or concept assertion is added to the constraint system as a corresponding role or concept constraint. Then the algorithm proceeds, completing the constraint system as in the terminological case. In Buchheit et al. [1993] the proof of correctness and completeness of the technique has been provided. Example 2.3 below, shows an knowledge base completion for an instance checking problem.

Example 2.3
Consider the following knowledge base with an empty terminology:

$$\Sigma = \langle \emptyset, \left\{ \begin{array}{l} \texttt{susan}:\texttt{Female}, \texttt{bill}:\neg\texttt{Female}, \\ \langle \texttt{sarah}, \texttt{susan}\rangle:\texttt{FRIEND}, \langle \texttt{sarah}, \texttt{andrea}\rangle:\texttt{FRIEND}, \\ \langle \texttt{susan}, \texttt{andrea}\rangle:\texttt{LOVES}, \langle \texttt{andrea}, \texttt{bill}\rangle:\texttt{LOVES} \end{array} \right\} \rangle$$

This is one of the classical examples for showing the necessity of reasoning by case in DL. The example pivots on the fact that the sex of the individual $\mathtt{andrea}$ is not specified, so the KB contains incomplete knowledge.[5]

We want to check whether $\mathtt{sarah}$ has a female friend loving a male, i.e. verifying the instance checking problem

$$\Sigma \models \mathtt{sarah}{:}(\exists\mathtt{FRIEND}.(\mathtt{Female} \sqcap (\exists\mathtt{LOVES}.\neg\mathtt{Female}))).$$

It is easy to verify the instance checking problem using reasoning by case: we can assume that $\mathtt{andrea}$ is either in the concept $\mathtt{Female}$ or $\neg\mathtt{Female}$. In the first case, $\mathtt{andrea}$ is the female friend of $\mathtt{sarah}$ loving a male ($\mathtt{bill}$). In the second case, $\mathtt{susan}$ is the female friend of $\mathtt{sarah}$ loving a male ($\mathtt{andrea}$).

This problem can be reformulated in term of KB satisfiability by verifying that the knowledge base augmented by the assertion

$$\mathtt{sarah}{:}\neg(\exists\mathtt{FRIEND}.(\mathtt{Female} \sqcap (\exists\mathtt{LOVES}.\neg\mathtt{Female})))$$

is unsatisfiable (see Donini et al. [1994]). The corresponding initial constraint system is:[6]

$$\left\{\begin{array}{l} \mathtt{sarah}{:}(\forall\mathtt{FRIEND}.(\neg\mathtt{Female} \sqcup (\forall\mathtt{LOVES}.\mathtt{Female}))) \\ \mathtt{susan}{:}\mathtt{Female}, \mathtt{bill}{:}\neg\mathtt{Female}, \\ \langle\mathtt{sarah}, \mathtt{susan}\rangle{:}\mathtt{FRIEND}, \langle\mathtt{sarah}, \mathtt{andrea}\rangle{:}\mathtt{FRIEND}, \\ \langle\mathtt{susan}, \mathtt{andrea}\rangle{:}\mathtt{LOVES}, \langle\mathtt{andrea}, \mathtt{bill}\rangle{:}\mathtt{LOVES} \end{array}\right\}$$

Using the universal rule ($\rightarrow_\forall$) applied to the first constraint and the role constraints, new constraints for $\mathtt{susan}$ and $\mathtt{andrea}$ are introduced:

$$\left\{\begin{array}{l} \mathtt{susan}{:}(\neg\mathtt{Female} \sqcup (\forall\mathtt{LOVES}.\mathtt{Female})) \\ \mathtt{andrea}{:}(\neg\mathtt{Female} \sqcup (\forall\mathtt{LOVES}.\mathtt{Female})) \end{array}\right\}$$

With the new constraint for $\mathtt{susan}$ the nondeterministic $\rightarrow_\sqcup$ rule can be applied. However choosing the $\mathtt{susan}{:}\neg\mathtt{Female}$ constraint leads to a clash with the initial $\mathtt{susan}{:}\mathtt{Female}$ assertion; therefore the second concept is chosen, and the constraint system is extended with:

$$\left\{\ \mathtt{susan}{:}(\forall\mathtt{LOVES}.\mathtt{Female})\ \right\}$$

[5]The name Andrea is considered a male name in Italy, and female in most of the rest of the world.
[6]All the negations have been pushed in front of concept names.

This constraint, together with the role constraint $\langle \texttt{susan}, \texttt{andrea} \rangle{:}\texttt{LOVES}$ can be used with the universal rule leading to the new constraint:

$$\left\{ \texttt{andrea}{:}\texttt{Female} \right\}$$

Again the $\rightarrow_{\sqcup}$ can be applied to the not yet considered constraint

$$\texttt{andrea}{:}(\neg\texttt{Female} \sqcup (\forall \texttt{LOVES}.\texttt{Female}))$$

containing a disjunction. As in the previous case, choosing $\texttt{andrea}{:}\neg\texttt{Female}$ leads to a clash, so the added constraint is:

$$\left\{ \texttt{andrea}{:}(\forall \texttt{LOVES}.\texttt{Female}) \right\}$$

which can be used with the $\langle \texttt{andrea}, \texttt{bill} \rangle{:}\texttt{LOVES}$ constraint to add the new constraint $\texttt{bill}{:}\texttt{Female}$, which generates a new clash. At this point all the possible nondeterministic choices have been explored, and all the possibilities leaded to a clash. Therefore the constraint system is not satisfiable, and the answer to the instance checking problem is affirmative.

2.3 Role axioms

The DLs we have introduced up to now are powerful languages for describing classes of elements, but the possibility of specifying "global" properties of the roles is limited. For instance, a DL with the transitive closure constructor (see Table 2.2) can "talk" about the transitive closure of a role but it cannot restrict the interpretation of a role to be transitive.

Most of the recently developed DL systems have introduced the possibility of adding axioms about roles into the terminology (see Horrocks [1998], Haarslev and Möller [2000b]). Usually these axioms state the inclusion relationship between role names; but, in addition, transitivity or functional restrictions can be imposed on role names. The choice of role axioms provided by a DL system depends on the balance between the representional usefulness of the construct, and the practical tractability of the KB satisfiability problem.

Although some of the role restrictions can be imposed by appropriate concept axioms, having an algorithm which handles them directly is usually more efficient. A typical case is the functional restriction; i.e. the fact that every element has at most one successor. This restriction can be enforced by a role axiom which states that a role F is functional, or by using the concept axiom $\top \sqsubseteq\ \leq 1\, F.\top$ (in DLs providing the number restriction constructor). In spite of the fact that they are equivalent, the latter method may introduce unnecessary nondeterminism in the tableau construction (see Section 2.2.1).

In our work we concentrate on three kinds of role restrictions: transitivity, functional, and inclusion (or role hierarchy). Indicated in the name of the logic respectively by the letters $_{R^+}$, $\mathcal{H}$, and *f*.

Transitivity This restriction on role names states that the transitive closure of a role must be contained in the role itself. Formally, this can be expressed as a requirement for interpretations, in the same way as in the case of concept axioms (see Section 2.1.2). For example, if the role R is transitive then an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the transitivity of R iff

$$\{(x, y), (y, z)\} \subseteq R^{\mathcal{I}} \text{ implies } (x, z) \in R^{\mathcal{I}}.$$

Transitivity can be extremely useful for modelling part-of relations (see Sattler [1996] for more details). The reader may have noticed the similarity of the transitive restriction with the transitive closure constructor on roles (see Table 2.2). In fact, the latter can be used in place of the transitive restriction (the converse is not true). However, transitive restriction has proved to be practically more tractable than the transitive closure constructor, leading to simpler satisfiability algorithms (see Horrocks et al. [1999b]).

Generally, the transitivity restrictions are not directly represented as axioms in the terminology; instead, we distinguish a subset of the role names as the set of transitive roles (i.e. $\mathcal{TRN} \subseteq \mathcal{RN}$).

Functional If a role name is functional, in every interpretation satisfying the restriction the relation associated to the role name must be a partial function (i.e. it relate any element of the domain to no more than one element). It is easy to see that this is equivalent to a concept axiom of the form $\top \sqsubseteq\ \leq 1\, F.\top$. However, functional restrictions are common in domain modelling, therefore they are usually handled by the satisfiability algorithm directly.

Analogusly to the transitivity restrictions, functional roles are considered a subset of the set of role names (i.e. $\mathcal{FRN} \subseteq \mathcal{RN}$). In addition, we impose the limitation (as in the FaCT system) that the sets of functional roles and transitive roles are disjoint. The reason for this limitation lies in the fact that is not clear whether transitive functional role are of any usefulness in the modelling process; moreover, the decidability of such a DL is still an open problem (we know that allowing transitive roles in number restriction leads to undecidability, see Horrocks et al. [1999b]).

Inclusion This kind of axiom is the role counterpart of the concept axioms shown in Section 2.1.2, with the difference that they relate sets of pair of elements of the domain instead of sets of elements. Role axioms are expressed by statements of the form $R_1 \sqsubseteq R_2$, where R_1 and R_2 are role expressions. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ satisfies the role axiom iff

$$R_1{}^{\mathcal{I}} \subseteq R_2{}^{\mathcal{I}}.$$

We consider only simple axioms where the role expressions are simply role names, in this way the role axioms express only a taxonomy of role names. It is worth noticing that axioms hide an implicit negation,[7] therefore without restrictions on the role axioms full boolean role expressions can be obtained if the DL has conjunction or disjunction role constructors. The problem is that having a language for role expressions closed w.r.t. negation push the DL close to the decidability border (see Lutz and Sattler [2000]).

Obviously, sub-roles of functional roles must be functional as well; therefore if the DL provides these three restrictions a transitive role cannot be included in a functional role (see the discussion about transitive functional roles above).

Role hierarchy can be “simulated” by using either role conjunction or disjunction. Roughly speaking, the idea is to substitute each node of the role taxonomy by either the conjunction of all the top level role names “above” the node, or the disjunction of the leaves below it. Each role name in then replaced by the corresponding expression; this is more or less the same process as the “unfolding” of the concept definitions. This role unfolding can be performed only if the DL does not provide the qualified number restriction constructor (i.e. $\leq\ n\, R.C$) because for keeping decidability complex role expressions are not allowed as argument of this constructor (see De Giacomo and Lenzerini [1994]).

An example of the expressivity that role restrictions can provide is given by the fact

[7]Just consider the concept case in which the axiom $C \sqsubseteq D$ is equivalent to assert that every element of the domain must belong to the interpretation of the concept $\neg C \sqcup D$

that arbitrary concept axioms can be represented by the combination of transitivity and role inclusion (see Baader [1991]).

2.4 The description logic $\mathcal{SH}f$

In the rest of the thesis we consider the DL $\mathcal{SH}f$; this is the very same logic handled by the DL system FaCT (see Horrocks [1998]) with the addition of the Abox.

$\mathcal{SH}f$ is an extension of the logic $\mathcal{ALC}$ to include transitive roles, role hierarchy and functional restriction. Following the naming convention introduced in Horrocks et al. [1999b], the DL $\mathcal{ALC}$ extended with transitive role (i.e. $\mathcal{ALC}_{\mathcal{R}^+}$) is abbreviated as $\mathcal{S}$. This is due to the relationship with the propositional (multi) modal logic $\mathbf{S4}_{(\mathbf{m})}$ (see Schild [1991]).

Summarising, the DL $\mathcal{SH}f$ is built over a signature of distinct concept ($\mathcal{CN}$), role ($\mathcal{RN}$) and individual ($\mathcal{O}$) set of names; in addition we distinguish two non–overlapping subsets $\mathcal{TRN}$, $\mathcal{FRN}$ of $\mathcal{RN}$ which denote the transitive and the functional roles. Valid concept expressions are defined by the abstract syntax:

$$C ::= \top \mid \bot \mid A \mid \neg A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C$$

where A is a concept name chosen from the set $\mathcal{CN}$ and R a role name from $\mathcal{RN}$.

A $\mathcal{SH}f$ knowledge base is composed by a Tbox containing axioms of the form $C_1 \sqsubseteq C_2$ or $R_1 \sqsubseteq R_2$, and an Abox containing individual assertions of the form $a{:}C$ or $\langle a, b\rangle{:}R$. The semantics is the one described in the previous sections.

Since role inclusion axioms contain role names only, from the set of role inclusion assertions a taxonomy of role names can be trivially build. In addition, if there is a cycle in the axioms, then all the names involved in the cycle must correspond to the same binary relation in every interpretation (satisfying the axioms). For this reason, we assume that the role taxonomy is acyclical, and we represent it by a partial order $\preceq$ defined over the set of role names.

In the next chapter we characterise the structure of the models for a $\mathcal{SH}f$ KB. The idea is to recognise a class of interpretations which completely describes $\mathcal{SH}f$, in the sense that whenever a KB has a model it has a model of this class as well.

This characterisation of the logic will be crucial for the following chapters describing the KB satisfiability and query answering algorithms.

Chapter 3

Model characterisation of $\mathcal{SH}f$

As anticipated in the previous chapters, many of the convenient properties of Description Logics stem from their tree model property; i.e. the fact that every satisfiable formula has a model that is a tree (see Vardi [1997], Grädel [1999]). In the logic we consider the picture is slightly more complicated by the fact that Tbox assertions on roles and the Abox can force non-tree shaped models. In particular, the transitivity forces the presence of "shortcuts" corresponding to the transitive closure of parts of the tree; while the Abox assertions can force any graph shape among the nodes corresponding to individual names. However, we still have a tree–like model property; the models look like a cloud containing the nodes corresponding to individual names with (possibly transitive) trees hanging off the nodes in the cloud (see Figure 3.1).

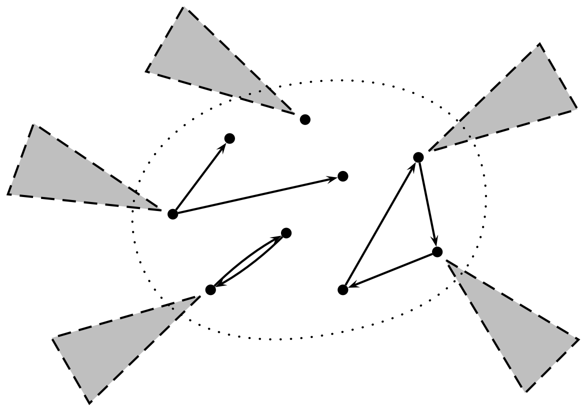


Figure 3.1: Structure of interpretations for $\mathcal{SH}f$

We call this kind of interpretations *shrubs,* because we distinguish a central interconnected part with branches coming out of it.[1] There are a few properties of these interpretations which are fundamental for the proofs we are going to present in the following chapters. The first one is that elements in distinct trees are not connected by any link; i.e. elements of the trees cannot "see" elements of other trees. The second is that each tree is "attached" to a single node corresponding to an individual; if there are links starting from other nodes in the cloud, then these edges must be the result of a transitive closure. This means that all the connections of the tree with the rest of the graph are "mediated" by the individual the tree is attached to.

This chapter will formally present the properties of this class of interpretations which we have called *quasi transitive shrubs*. In addition, we define a transformation for arbitrary interpretations which enables us to present a completeness result for both the problem of satisfiability (see Section 3.3) and logical implication (see Section 7.2) for $\mathcal{SH}f$.

Note that completeness w.r.t. KB satisfiability can be proved by using a technique simpler than the one we are presenting in this chapter. In fact, it is well know that the DL $\mathcal{SH}f$ has the finite model property (see Horrocks [1998]),[2] while our technique always generates infinite interpretations (even uncountable if the starting domain is uncountable). So the natural question is whether we need a different mechanism. Our answer to this doubt is obviously positive, because we need to show the completeness w.r.t. a problem not directly reducible to KB satisfiability (see Section 7.2). We will come back to this latter point in the last section of this chapter.

3.1 Quasi transitive shrub interpretations

As explained above we are interested in interpretations being structured as a collection of trees whose roots are possibly interconnected by links. We start from a set of trees, then we add the required links connecting the roots to the rest of elements.

There are different way of building a tree; the one we have chosen consists of selecting an alphabet (without any restriction of cardinality), and then consider the set of all finite sequences of elements of that alphabet. When it comes to actually build the tree by adding the edges, we are going to connect two sequences only if the second one is equal to the first one with one more element of the alphabet (i.e. a sequence

[1]Strictly speaking a proper shrub does not have the interconnected core, but from an adequate distance they looks like balls of leaves with a few branches coming off them.

[2]Without Abox assertions, however Abox is not affecting the finiteness.

$\alpha_1 \ldots \alpha_n$ can have only sequences $\alpha_1 \ldots \alpha_n \beta$ as successors). This process naturally leads to a forest where each element of the selected alphabet (or more precisely the sequence containing a single element) can be the root of one of the trees. Note that any node of the trees can have as many successors as the cardinality of the alphabet (which is not restricted in any way). Note that this method does not satisfy completely our requirements, because there are still the problems of transitive relations, and Abox role assertions. These two aspects are indeed considered in Definition 3.2.

We chosen this method because what we want to do is to take an arbitrary interpretation and transforming it into a quasi transitive shrub interpretation still having the same properties (in terms of satisfiability of a KB). In addition, we would like to maintain a strict relation between the elements of the original interpretation domain and those of the new interpretation; the idea is taking the original interpretation domain as an alphabet (see Section 3.2.1 for the details).

Let us start with the definition of the set of sequences of elements from a given domain.

Definition 3.1. Let Δ be an arbitrary domain then $\overrightarrow{\Delta}$ is the set of all the finite sequences of elements of Δ. We indicate the empty sequence as ϵ, which by assumption is not in $\overrightarrow{\Delta}$.

To distinguish the elements of the "alphabet" from the sequences we are going to use the letters x, y for elements of the "alphabet" domain, and the letters u, v, w for sequences. We indicate with $\epsilon x_1 \ldots x_n$ the sequence containing the elements $x_1, \ldots, x_n$ of Δ; while ux represents the sequence resulting from the concatenation of the the element x to the sequence u.

We use the set of finite sequences as the domain for the quasi transitive shrub interpretations; in addition, we impose some restriction on the interpretation function. In particular, we require that individual names are mapped to sequences of length 1, and we add a few requirements on the interpretation of roles, which we will detail below.

Definition 3.2 (Quasi transitive shrubs). Let Δ be an arbitrary domain. An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ over a set of individual names $\mathcal{O}$, role names $\mathcal{RN}$, and concept names $\mathcal{CN}$ is a *quasi transitive shrub* interpretation iff it satisfies the following properties. Let u, v be arbitrary elements of $\Delta^{\mathcal{I}}$, x an element of Δ, and R a name in $\mathcal{RN}$:

$$\Delta^{\mathcal{I}} \subseteq \overrightarrow{\Delta} \tag{3.2a}$$

$$\text{if } u \in \mathcal{O}^{\mathcal{I}} \text{ then there is } y \in \Delta \text{ such that } u = \epsilon y \tag{3.2b}$$

$$\text{if } (u, \epsilon x) \in R^{\mathcal{I}} \text{ then } \{u, \epsilon x\} \subseteq \mathcal{O}^{\mathcal{I}} \tag{3.2c}$$

$$\text{if } (u, vx) \in R^{\mathcal{I}} \text{ then either } u = v \text{ or } \{(u, v), (v, vx)\} \subseteq R^{\mathcal{I}} \tag{3.2d}$$

$$\text{if } \{(u, v), (u, vx_1 \ldots x_n)\} \subseteq R^{\mathcal{I}} \text{ then } (v, vx_1 \ldots x_n) \in R^{\mathcal{I}} \tag{3.2e}$$

The first two restrictions of Definition 3.2 correspond to the two above mentioned properties that the domain is composed by finite sequences of given elements, and the individual names are always interpreted as “singleton” sequences. The third property (3.2c) ensures that there is not any link going back from one of the trees to the root of any other tree; i.e. the interaction between different trees is always mediated by the roots, and is restricted to elements mapped from individual names.

The last two properties ((3.2d) and (3.2e)) ensure the tree–like shape, extended to the transitive case. Property (3.2d) extends the tree–shape idea to cover the transitive parts of the interpretations. In fact, if the transitive closure is not considered, the presence of an edge (u, vx) implies that $u = v$ (the successors of a node are sequences containing just one more element). However, we need to consider the case in which a role is transitive or contains a transitive sub-role; therefore we must allow the case in which (u, vx) is a shortcut for the two edges (u, v) and (v, vx). The last property relates the elements contained into a subtree with elements outside the given subtree. In fact, given a node v, all the nodes in the form $vx_1 \ldots x_n$ (with $x_i \in \Delta$) are in the subtree rooted in v. The property describes the case in which there is an edge connecting a node outside this subtree (u) to both the root (v) and an element of the subtree.[3] In this case it is ensured that the edge $(u, vx_1 \ldots x_n)$ is a shortcut for the path (u, v) and $(v, vx_1 \ldots x_n)$.

Clearly, the the structures characterised by Definition 3.2 are neither trees nor acyclic graphs. However, we can distinguished the tree–like structures defined by all the elements beginning with the same sequence. With an abuse of terminology we call these substructures trees. In fact, if we consider their intuitive support tree, the added edges are only the result of transitive closure of the support tree. This restriction does not apply to the roots of these trees, which can be connected in any way (even with cycles). The idea is that the relation among the roots reflects the role assertions in the

[3]The case in which $u = v$ is trivially satisfied.

Abox.

The restriction on the structure of q.t. shrub interpretations are strong enough to characterise two sequences connected by a role. This is shown by the next lemma which is used in Chapter 7 for showing the completeness of the proposed algorithm for query answering (see Proposition 7.16).

Lemma 3.3. *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a quasi transitive shrub interpretation built from the domain Δ (i.e. $\Delta^{\mathcal{I}} \subseteq \overrightarrow{\Delta}$). Let R be an arbitrary role name, u and $v = \epsilon x_1 . . . x_n$ be elements of $\Delta^{\mathcal{I}}$ ($x_i \in \Delta$ for $i = 1, \ldots, n$) for $n > 1$.*[4] *Then the pair (u, v) is in $R^{\mathcal{I}}$ only if:*

$$\text{there is an index } k < n \text{ s.t. } u = \epsilon x_1 . . . x_k \tag{3.3a}$$

$$\text{or } \{u, \epsilon x_1\} \subseteq \mathcal{O}^{\mathcal{I}} \text{ and } \{(u, \epsilon x_1), (\epsilon x_1, v)\} \subseteq R^{\mathcal{I}} \tag{3.3b}$$

Proof. Since the length of the sequence v is finite, then we prove the lemma by induction on the length of v.

Let be $v = \epsilon x_1 x_2$, then by (3.2d) either $u = \epsilon x_1$ or $\{(u, \epsilon x_1), (\epsilon x_1, \epsilon x_1 x_2)\} \subseteq R^{\mathcal{I}}$; indeed, these are exactly the two cases (3.3a) and (3.3b).

Let us assume that the lemma holds for any sequence whose length is less than n, and let us consider $(u, \epsilon x_1 . . . x_n) \in R^{\mathcal{I}}$. By (3.2d) either $u = \epsilon x_1 . . . x_{n-1}$ or $\{(u, \epsilon x_1 . . . x_{n-1}), (\epsilon x_1 . . . x_{n-1}, \epsilon x_1 . . . x_{n-1} x_n)\} \subseteq R^{\mathcal{I}}$. In the first case (3.3a) is satisfied by $k = n - 1$, while for the latter case we can use the inductive hypothesis for the pair $(u, \epsilon x_1 . . . x_{n-1}) \in R^{\mathcal{I}}$. Therefore, either there is an index $k < n - 1$ such that $u = \epsilon x_1 . . . x_k$ or $\{(u, \epsilon x_1), (\epsilon x_1, \epsilon x_1 . . . x_{n-1})\} \subseteq R^{\mathcal{I}}$. In the first case (3.3a) is satisfied, while in the second we have $(u, \epsilon x_1) \in R^{\mathcal{I}}$. We need to show that $(\epsilon x_1, \epsilon x_1 . . . x_{n-1} x_n) \in R^{\mathcal{I}}$. This is verified by (3.2e) because $(u, \epsilon x_1) \in R^{\mathcal{I}}$ and $(u, \epsilon x_1 . . . x_n) \in R^{\mathcal{I}}$ by hypothesis. □

The previous lemma describes the situation in which there is a relation $R^{\mathcal{I}}$ between an arbitrary element u and an element v which is "inside" one of the trees (it cannot be one of the roots because it is a sequence longer than a single element by assumption). When this is the case then either v is contained in the subtree rooted in u (condition (3.3a)), or the relation is "coming" via the root of the tree containing v (i.e. the sequence ϵx_1 (condition (3.3b)). Note that by Definition (3.2c), when the latter condition is verified the element u must correspond to one of the individual names.

[4]We exclude the case $v = \epsilon x$, because this is trivially covered by the definition in (3.2c).

This property is important because it guarantees that inside one of the trees links can only go downward. Therefore there cannot be any link going back to one of the roots, or to elements contained in any other tree.

3.1.1 Canonical interpretations

The definition of a quasi transitive shrub interpretation is independent of any given knowledge base. In particular it does not take into account restrictions on the interpretations imposed by a particular KB, like transitivity restrictions on role names, or inclusion relations between roles. Therefore, by using Definition 3.2 we cannot prevent the transitive closure of relations or multiple relations connecting two elements, even when they are not required (by the kb). The point is that we really need to minimise the non–tree characteristics of the structure of an interpretation (for reasons which will be clear in Chapter 6 and Chapter 7).

Let us consider for example the assertion $S \sqsubseteq R$. In terms of the structure of interpretations, the axiom guarantees that whenever there is an edge labelled with S between two nodes then there must be an edge labelled R as well. This is not a general property of the $\mathcal{SH}f$ logic, but only of interpretations satisfying the knowledge base containing the given assertion.

This problem is specifically related to the possibility of specifying terminological axioms about roles. For example with $\mathcal{ALC}$, or even a DL as expressive as PDL, the tree–model property can be stated independently from any particular kb.

In our case the DL $\mathcal{SH}f$ allows the presence of role axioms; therefore, what we really need is a definition characterising an interpretation w.r.t. a given KB. For this reason we introduce the notion of *canonical interpretations* w.r.t. a given KB (Definition 3.6). Roughly speaking, the definition considers the transitivity and the role ordering induced by the KB in order to restrict the structure of allowed interpretations. It is important to notice that a canonical interpretation does not necessarily satisfy the knowledge base; the two notions are orthogonal one to the other.

Given a knowledge base Σ, the role inclusion axioms in the terminology define a partial order over the set of role names $\mathcal{RN}$, we use the symbol $\preceq_\Sigma$ to denote this reflexive order (by definition $R \sqsubseteq R$). For the sake of simplicity, wherever there is only a single terminology the KB is omitted and the ordering of role names is denoted by the symbol $\preceq$ alone. Moreover, some of the role names have additional restrictions: namely being functional or transitive. Note that the functionality restriction propagates w.r.t. the role ordering, forcing each subrole of a functional role to be functional as well. We distinguish these two non-overlapping subsets of $\mathcal{RN}$ as $\mathcal{FRN}$ (functional) and $\mathcal{TRN}$ (transitive).

Let us consider the requirement that two elements of the domain of a canonical q.t. shrub interpretation are related by two different roles only if this is required in every interpretation. Suppose that we have two elements x, y in the original interpretation $\mathcal{I}$ related by two different roles ($(x, y) \in R^{\mathcal{I}} \cap S^{\mathcal{I}}$); in the q.t. shrub interpretation, all the sequences of the form ux and uxy would be related by the same pair of roles. In order to avoid this, we can duplicate the element y into y_R and y_S, then add the pair (ux, uxy_R) to the interpretation of R and (ux, uxy_S) to the interpretation of S. This technique cannot be applied in every case; for example, if the role S is included in R (i.e. $S \preceq R$), whenever a pair (x, y) is in $S^{\mathcal{I}}$ it must be in $R^{\mathcal{I}}$ as well.

Therefore we must consider groups of role names that must "stay together". Fortunately, this can be discovered by looking at the structure of the relation $\preceq$ induced by a knowledge base, together with the functional restrictions. We identify a subset of all the subsets of $\mathcal{RN}$, calling the elements of it *labels*. Intuitively, we are going to make as many copies of elements of the original domain as the number of possible labels.

The following definition provides the formal description of the set of labels w.r.t. a given knowledge base. It is important to stress the fact that this set is induced by the assertions about the roles in a knowledge base, therefore they are not intrinsic to the logic $\mathcal{SH}f$.

The first thing we need to do is to formalise the notion of the fact that two roles can "stay together"; i.e. the fact that when we have different roles connecting two elements they cannot be split. The simplest case is when a role name is included in an other (i.e. $R \preceq S$), but the interaction between role hierarchy and functional restriction makes the story slightly more complicated. For example, let us consider two role names R_1 and R_2 both included in a functional role F. In an arbitrary interpretation $\mathcal{I}$, whenever $(x, y) \in {R_1}^{\mathcal{I}}$ and $(x, z) \in {R_2}^{\mathcal{I}}$ we know that both (x, y) and (x, z) must be in $F^{\mathcal{I}}$ because of the inclusion; in addition, $F^{\mathcal{I}}$ is functional therefore $y = z$. This scenario may be even more elaborated by considering three roles R_1, R_2, R_3 and two unrelated functional roles F_1, F_2 having the inclusion relation like $R_1 \preceq F_1$, $R_2 \preceq F_1$, $R_2 \preceq F_2$, and $R_3 \preceq F_2$. With arguments which are similar to the simpler case, we can conclude that if in all three relations an element has a successor (i.e. $(x, y_1) \in {R_1}^{\mathcal{I}}$, $(x, y_2) \in {R_2}^{\mathcal{I}}$, and $(x, y_3) \in {R_3}^{\mathcal{I}}$) then this successor must be the same (i.e. $y_1 = y_2 = y_3$).

Note that we are not saying that $(x, y) \in {R_1}^{\mathcal{I}}$ implies the presence of $(x, y) \in {R_2}^{\mathcal{I}}$

as well (like the case in which $R_1 \preceq R_2$), but only that if they are in the interpretation then they must coincide. This property is used in the proofs for the query answering algorithm presented in Chapter 7 (see Proposition 7.15).

The broad idea in the definition is the fact that for a non–functional role R every "preceding" role in the hierarchy must stay with it; while when R is functional we must take the "successors" as well. This should be propagated for covering the cases shown above. So we define a relation $\lesssim$ describing the "staying together" of the roles. Note that this relation is not always symmetric because if it is the case that $S \preceq R$ and $R \not\preceq S$, then $S \lesssim R$ but not the converse, while when there are functional roles involved it becomes symmetric (e.g. in the first of the example above, we have that $R_1 \lesssim R_2$ and the converse as well).

Definition 3.4. The relation $\lesssim$ is recursively defined by

$$R_1 \lesssim R_2 \text{ iff } \begin{cases} R_1 \preceq R_2, & \text{or} \\ R_2 \preceq R_1 & \text{and } R_1 \text{ is functional, or} \\ R_1 \lesssim R \text{ and } R \lesssim R_2 & \text{for some role } R \text{ if neither} \\ & \quad R_1 \preceq R_2 \text{ nor } R_2 \preceq R_1 \end{cases} \tag{3.4a}$$

The relation is well defined because it can be build by using the structure of the role hierarchy. We use the newly defined relation for introducing the *set of labels* of a KB.

A label L is a subset of $\mathcal{RN}$ ($L \in 2^{\mathcal{RN}}$). For a fixed ordering $\preceq$ and set of functional roles $\mathcal{FRN}$ we define the set of labels of $\mathcal{RN}$ as a maximal[5] subset $\mathcal{L}$ of $2^{\mathcal{RN}}$ satisfying the properties that for any label $L \in \mathcal{L}$

$$\text{if } \{R, S\} \subseteq L \text{ then } R \lesssim S \text{ or } S \lesssim R; \tag{3.4b}$$

$$\text{if } R \in L \text{ then } \{S \mid R \lesssim S\} \subseteq L. \tag{3.4c}$$

First we show a few properties of the sets of labels of a given KB.

[5]Maximal w.r.t. set inclusion.

Proposition 3.5. *Let us consider a set of labels $\mathcal{L}$ of a given KB:*

$$\emptyset \in \mathcal{L}; \tag{3.5a}$$

$$\textit{for all } R \in \mathcal{RN} \textit{ there is } L \in \mathcal{L} \textit{ s.t. } R \in L; \tag{3.5b}$$

$$\textit{the set of labels is unique;} \tag{3.5c}$$

$$\textit{if two labels share a functional role, then they are equal.} \tag{3.5d}$$

Proof. **3.5a** Let assume that $\emptyset \notin \mathcal{L}$, clearly $\emptyset$ satisfies the properties in Definition 3.4. Therefore $\{\emptyset\} \cup \mathcal{L}$ is a set of labels which strictly includes $\mathcal{L}$, contradicting the hypothesis that $\mathcal{L}$ is maximal.

3.5b Let us assume that there is a role R s.t. $R \notin L$ for any label L in $\mathcal{L}$. Let us build the new label $L' = \{S \mid R \lesssim S\}$, note that L' satisfies both the properties in Definition 3.4; therefore $\{L'\} \cup \mathcal{L}$ is a set of labels which strictly includes $\mathcal{L}$, contradicting the hypothesis that $\mathcal{L}$ is maximal.

3.5c Let $\mathcal{L}_1$, $\mathcal{L}_2$ be two sets of labels satisfying the Definition 3.4. We are going to show that $\mathcal{L}_1 \subseteq \mathcal{L}_2$, by choosing a label L in $\mathcal{L}_1$ and proving that it must be in $\mathcal{L}_2$ as well.

If L is the empty set, then it is contained in $\mathcal{L}_2$ by (3.5a), so we consider the case in which $L \neq \emptyset$. Let us assume that $L \notin \mathcal{L}_2$; the label L satisfies the properties in Definition 3.4 because $\mathcal{L}_1$ is a set of labels. Therefore $\{L\} \cup \mathcal{L}_2$ is a set of labels which strictly includes $\mathcal{L}$, contradicting the hypothesis that $\mathcal{L}$ is maximal.

Using the very same arguments we can show that $\mathcal{L}_2 \subseteq \mathcal{L}_1$, therefore they are equal.

3.5a Let L_1,L_2 be two labels in $\mathcal{L}$ sharing a functional role F.

First we show that $L_1 \subseteq L_2$. Let us consider an arbitrary role name R in L_1; if R is equal to F, then it is in L_2 by definition. Let us assume that R is different from F; by Property (3.4b) we have that either $F \lesssim R$ or $R \lesssim F$. If $F \lesssim R$ then $R \in L_2$ by Property (3.4c). Let us suppose that $R \lesssim F$, since $R \preceq R$ we can use Definition (3.4a) for showing that $F \lesssim R$ as well (substitute R_1 with F, and R_2, R' with R. Therefore $R \in L_2$ by Property (3.4c).

Analogously, we can show that $L_2 \subseteq L_1$, therefore $L_1 = L_2$.

□

Definition 3.6 (Canonical Interpretations). A quasi transitive shrub interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is said to be *canonical* w.r.t. a KB Σ if the following two conditions apply. Given arbitrary role names $R, R_1, \ldots, R_n$, and different elements u, v, w of $\Delta^{\mathcal{I}}$ with $w \notin \mathcal{O}^{\mathcal{I}}$:

$$\begin{aligned}&\text{if } \{(u,v),(v,w),(u,w)\} \subseteq R^{\mathcal{I}} \text{ then}\\ &\quad \text{there is a transitive role } S \preceq R \text{ such that } \{(u,w),(v,w)\} \subseteq S^{\mathcal{I}}\end{aligned} \tag{3.6a}$$

$$\text{if } \{(u,w)\} \subseteq {R_1}^{\mathcal{I}} \cap \ldots \cap {R_n}^{\mathcal{I}} \text{ then there is a label } L \text{ s.t. } \{R_1, \ldots, R_n\} \subseteq L \tag{3.6b}$$

The definition of a canonical interpretations is given in such a way that it tries to minimize the non–tree characteristics of the structure of an interpretation. In fact, given the $\mathcal{SH}f$ DL we cannot restrict ourselves to tree structures; for example the role hierarchy can force two different edges connecting the very same pair of elements, or the transitivity can force “shortcuts” of role paths.

The first restriction (3.6a) ensures that a canonical interpretation contains only the necessary shortcuts; i.e. those required for satisfying the transitivity restrictions. The property states that if the interpretation contains a shortcut (i.e. $\{(u,v),(v,w),(u,w)\} \subseteq R^{\mathcal{I}}$), this must be caused by a transitive role. Because of the interaction between transitivity and role inclusion, it is not necessary that the role R is transitive: a transitive role included in R can force the shortcut as well.

This is not enough, in fact the restriction (3.6a) does not require the expected $(u,v) \in S^{\mathcal{I}}$. The reason comes from the interaction of assertions about the individuals with the role structure; in fact, we know that Abox assertions can impose any shape in the part of interpretations concerning the individuals.

Example 3.1
Let us consider the Abox assertions

$$\begin{gathered}\langle a,b\rangle{:}R, \langle a,c\rangle{:}S, \langle b,c\rangle{:}S',\\ a{:}A, b{:}A, c{:}(A \sqcap \exists S'.\neg A)\end{gathered}$$

together with the inclusion assertions $S' \sqsubseteq S$, $S \sqsubseteq R$, and the role names S,S' being transitive. One interpretation ($\mathcal{I}$) satisfying the assertions is depicted as the graph below

$$\begin{tikzcd}
\{A\}\ u \arrow[dd, "\{R\}"'] \arrow[dr, "\{S,R\}"] \arrow[drr, bend left, "\{S,R\}"] & & \\
& \overset{\{A\}}{w'} \arrow[r, "\{S',S,R\}"'] & w \\
\{A\}\ v \arrow[ur, "\{S',S,R\}"] \arrow[urr, bend right, "\{S',S,R\}"'] & &
\end{tikzcd}$$

where the element u corresponds to a (i.e. $a^{\mathcal{I}} = u$), v to b, and w' to c; in addition w is an anonymous element. It is not difficult to realise that $\mathcal{I}$ is a sort of "minimal" interpretation satisfying the set of assertions; in addition, it does not force any transitive subrole of R connecting the elements corresponding to a and b.

The restriction (3.6b) concerns the case of multiple edges connecting the very same pair. Again, these cases are restricted to only those which are necessarily imposed by the knowledge base.

3.2 Transforming interpretations into q.t. shrubs

In this section we show how to build a canonical q.t. shrub interpretation from an arbitrary interpretation satisfying a given knowledge base. First we define the transformation, leading to what we call the unravelled interpretation and its transitive closure (Section 3.2.1), then we highlight some of their properties (Section 3.2.2). These results are used to show the completeness of the class of q.t. shrub interpretation w.r.t. the problem of kb satisfiability (Section 3.3), and conjunctive query answering as presented in Chapter 6 and Chapter 7 (see Section 7.2).

The underlying idea of the transformation is simple: we take the domain of the interpretation as the starting alphabet for the sequences, then we define the interpretation function according to the properties, w.r.t. the original interpretation, of the last element of each sequence. Roughly speaking, a sequence is in the extension of a concept name if the last element is in the extension of the same concept w.r.t. the original interpretation. Analogously, two sequences are related if the second one is equal to the first one with the addition of an element (see Definition 3.2), and their last elements are related in the original interpretation (see Definition 3.7).

Example 3.2

Let us consider the following simple interpretation without individual names:

$$\begin{tikzcd}
u \arrow[r, "{\{S,R\}}"'] \arrow[rr, bend left, "{\{S\}}"] & v \arrow[r, "{\{S\}}"'] & z \\
\{A\} & & \{B\}
\end{tikzcd}$$

The idea is transforming it into into an interpretation containing an infinite number of trees like:

$$\begin{tikzcd}
 & \{A\} & & & \{A\} \\
 & \omega u \arrow[dl, "{\{S,R\}}"'] \arrow[dr, "{\{S\}}"] & & \omega v \arrow[d, "{\{S\}}"] & \omega z \\
\omega uv \arrow[d, "{\{S\}}"'] & & \omega uz & \omega vz & \\
 & & \{A\} & \{A\} & \\
\omega uvz \; \{A\} & & & &
\end{tikzcd}$$

where ω is an arbitrary sequence of elements from the set $\{u, v, z\}$ or the empty sequence (ϵ).[6]

As shown in the example above, even starting from a finite interpretation we always build a new infinite interpretation (because of the arbitrary sequence ω in the example). It is conceivable to impose more restrictions in order to obtain smaller interpretations (e.g. in the example we may restrict the prefix ω to the empty sequence ϵ); however, more restrictions means more complications and the construction we use is sufficient for our purpose.

The construction we sketched makes the new interpretations violating any transitive restriction. For instance, the original interpretation of the example above satisfies a transitive restriction on the role S, but the newly build interpretation do not satisfies it. Note that this is something intrinsic in the construction; therefore in a subsequent step the appropriate relations are transitively closed (see Definition 3.8). In addition, this simple initial idea is complicated by the fact that the interpretation must be canonical w.r.t. a given knowledge base, together with the pernicious interaction of transitive and functional restrictions with the role hierarchy.

[6]Actually, the transformation is more involved as we are going to show in the following sections. This example is giving the intuition behind the transformation.

3.2.1 Unravelling the interpretation

We assume that the interpretation we are transforming is defined over a set of individual names $\mathcal{O}$, of concept names $\mathcal{CN}$, and role names $\mathcal{RN}$. The set of role names includes two non-overlapping sets: the functional ($\mathcal{FRN}$) and transitive ($\mathcal{TRN}$) names. In addition, a partial ordering $\preceq$ is defined on $\mathcal{RN}$; this ordering must satisfy the conditions that names included in functional names are functional as well, and transitive roles cannot be included into functional roles. Formally, given two names S, R in $\mathcal{RN}$ if $S \preceq R$ then $R \in \mathcal{FRN}$ implies $S \in \mathcal{FRN}$, and $S \in \mathcal{TRN}$ implies $R \notin \mathcal{FRN}$.

Now we define the *unravelled interpretation* of $\mathcal{I}$. We start by building as many copies of Δ as the number of labels in $\mathcal{L}$ (Δ_L for any $L \in \mathcal{L}$). For each label L there is a bijection δ_L mapping each copy in Δ_L to the original element in Δ. We define a new domain Δ' by taking the union of all the copies of Δ (i.e. $\Delta' = \bigcup_{L\in\mathcal{L}} \Delta_L$). For any element of this new set we define two mappings: the first (δ) maps any element to the original element of Δ it was copy of (i.e. $\delta = \bigcup_{L\in\mathcal{L}} \delta_L$); while the second ($\lambda$) maps any element to the label corresponding to the given copy of Δ (i.e. $\lambda : \Delta' \to \mathcal{L}$, and $\lambda(x) = L$ iff $x \in \Delta_L$). Using the domain Δ' we define the set $\overrightarrow{\Delta'}$ of all the finite nonempty sequences of elements of Δ' (see Definition 3.1). The mappings δ and λ can easily be extended to $\overrightarrow{\Delta'}$ by using the value of the last element of the given sequence (i.e. $\delta(ux) = \delta(x)$ and $\lambda(ux) = \lambda(x)$ for any $u \in \overrightarrow{\Delta} \cup \{\epsilon\}$).

Elements of $\overrightarrow{\Delta'}$ can be naturally associated to nodes in a forest, where ϵx are the roots and ux is a successor of u. We then use the relation structure in the original interpretation $\mathcal{I}$ for adding the necessary edges between nodes. Since we did not make any assumption on the original domain Δ, we cannot fix the width of the resulting tree. In fact it can be even uncountable. The following definition formally describes the unravelled interpretation of $\mathcal{I}$.

To simplify the notation we indicate with $\mathcal{O}^{\mathcal{I}}$ the set of elements of Δ to which the individual names in $\mathcal{O}$ are mapped by the interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ (i.e. $\mathcal{O}^{\mathcal{I}} = \big\{x \in \Delta \mid \exists o \in \mathcal{O}. o^{\mathcal{I}} = x\big\}$).

Definition 3.7 (Unravelled interpretation). The unravelled interpretation $\widehat{\mathcal{I}} = (\Delta^{\widehat{\mathcal{I}}}, \cdot^{\widehat{\mathcal{I}}})$ of $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ is defined as:

- the domain $\Delta^{\widehat{\mathcal{I}}}$ is the set of all the finite nonempty sequences of elements of Δ'

$$\Delta^{\widehat{\mathcal{I}}} = \overrightarrow{\Delta'};$$

- given an individual name $o \in \mathcal{O}$

$$o^{\widehat{\mathcal{I}}} = \epsilon x \text{ for some } x \in \Delta' \text{ s.t. } \delta(x) = o^{\mathcal{I}} \text{ and } \lambda(x) = \emptyset;$$

- given a role name $R \in \mathcal{RN}$,

$$R^{\widehat{\mathcal{I}}} = \left\{(u,v) \in \mathcal{O}^{\widehat{\mathcal{I}}} \times \mathcal{O}^{\widehat{\mathcal{I}}} \mid (\delta(u), \delta(v)) \in R^{\mathcal{I}}\right\} \tag{3.7a}$$

$$\cup \{(u, ux) \in \Delta^{\widehat{\mathcal{I}}} \times \Delta^{\widehat{\mathcal{I}}} \mid (\delta(u), \delta(x)) \in R^{\mathcal{I}}, R \in \lambda(x), \text{ and } u \notin \mathcal{O}^{\widehat{\mathcal{I}}} \text{ or } \delta(x) \notin \mathcal{O}^{\mathcal{I}}\}. \tag{3.7b}$$

Note that the two sets (3.7a) and (3.7b) are distinct because in the first one all the pairs are of the form $(\epsilon x, \epsilon y)$, while in the second (u, ux). The condition "$u \notin \mathcal{O}^{\widehat{\mathcal{I}}}$ or $\delta(x) \notin \mathcal{O}^{\mathcal{I}}$" in the (3.7b) part prevents pairs of the form $(\epsilon x, \epsilon xy)$ being added to the interpretation when an analogous pair can be added by the (3.7a) part. Without the condition, an assertion like $\langle a, b\rangle{:}F$ in the Abox with F functional would cause the functional restriction on F to be violated in the unravelled interpretation.

- given an atomic concept $A \in \mathcal{CN}$

$$A^{\widehat{\mathcal{I}}} = \left\{u \in \Delta^{\widehat{\mathcal{I}}} \mid \delta(u) \in A^{\mathcal{I}}\right\}.$$

The mapping $\cdot^{\widehat{\mathcal{I}}}$ is well defined; in particular the only problem can arise from the non–uniqueness of the element x in the interpretation of individual names. However, it is easy to realise that we map an individual name o to a single copy of the original element $o^{\mathcal{I}}$ (the one in the domain $\Delta_{\emptyset}$).

The unravelling is a well known technique for showing that a modal logic has the tree model property (see Vardi [1997], Streett [1982], Grädel [1999]). Unfortunately, the standard technique is not enough in our case, in fact the unravelling of an interpretation would not satisfies the transitive restriction on roles. It is easy to see the problem by considering the fact that only pairs like (u, ux) are added to the interpretation of a role by the (3.7b) set; therefore if both (u, ux) and (ux, uxy) are in the interpretation of a transitive role, the required (u, uxy) pair is obviously missing. We overcome this shortcoming by simply adding the transitive closure of relations corresponding to transitive role names. We call this operation the *transitive closure* of the interpretation.

Definition 3.8. Let $\widehat{\mathcal{I}} = (\Delta^{\widehat{\mathcal{I}}}, \cdot^{\widehat{\mathcal{I}}})$ be the unravelled interpretation of $\mathcal{I}$. The transitive closure of $\widehat{\mathcal{I}}$, written as $\widehat{\mathcal{I}}^{+} = (\Delta^{\widehat{\mathcal{I}}+}, \cdot^{\widehat{\mathcal{I}}+})$, is defined as following:

- the domain $\Delta^{\widehat{\mathcal{I}}+}$ is equal to the unravelled domain $\Delta^{\widehat{\mathcal{I}}}$;
- individuals are mapped as in $\widehat{\mathcal{I}}$

$$o^{\widehat{\mathcal{I}}+} = o^{\widehat{\mathcal{I}}};$$

- given an atomic concept $A \in \mathcal{CN}$

$$A^{\widehat{\mathcal{I}}+} = A^{\widehat{\mathcal{I}}};$$

- given a non–transitive role name $R \in \mathcal{RN} \setminus \mathcal{TRN}$

$$R^{\widehat{\mathcal{I}}+} = R^{\widehat{\mathcal{I}}} \cup \bigcup_{S \in \mathcal{TRN}, S \preceq R, R \not\preceq S} S^{\widehat{\mathcal{I}}+};$$

- given a transitive role name $R \in \mathcal{TRN}$

$$\begin{aligned} R^{\widehat{\mathcal{I}}+} = \{(u,v) \mid \exists \{z_1,\ldots,z_n\} \subseteq \Delta^{\widehat{\mathcal{I}}} \text{ s.t. } \{z_1,\ldots,z_n\} \text{ is finite and} \\ z_1 = u, z_n = v, \{(z_i, z_{i+1}) \mid i = 1,\ldots,n-1\} \subseteq R^{\widehat{\mathcal{I}}}\} \end{aligned}$$

The definition of transitive closure of $\widehat{\mathcal{I}}$ is recursive and well defined because we assume an acyclic ordering $\preceq$, therefore $\widehat{\mathcal{I}}$ can be build "bottom-up" w.r.t. the ordering over role names. It is easy to realise that the transitive closure satisfies the transitive restrictions on role names, while it does not lose the other properties of the unravelled interpretation. This will enable us to conclude that given an arbitrary interpretation we have an effective way of providing an example of an interpretation satisfying the required properties. This is the fundamental brick which provides the foundations for the completeness proof as we show in the following sections.

3.2.2 Properties of unravelled interpretations

In this section we present two results which constitute the foundation for the completeness proof. The first two propositions highlight some of the properties of the (transitive closure of) unravelled interpretations; while the last one shows that these generated interpretations are indeed canonical q.t. shrubs.

We start by showing that the unravelled interpretation $\widehat{\mathcal{I}}$ satisfies some fundamental properties; then in Proposition 3.10 we show that the same properties are maintained even after the transformation in transitive closure.

Proposition 3.9. *Let $\widehat{\mathcal{I}} = (\Delta^{\widehat{\mathcal{I}}}, \cdot^{\widehat{\mathcal{I}}})$ be the unravelled interpretation of $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$. Then, $\widehat{\mathcal{I}}$ satisfies the following properties:*

For any $o \in \mathcal{O}$, $\delta(o^{\widehat{\mathcal{I}}}) = o^{\mathcal{I}}$ (3.9a)

For any $\{u, v\} \subseteq \Delta^{\widehat{\mathcal{I}}}$ and role R, if $(u, v) \in R^{\widehat{\mathcal{I}}}$, then $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$ (3.9b)

For any $u \in \Delta^{\widehat{\mathcal{I}}}$, $x \in \Delta$, and role R, if $(\delta(u), x) \in R^{\mathcal{I}}$, then there is $v \in \Delta^{\widehat{\mathcal{I}}}$ s.t. $(u, v) \in R^{\widehat{\mathcal{I}}}$ and $\delta(v) = x$ (3.9c)

For any $u \in \Delta^{\widehat{\mathcal{I}}}$ and concept C, $u \in C^{\widehat{\mathcal{I}}}$ iff $\delta(u) \in C^{\mathcal{I}}$ (3.9d)

For any $u \in \Delta^{\widehat{\mathcal{I}}}$ and functional role $R \in \mathcal{FRN}$,
$\sharp\Big\{v \mid (u, v) \in R^{\widehat{\mathcal{I}}}\Big\} = \sharp\{x \mid (\delta(u), x) \in R^{\mathcal{I}}\}$ (3.9e)

For any u, v in $\Delta^{\widehat{\mathcal{I}}}$, and role names R, S s.t. $S \preceq R$, if $(u, v) \in S^{\widehat{\mathcal{I}}}$ and $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$, then $(u, v) \in R^{\widehat{\mathcal{I}}}$ (3.9f)

Proof.

(3.9a) This is true by definition, because $\delta(o^{\widehat{\mathcal{I}}}) = \delta(x) = o^{\mathcal{I}}$.

(3.9b) If $(u, v) \in R^{\widehat{\mathcal{I}}}$ then by Definition 3.7 in both the two sets (3.7a) and (3.7b), the condition $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$ must be satisfied.

(3.9c) Let us assume that $(\delta(u), x) \in R^{\mathcal{I}}$ with $x \in \Delta$. We distinguish the two cases in which u is or is not mapped from an individual name (i.e. $u \in \mathcal{O}^{\widehat{\mathcal{I}}}$).

- Let assume that $u \in \mathcal{O}^{\widehat{\mathcal{I}}}$. If $x \in \mathcal{O}^{\mathcal{I}}$, then there is an element $x' \in \Delta'$ such that $\lambda(x') = \emptyset$, and $\delta(x') = x$ ($\emptyset \in \mathcal{L}$ by Proposition (3.5a)). In addition, $\epsilon x' \in \mathcal{O}^{\widehat{\mathcal{I}}}$ by definition; therefore $(u, \epsilon x') \in R^{\widehat{\mathcal{I}}}$ because it is in the set (3.7a). The element v we are looking for is $\epsilon x'$.

  If $x \notin \mathcal{O}^{\mathcal{I}}$, then there is an element $x' \in \Delta'$ such that $R \in \lambda(x')$ and $\delta(x') = x$ (by Proposition (3.5b) there is at least a label containing R). We are going to show that the element ux' is the v we need. Clearly $ux' \notin \mathcal{O}^{\widehat{\mathcal{I}}}$ because $x \notin \mathcal{O}^{\mathcal{I}}$; in addition, $(\delta(u), \delta(ux')) \in R^{\mathcal{I}}$ because $\delta(ux') =$

$\delta(x') = x$. Therefore the pair (u, ux') is in the set (3.7b), so it is in $R^{\widehat{\mathcal{I}}}$ as well.

– Now we assume that $u \notin \mathcal{O}^{\widehat{\mathcal{I}}}$. By definition there is an element $x' \in \Delta'$ such that $R \in \lambda(x')$ and $\delta(x') = x$. We can proceed as in the previous case by considering ux'. Clearly $\delta(ux') = \delta(x') = x$, so $(\delta(u), \delta(ux')) \in R^{\mathcal{I}}$; therefore all the conditions for the set (3.7b) are satisfied (we assumed that $u \notin \mathcal{O}^{\widehat{\mathcal{I}}}$) and $(u, ux') \in R^{\widehat{\mathcal{I}}}$.

(3.9e) Let us consider the mapping δ restricted to the domain $\left\{v \mid (u,v) \in R^{\widehat{\mathcal{I}}}\right\}$. First we show that the codomain is equal to the set $\left\{x \mid (\delta(u), x) \in R^{\mathcal{I}}\right\}$, and then that the restriction of δ is bijective. This is enough for prooving that the cardinality of the two sets are equal.[7]

If $v \in \left\{v \mid (u,v) \in R^{\widehat{\mathcal{I}}}\right\}$ then $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$ by the already shown Proposition (3.9b), therefore $\delta(v) \in \left\{x \mid (\delta(u), x) \in R^{\mathcal{I}}\right\}$. For the other direction, let x be an element of $\left\{x \mid (\delta(u), x) \in R^{\mathcal{I}}\right\}$. By Proposition (3.9c) there is an element v such that $(u,v) \in R^{\widehat{\mathcal{I}}}$ and $\delta(v) = x$; therefore $v \in \left\{v \mid (u,v) \in R^{\widehat{\mathcal{I}}}\right\}$ (and the restriction of δ we are considering is surjective).

For showing that the restriction of δ is bijective we only need to show that it is injective. This is trivially true in the case that the cardinality of $\left\{v \mid (u,v) \in R^{\widehat{\mathcal{I}}}\right\}$ is less than 2. Therefore we assume that it contains at least two elements. Let v_1 and v_2 be two elements of the first set such that $v_1 \neq v_2$. We reason by contradiction by assuming that $\delta(v_1) = \delta(v_2)$.

The pair (u, v_1) either belongs to the set (3.7a) or (3.7b), and the same applies to (u, v_2). If both of them belong to (3.7a), then $v_1 = \epsilon x_1$ and $v_2 = \epsilon x_2$, with both $\lambda(x_1)$ and $\lambda(x_2)$ equal to $\emptyset$. This means that x_1 and x_2 belong to $\Delta_{\emptyset}$; in addition $\delta_{\emptyset}(x_1) = \delta_{\emptyset}(x_2)$ because $\delta(v_1) = \delta(v_2)$. Therefore $x_1 = x_2$ because $\delta_{\emptyset}$ is a bijection: this is clearly in contradiction with the hypothesis that $v_1 \neq v_2$.

If (u, v_1) belongs to the set in (3.7a), then $u \in \mathcal{O}^{\widehat{\mathcal{I}}}$; therefore (u, v_2) cannot be in the set in (3.7b). The converse applies if we assume (u, v_2) in the set (3.7a). Therefore both (u, v_1) and (u, v_2) must be in the set in (3.7b).

This means that $v_1 = ux_1$ and $v_2 = ux_2$ with $x_1 \neq x_2$ because $v_1 \neq v_2$ by hypothesis. By definition of the domain Δ' there should be two labels L_1 and L_2 such that $x_1 \in \Delta_{L_1}$ and $x_2 \in \Delta_{L_2}$. In addition, $\delta_{L_1}(x_1) = \delta_{L_2}(x_2)$ because

[7]Note that we do not make any assumption on the initial interpretation $\mathcal{I}$.

$\delta(v_1) = \delta(v_2)$ therefore $L_1 \neq L_2$ because both δ_{L_1} and δ_{L_2} are bijective. By definition of the set (3.7b) the role R must be in both the labels L_1 and L_2.

The role R is functional by hypothesis, and we already showed that if two labels have a functional role in common they must coincide. This is in contradiction with the fact that L_1 and L_2 are different. Therefore, the assumption that $\delta(v_1)$ can be equal to $\delta(v_2)$ is false, so δ restricted to the two given sets of (3.9e) is injective as well as surjective.

The existence of a bijection between two sets guarantees that the cardinality of the two sets is the same.

(3.9f) If (u, v) is in $S^{\widehat{\mathcal{I}}}$ then, by Definition 3.7, the pair (u, v) is either in the set (3.7a) or in the set (3.7b). In the first case $(u, v) \in \mathcal{O}^{\widehat{\mathcal{I}}} \times \mathcal{O}^{\widehat{\mathcal{I}}}$ and $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$ by assumption[8] therefore $(u, v) \in R^{\widehat{\mathcal{I}}}$.

In the second case $v = ux$ for some x in Δ', $S \in \lambda(v)$, and it is not the case that $u \in \mathcal{O}^{\widehat{\mathcal{I}}}$ and $\delta(x) \in \mathcal{O}^{\mathcal{I}}$. Note that $S \lesssim R$ (see Definition (3.4a)), so $R \in \lambda(v)$ as well by Definition (3.4c). This enable us to conclude that $(u, v) \in R^{\widehat{\mathcal{I}}}$ by using Definition 3.7.

(3.9d) We prove this point by induction on the structure of the concept expression C. For atomic concept names it is true by definition.

For the inductive step we assume that (3.9d) holds for the concepts C, D, then we show that it holds also for $\neg C$, $C \sqcap D$, and $\exists R.D$.

$\neg C$ For any u, $u \in (\neg C)^{\widehat{\mathcal{I}}}$ iff $u \notin C^{\widehat{\mathcal{I}}}$. For the induction hypothesis this is true iff $\delta(u) \notin C^{\mathcal{I}}$, which is the case iff $\delta(u) \in (\neg C)^{\mathcal{I}}$.

$C \sqcap D$ For any u, $u \in (C \sqcap D)^{\widehat{\mathcal{I}}}$ iff $u \in C^{\widehat{\mathcal{I}}}$ and $u \in D^{\widehat{\mathcal{I}}}$. As in the previous case this is the case iff $\delta(u) \in C^{\mathcal{I}}$ and $\delta(u) \in D^{\mathcal{I}}$, which is true iff $\delta(u) \in (C \sqcap D)^{\mathcal{I}}$.

$\exists R.C$ If $u \in (\exists R.C)^{\widehat{\mathcal{I}}}$, then there is a $v \in \Delta^{\widehat{\mathcal{I}}}$ such that $(u, v) \in R^{\widehat{\mathcal{I}}}$ and $v \in C^{\widehat{\mathcal{I}}}$. By (3.9b) $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$, and $\delta(v) \in C^{\mathcal{I}}$ by the inductive hypothesis. Therefore $\delta(u) \in (\exists R.C)^{\mathcal{I}}$.

For the other direction, let us assume that $\delta(u) \in (\exists R.C)^{\mathcal{I}}$. Then there is $x \in \Delta$ such that $(\delta(u), x) \in R^{\mathcal{I}}$ and $x \in C^{\mathcal{I}}$. By (3.9c) there is $v \in \Delta^{\widehat{\mathcal{I}}}$

[8]Note that since we do not assume that the interpretation $\mathcal{I}$ satisfies the role inclusion, we explicitly add $(\delta(u), \delta(x)) \in R^{\mathcal{I}}$ as a prerequisite.

such that $(u, v) \in R^{\widehat{\mathcal{I}}}$ and $\delta(v) = x$. Since $v \in C^{\widehat{\mathcal{I}}}$ by induction hypothesis $u \in (\exists R.C)^{\widehat{\mathcal{I}}}$.

□

Now we turn our attention to the transitive closure of the unravelled interpretation, showing that this operation would not destroy the properties described in Proposition 3.9, provided that the original interpretation satisfies the transitive and inclusion restrictions for role names. In addition, in the case of the transitive closure we assume that the original interpretation satisfies the restriction on the roles imposed by the knowledge base. This is necessary because Definition 3.8 adds the pairs for ensuring the satisfiability of these restrictions to the interpretation of roles. If these are not satisfied in the original interpretation, then there is a dichotomy between the original and the result interpretations which may invalidate part of the properties we stated in Proposition 3.9.

Proposition 3.10. *Let $\widehat{\mathcal{I}}^+ = (\Delta^{\widehat{\mathcal{I}}^+}, \cdot^{\widehat{\mathcal{I}}^+})$ be the transitive closure of the unravelled interpretation $\widehat{\mathcal{I}} = (\Delta^{\widehat{\mathcal{I}}}, \cdot^{\widehat{\mathcal{I}}})$. If the original interpretation $\mathcal{I}$ satisfies the transitive and inclusion restrictions then $\widehat{\mathcal{I}}^+$ satisfies all the properties in Proposition 3.9.*

Proof.

(3.9a) Since $o^{\widehat{\mathcal{I}}^+} = o^{\widehat{\mathcal{I}}}$ then $\delta(o^{\widehat{\mathcal{I}}^+}) = o^{\mathcal{I}}$.

(3.9b) If $(u, v) \in R^{\widehat{\mathcal{I}}^+}$ then either $(u, v) \in R^{\widehat{\mathcal{I}}}$ or there is a transitive role $S \preceq R$ such that there is a sequence $u = z_1, \ldots, z_n = v$ of elements of $\Delta^{\widehat{\mathcal{I}}}$ such that $\{(z_i, z_{i+1}) \mid i = 1, \ldots, n\} \subseteq S^{\widehat{\mathcal{I}}}$. In the first case $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$ by (3.9b). In the second $\{(\delta(z_i), \delta(z_{i+1})) \mid i = 1, \ldots, n\} \subseteq S^{\mathcal{I}}$; therefore $(\delta(u), \delta(v)) \in S^{\mathcal{I}}$ because $S^{\mathcal{I}}$ is transitive by hypothesis (transitive restrictions are satisfied). In addition, $S^{\mathcal{I}} \subseteq R^{\mathcal{I}}$ because the inclusion restrictions are satisfied; therefore $(\delta(u), \delta(v)) \in R^{\mathcal{I}}$.

(3.9c) This is trivially satisfied because $R^{\widehat{\mathcal{I}}} \subseteq R^{\widehat{\mathcal{I}}^+}$ for any R.

(3.9e) If a role R is functional, then by the restriction on the language it has not any transitive subroles (i.e. $S \not\preceq R$ for any $S \in \mathcal{TRN}$. Therefore $R^{\widehat{\mathcal{I}}^+} = R^{\widehat{\mathcal{I}}}$ by definition, so the property is satisfied because $R^{\widehat{\mathcal{I}}}$ satisfies (3.9e).

(3.9f) Let $(u, v) \in S^{\widehat{\mathcal{I}}+}$, we distinguish the two cases in which S is or is not transitive.

If S is transitive and R is not transitive, then $S^{\widehat{\mathcal{I}}+} \subseteq R^{\widehat{\mathcal{I}}+}$ by Definition 3.8; so let us assume that R is transitive as well. By Definition 3.8 there is a sequence $u = z_1, \ldots, z_n = v$ of elements of $\Delta^{\widehat{\mathcal{I}}}$ such that $\{(z_i, z_{i+1}) \mid i = 1, \ldots, n\} \subseteq S^{\widehat{\mathcal{I}}}$. By Proposition (3.9b)

$$\{(\delta(z_i), \delta(z_{i+1})) \mid i = 1, \ldots, n\} \subseteq S^{\mathcal{I}}, \text{ and}$$
$$\{(\delta(z_i), \delta(z_{i+1})) \mid i = 1, \ldots, n\} \subseteq R^{\mathcal{I}}$$

because $S \preceq R$ and the inclusion restrictions are satisfied by hypothesis. We can use Proposition (3.9f) for showing that $\{(z_i, z_{i+1}) \mid i = 1, \ldots, n\} \subseteq R^{\widehat{\mathcal{I}}}$, therefore $(u, v) \in R^{\widehat{\mathcal{I}}+}$ by construction (see Definition 3.8).

If S is not transitive and $(u, v) \in S^{\widehat{\mathcal{I}}+}$, then either $(u, v) \in S^{\widehat{\mathcal{I}}}$ or there is a transitive role S' s.t. $S' \preceq S$, $S \not\preceq S'$, and $(u, v) \in S'^{\widehat{\mathcal{I}}+}$. In the first case $(u, v) \in R^{\widehat{\mathcal{I}}}$ by Proposition (3.9f) and $R^{\widehat{\mathcal{I}}} \subseteq R^{\widehat{\mathcal{I}}+}$. In the second case $S' \preceq R$ because $S \preceq R$ and we can proceed as in the previous case in which S was transitive, by using S' in place of S.

(3.9d) We can proceed exactly as in the corresponding proof in Proposition 3.9. In fact, the proof relies on the previously shown (3.9b) and (3.9c).

□

Before verifying that whenever an interpretation satisfies a KB, the transitive closure of its unravelled interpretation satisfies the very same KB; we want to be sure that the interpretation we defined is a canonical quasi transitive shrub. This is what we are going to prove with the next proposition.

Proposition 3.11. *Let $\widehat{\mathcal{I}}^+ = (\Delta^{\widehat{\mathcal{I}}+}, \cdot^{\widehat{\mathcal{I}}+})$ be the transitive closure of the unravelled interpretation $\widehat{\mathcal{I}} = (\Delta^{\widehat{\mathcal{I}}}, \cdot^{\widehat{\mathcal{I}}})$. Then if $\mathcal{I}$ satisfies the role ordering, $\widehat{\mathcal{I}}^+$ is a canonical quasi transitive shrub interpretation.*

Proof. First we show that $\widehat{\mathcal{I}}^+$ is a quasi transitive shrub (Definition 3.2). The first properties (3.2a) and (3.2b) are trivially satisfied by definition.

(3.2c) Let $(u, \epsilon x)$ be a pair in $R^{\widehat{\mathcal{I}}+}$ for an arbitrary role name R. Summarising the Definition 3.8, either $(u, \epsilon x) \in R^{\widehat{\mathcal{I}}}$ or there is a transitive role $S \preceq R$ and a finite sequence of elements $z_1, \ldots, z_n$ of $\Delta^{\widehat{\mathcal{I}}}$ such that

$$\{(z_i, z_{i+1}) \mid i = 1, \ldots, n-1, z_1 = u, z_n = \epsilon x\} \subseteq S^{\widehat{\mathcal{I}}}.$$

In the first case $(u, \epsilon x)$ must be in the set (3.7a) (because $u \neq \epsilon$ by definition); therefore $\{u, \epsilon x\} \subseteq \mathcal{O}^{\widehat{\mathcal{I}}}$. In the second case, the same property can be shown for all the elements $z_{n-1}, \ldots, z_1$.

(3.2d) Let (u, vx) be a pair in $R^{\widehat{\mathcal{I}}+}$ for an arbitrary role name R.

If R is transitive then there is a finite sequence of element $z_1, \ldots, z_n$ of $\Delta^{\widehat{\mathcal{I}}}$ such that $\{(z_i, z_{i+1}) \mid i = 1, \ldots, n-1, z_1 = u, z_n = vx\} \subseteq R^{\widehat{\mathcal{I}}}$. Note that $z_{n-1} = v$ because v cannot be the empty sequence (see (3.7b)). If $n = 2$ then $u = z_{n-1} = v$; otherwise $(z_{n-1}, z_n) = (v, vx)$ is in $R^{\widehat{\mathcal{I}}}$ and $(u, z_{n-1}) = (u, v)$ is in $R^{\widehat{\mathcal{I}}}$ by definition (take the sequence $z_1, \ldots, z_{n-1}$ in Definition 3.8).

If R is not transitive, then $(u, vx) \in R^{\widehat{\mathcal{I}}}$ or $(u, vx) \in \bigcup_{S \in \mathcal{TRN}, S \preceq R, R \not\preceq S} S^{\widehat{\mathcal{I}}+}$. In the first case $v = u$ by definition (see (3.7b)). In the latter case, there is a transitive role S s.t. $S \preceq R$, $R \not\preceq S$ and $(u, vx) \in S^{\widehat{\mathcal{I}}+}$. We just showed that if the role is transitive then either $u = v$ or $\{(u, v), (v, vx)\} \subseteq S^{\widehat{\mathcal{I}}+}$. Moreover, $S^{\widehat{\mathcal{I}}+} \preceq R^{\widehat{\mathcal{I}}+}$ because $S \preceq R$.

(3.2e) Suppose $\{(u, v), (u, vx_1 \ldots x_n)\} \subseteq R^{\widehat{\mathcal{I}}+}$, we distinguish two cases where $n = 1$ and $n > 1$.

- If $n = 1$ then $(u, vx_1) \in R^{\widehat{\mathcal{I}}+}$. Since we already showed that (3.2d) holds, we can use it to conclude that either $u = v$ (i.e. $(u, vx_1) = (v, vx_1) \in R^{\widehat{\mathcal{I}}+}$) or $(v, vx_1) \in R^{\widehat{\mathcal{I}}+}$. In both cases the property is satisfied.
- Let us assume $n > 1$. Since $(u, v) \in R^{\widehat{\mathcal{I}}+}$, then $u \neq vx_1 \ldots x_i$ for all $i = 1, \ldots, n$ by construction (see Definitions 3.8 and 3.7).

  Firstly we show that for any role R, when $u \neq vx_1 \ldots x_i$ for all $i = 1, \ldots, n$ and $(u, vx_1 \ldots x_n) \in R^{\widehat{\mathcal{I}}+}$, then $(vx_1 \ldots x_{i-1}, vx_1 \ldots x_i) \in R^{\widehat{\mathcal{I}}+}$ for $i = 2, \ldots, n$. Since $(u, vx_1 \ldots x_n) \in R^{\widehat{\mathcal{I}}+}$, then either $u = vx_1 \ldots x_{n-1}$ or $\{(u, vx_1 \ldots x_{n-1}), (vx_1 \ldots x_{n-1}, vx_1 \ldots x_n)\} \subseteq R^{\widehat{\mathcal{I}}+}$. The first case can be ruled out because $u \neq vx_1 \ldots x_i$ for all $i = 1, \ldots, n$; therefore both $(u, vx_1 \ldots x_{n-1})$ and $(vx_1 \ldots x_{n-1}, vx_1 \ldots x_n)$ are in $R^{\widehat{\mathcal{I}}+}$. We can apply the same arguments to $(u, vx_1 \ldots x_{n-1})$ to conclude that both $(u, vx_1 \ldots x_{n-2})$ and $(vx_1 \ldots x_{n-2}, vx_1 \ldots x_{n-1})$ are in $R^{\widehat{\mathcal{I}}+}$ as well; this inductive argument can be carried on for each $vx_1 \ldots x_{n-\ell}$, as long as $n - \ell \geq 2$. Therefore $\{(u, vx_1 \ldots x_{n-i}), (vx_1 \ldots x_{i-1}, vx_1 \ldots x_i)\} \subseteq R^{\widehat{\mathcal{I}}+}$ for any $i = 2, \ldots, n$.

In addition, we can use the fact that $(u, vx_1x_2) \in R^{\widehat{\mathcal{I}}+}$ to conclude that (v, vx_1) by using the same arguments.

If R is transitive then, $(v, vx_1) \in R^{\widehat{\mathcal{I}}}$ and $(vx_1 \ldots x_{i-1}, vx_1 \ldots x_i) \in R^{\widehat{\mathcal{I}}}$ for any $i = 2, \ldots, n$ (by construction, see Definition 3.8). Therefore $\{(v, vx_1 \ldots x_i) \mid i = 1, \ldots, n\} \subseteq R^{\widehat{\mathcal{I}}+}$, so $(v, vx_1 \ldots x_n) \in R^{\widehat{\mathcal{I}}+}$.

If R is not transitive, then there is a transitive role $S \preceq R$ such that $(u, x_1 \ldots x_n) \in R^{\widehat{\mathcal{I}}+}$ because $n > 1$ by assumption, so $(u, vx_1 \ldots x_n)$ cannot be in $R^{\widehat{\mathcal{I}}}$. By using the same arguments as the transitive case we can conclude that $(v, vx_1 \ldots x_n) \in S^{\widehat{\mathcal{I}}+}$, therefore $(v, vx_1 \ldots x_n) \in R^{\widehat{\mathcal{I}}+}$ as well.

The next step is to show that $\widehat{\mathcal{I}}^+$ is canonical (Definiton 3.6).

(3.6b) Let $R_1, \ldots, R_n$ be role names and u, w elements of $\Delta^{\widehat{\mathcal{I}}+}$ such that $w \notin \mathcal{O}^{\mathcal{I}}$. We have to show that if $\{(u, w)\} \subseteq R_1{}^{\mathcal{I}} \cap \ldots \cap R_n{}^{\mathcal{I}}$ then there is a label L s.t. $\{R_1, \ldots, R_n\} \subseteq L$. Since $w \notin \mathcal{O}^{\mathcal{I}}$ and $\{(u, w)\} \subseteq R_1{}^{\mathcal{I}}$, then there are two elements v in $\Delta^{\widehat{\mathcal{I}}+}$ and x in Δ' such that $w = vx$ (by Definition (3.2c), because it has been proved above that $\widehat{\mathcal{I}}^+$ is a q.t. shrub). By using Definition (3.2d) together with the fact that $(u, vx) \in R_i{}^{\widehat{\mathcal{I}}+}$, for each role R_i, we can conclude that either $u = v$ or $\{(u, v), (v, vx)\} \subseteq R_i{}^{\widehat{\mathcal{I}}+}$. Note that if $u = v$, this must be true for all the roles $R_1, \ldots, R_n$ and the hypotesis would become $\{(u, ux)\} \subseteq R_1{}^{\mathcal{I}} \cap \ldots \cap R_n{}^{\mathcal{I}}$; on the other hand, v would be the same for all the roles again, therefore $\{(v, vx)\} \subseteq R_1{}^{\mathcal{I}} \cap \ldots \cap R_n{}^{\mathcal{I}}$. This shows that we can restrict ourselves to the case $\{(u, ux)\} \subseteq R_1{}^{\mathcal{I}} \cap \ldots \cap R_n{}^{\mathcal{I}}$ without loss of generality, which allows us to consider only the Definition 3.7 of unravelled interpretation, by ignoring the transitive closure.[9]

By the fact that the pair has the structure (u, ux), we know that for every role R_i

$$(u, ux) \in \left\{(u, ux) \in \Delta^{\widehat{\mathcal{I}}+} \times \Delta^{\widehat{\mathcal{I}}+} \mid (\delta(u), \delta(x)) \in R_i{}^{\mathcal{I}} \text{ and } R_i \in \lambda(x)\right\}$$

therefore $R_i \in \lambda(x)$ for all roles $R_1, \ldots, R_n$, which shows that there is a label $(\lambda(x))$ which includes all the roles.

[9]The role hierarchy does not add any pair (u, ux) to a role because of the Property (3.9c) and the fact that $\mathcal{I}$ satisfies the role ordering by hypothesis.

(3.6a) Let u, v, w be different elements in $\Delta^{\widehat{\mathcal{I}}+}$ such that $w \notin \mathcal{O}^{\mathcal{I}}$ and

$$\{(u, v), (v, w), (u, w)\} \subseteq R^{\widehat{\mathcal{I}}+}.$$

If R is transitive then (3.6a) is verified by definition because $R \preceq R$. So let R be non–transitive: this means that either $(u, w) \in R^{\widehat{\mathcal{I}}}$ or there is a transitive role S s.t. $S \preceq R$, $R \not\preceq S$, and $(u, w) \in S^{\widehat{\mathcal{I}}+}$ (see Definition 3.8). Analogously, either $(v, w) \in R^{\widehat{\mathcal{I}}}$ or there is a transitive role S' s.t. $S' \preceq R$, $R \not\preceq S'$, and $(v, w) \in S'^{\widehat{\mathcal{I}}+}$. Let us consider this case by case.

- If (u, w) is in $R^{\widehat{\mathcal{I}}}$, then $w = ux$ for some $x \in \Delta'$ because $w \notin \mathcal{O}^{\widehat{\mathcal{I}}+}$ (see Definition 3.7). If $(v, w) \in R^{\widehat{\mathcal{I}}}$ then $w = vx$ as well, therefore $u = v$. This is in contradiction with the assumption that u and v are distinct.
- Let (u, w) be in $R^{\widehat{\mathcal{I}}}$ and (v, w) be in $S'^{\widehat{\mathcal{I}}+}$. Then $w = ux$ and there are $z_1, \ldots, z_n$ in $\Delta^{\widehat{\mathcal{I}}}$ s.t. $z_1 = v$, $z_n = w$, and $\{(z_i, z_{i+1}) \mid i = 1, \ldots, n-1\} \subseteq S'^{\widehat{\mathcal{I}}}$. Since $w = ux$ then $z_{n-1} = u$, therefore $(u, w) = (z_{n-1}, z_n)$ is in $S'^{\widehat{\mathcal{I}}}$ which is included in $S'^{\widehat{\mathcal{I}}+}$.
- If (u, w) is in $S^{\widehat{\mathcal{I}}+}$, and (v, w) is in $R^{\widehat{\mathcal{I}}}$, then we can conclude that (v, w) is in $S^{\widehat{\mathcal{I}}+}$ as well by using the very same arguments as for the previous case.
- Let (u, w) be in $S^{\widehat{\mathcal{I}}+}$and (v, w) be in $S'^{\widehat{\mathcal{I}}+}$. Then there are $z_1, \ldots, z_n$ and $z'_1, \ldots, z'_{n'}$ in $\Delta^{\widehat{\mathcal{I}}}$ s.t. $z_1 = u$, $z'_1 = v$, $z_n = z'_{n'} = w$,

$$\{(z_i, z_{i+1}) \mid i = 1, \ldots, n-1\} \subseteq S^{\widehat{\mathcal{I}}},$$

  and

$$\{(z'_i, z'_{i+1}) \mid i = 1, \ldots, n'-1\} \subseteq S'^{\widehat{\mathcal{I}}}.$$

  As in the previous cases we note that $z_{n-1} = z'_{n'-1}$; therefore $(z_{n-1}, w) \in S^{\widehat{\mathcal{I}}} \cap S'^{\widehat{\mathcal{I}}}$. Since $S^{\widehat{\mathcal{I}}} \subseteq S^{\widehat{\mathcal{I}}+}$ by definition (see Definition 3.8), and $S'^{\widehat{\mathcal{I}}} \subseteq S'^{\widehat{\mathcal{I}}+}$, then $(z_{n-1}, w) \in S^{\widehat{\mathcal{I}}+} \cap S'^{\widehat{\mathcal{I}}+}$.

  By using the already proved Property (3.6b), we know that there is a label L containing both S and S'; by Definition (3.4b), either $S' \lesssim S$ or $S \lesssim S'$. Note that by assumption neither S nor S' can have any functional role which includes them (or be functional themselves). Let us assume that $S' \lesssim S$; since S' is not functional then either $S' \preceq S$ or there is a role R' such that $S' \preceq R'$ and $R' \lesssim S$. The first case is our goal, while in the latter case we can assume that R' must not be functional because S' cannot be included into a functional role. Since the ordering $\lesssim$ is well defined, sooner or later we are going to hit S without reaching any functional role; therefore the result is a chain of inclusions which shows that $S' \preceq S$ by the transitivity of $\preceq$.[10] Assuming that $S \lesssim S'$ we can conclude that $S \preceq S'$ by using the same arguments we used for the dual case.

Therefore we have just shown that either $S' \preceq S$ or $S \preceq S'$. Let us consider the case in which $S \preceq S'$. We assumed that $\mathcal{I}$ satisfies the role ordering, then by using (3.9b) and (3.9f) it is easy to show that if $S \preceq S'$ then $S^{\widehat{\mathcal{I}}^+} \subseteq S'^{\widehat{\mathcal{I}}^+}$ (see the proof for Lemma 3.12). Therefore both (u, w) and (v, w) are in $S'^{\widehat{\mathcal{I}}^+}$. If we consider the case where $S' \preceq S$ we obtain the same result with S instead of S'.

$\square$

3.3 Completeness of q.t. shrub interpretations

Now all the pieces are in place for showing the completeness of the described models. Obviously, if a knowledge base has a quasi transitive shrub model it has a model. The difficult bit is showing that having an unrestricted model implies that there is a quasi transitive shrub model as well. Clearly, the complex translation which has been described in the previous sections provides the basis for such a proof. We start by showing that given an arbitrary interpretation satisfying a KB, the transitive closure of its unravelled interpretation satisfies the KB as well.

Lemma 3.12. *Let Σ be a knowledge base, $\preceq$ be the role ordering induced by Σ, and $\mathcal{TRN}$ $\mathcal{FRN}$ be the transitive and functional role names. We assume that $\mathcal{TRN}$ and $\mathcal{FRN}$ are consistent w.r.t. $\preceq$. If an arbitrary interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ satisfies Σ, then Then the transitive closure of its unravelled interpretation $\widehat{\mathcal{I}}^+ = (\Delta^{\widehat{\mathcal{I}}^+}, \cdot^{\widehat{\mathcal{I}}^+})$ satisfies Σ as well.*

Proof. We prove the lemma by considering all the assertions in Σ and showing that they are satisfied by $\widehat{\mathcal{I}}^+$. Since $\mathcal{I}$ satisfies Σ, the requirement for Proposition 3.10 are satisfied. Therefore we are guaranteed that the properties in Proposition 3.9 hold for $\widehat{\mathcal{I}}^+$.

[10]The last point can be demostrated more formally by induction on $\lesssim$, but we think that it is clear enough.

- Let $C \sqsubseteq D$ be a concept axiom in Σ. Assume that the axiom is not satisfied, then there is an element u such that $u \in C^{\widehat{\mathcal{I}}+}$ and $u \notin D^{\widehat{\mathcal{I}}+}$. By (3.9d), $\delta(u) \in C^{\mathcal{I}}$ and $\delta(u) \notin D^{\mathcal{I}}$ therefore $\mathcal{I}$ do not satisfies Σ. This is in contradiction with the hypothesis, therefore $C \sqsubseteq D$ must be satisfied.

- Let $S \sqsubseteq R$ be a role axiom in Σ; then $S \preceq R$ by definition. Let (u,v) be a pair in $S^{\widehat{\mathcal{I}}+}$, then $(\delta(u),\delta(v)) \in S^{\mathcal{I}}$ by (3.9b). We can conclude that $(u,v) \in R^{\widehat{\mathcal{I}}+}$ by using (3.9f).

- If $R \in \mathcal{TRN}$, then $R^{\widehat{\mathcal{I}}+}$ is transitive by construction (see Definition 3.8). This can be proved by considering that if $\{(u,v),(v,z)\} \in R^{\widehat{\mathcal{I}}+}$, then we have two paths in $R^{\widehat{\mathcal{I}}}$: one connecting u to v, and a second connecting v to z. Therefore there is a path in $R^{\widehat{\mathcal{I}}}$ connecting u to z.

- If R is functional, then for any $u \in \Delta^{\widehat{\mathcal{I}}+}$,

$$\sharp\left\{v \mid (u,v) \in R^{\widehat{\mathcal{I}}+}\right\} = \sharp\left\{x \mid (\delta(u),x) \in R^{\mathcal{I}}\right\}$$

  by (3.9e). In addition, $\sharp\left\{x \mid (\delta(u),x) \in R^{\mathcal{I}}\right\} \le 1$ because $R^{\mathcal{I}}$ is functional; therefore $R^{\widehat{\mathcal{I}}+}$ is functional as well.

- Let $a{:}C$ be a concept assertion in Σ. It is easy to see that $a^{\widehat{\mathcal{I}}+} = \epsilon a^{\mathcal{I}}$, therefore $\delta(a^{\widehat{\mathcal{I}}+}) = a^{\mathcal{I}}$. The assumption that $a^{\widehat{\mathcal{I}}+} \notin C^{\widehat{\mathcal{I}}+}$ implies that $\delta(a^{\widehat{\mathcal{I}}+}) \notin C^{\mathcal{I}}$ by (3.9d). Obviously this is a contradiction with the hypothesis that $\mathcal{I}$ satisfies Σ; therefore $a^{\widehat{\mathcal{I}}+} \in C^{\widehat{\mathcal{I}}+}$.

- Let $(a,b){:}R$ be a role assertion in Σ. Let us consider the pair $(a^{\widehat{\mathcal{I}}+}, b^{\widehat{\mathcal{I}}+})$: by Definition 3.8 $a^{\widehat{\mathcal{I}}+} = a^{\widehat{\mathcal{I}}}$ and $b^{\widehat{\mathcal{I}}+} = b^{\widehat{\mathcal{I}}}$. Note that $(a^{\widehat{\mathcal{I}}}, b^{\widehat{\mathcal{I}}}) \in \mathcal{O}^{\widehat{\mathcal{I}}} \times \mathcal{O}^{\widehat{\mathcal{I}}}$, and $(\delta(a^{\widehat{\mathcal{I}}}), \delta(b^{\widehat{\mathcal{I}}})) = (a^{\mathcal{I}}, b^{\mathcal{I}})$. Moreover, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ because $\mathcal{I}$ satisfies the role assertion. Therefore, $(a^{\widehat{\mathcal{I}}+}, b^{\widehat{\mathcal{I}}+}) \in R^{\widehat{\mathcal{I}}}$ because the pair is in the set (3.7a), and $(a^{\widehat{\mathcal{I}}+}, b^{\widehat{\mathcal{I}}+}) \in R^{\widehat{\mathcal{I}}+}$ because $R^{\widehat{\mathcal{I}}} \subseteq R^{\widehat{\mathcal{I}}+}$.

□

We can now conclude by stating the completeness of quasi transitive shrub models w.r.t. our logic. In Chapter 7 we present a slightly different version of the present theorem for deduction instead of satisfiability.

Theorem 3.13. *A knowledge base Σ is satisfiable iff there is a canonical quasi transitive shrub interpretation satisfying it.*

Proof. The “only if” direction direction is trivial because if there is a quasi transitive shrub interpretation satisfying Σ, then Σ is satisfiable by definition. For the “if” direction, let be $\mathcal{I}$ an interpretation satisfying Σ, and $\widehat{\mathcal{I}}^+ = (\Delta^{\widehat{\mathcal{I}}^+}, \cdot^{\widehat{\mathcal{I}}^+})$ the transitive closure of its unravelled interpretation. According to Lemma 3.12 $\widehat{\mathcal{I}}^+$ satisfies Σ as well; in addition $\widehat{\mathcal{I}}^+$ is a canonical quasi transitive shrub interpretation by Proposition 3.11. □

3.4 Application to terminological reasoning

We can use the very same technique to establish the completeness of the class of interpretations we described w.r.t. terminological reasoning. In particular we concentrate on the problem of verifying the satisfiability of a concept expression w.r.t. a given terminology (see Chapter 2). We have already shown that the structure of models for a generic kb is a set of quasi transitive trees “glued” together through their roots. For terminological reasoning (i.e. without individuals) models are much simpler because they are single transitive trees. This property is exploited in Chapter 5 to show the completeness of the presented kb satisfiability algorithm.

A given concept C is satisfiable w.r.t. a terminology $\mathcal{T}$ iff there is an interpretation $\mathcal{I}$ satisfying $\mathcal{T}$, such that the set $C^{\mathcal{I}}$ is non empty. The property we want to show is that this is the case iff there is a quasi transitive shrub interpretation $\mathcal{I}_t$ s.t.: it satisfies $\mathcal{T}$, it is a single quasi transitive tree, and the root of this tree is in $C^{\mathcal{I}_t}$. These properties will be used in Chapter 5.

We notice that the given satisfiability problem can easily be reduced to knowledge base satisfiability (see Schaerf [1994]). In fact, C is satisfiable w.r.t. a terminology $\mathcal{T}$ iff the knowledge base $\Sigma_C = \langle \mathcal{T}, \{a{:}C\} \rangle$ is satisfiable (where a is an arbitrary individual name). This can easily be proved by considering that if $\mathcal{I}$ is an interpretation satisfying $\mathcal{T}$, such that $C^{\mathcal{I}}$ is not empty; then, we can build an intrepretation for Σ_C by mapping a to one of the elements of $C^{\mathcal{I}}$. Conversly, if $\mathcal{I}$ satisfies Σ_C then $a^{\mathcal{I}}$ is an element of $C^{\mathcal{I}}$; therefore C is satisfiable w.r.t. $\mathcal{T}$.

We use this reduction to establish the completeness result. Let us start from an arbitrary interpretation $\mathcal{I}$ satisfying C. As we have just shown, there is a direct way of obtaining an interpretation $\mathcal{I}'$ satisfying Σ_C, by taking $\mathcal{I}'$ as equal to $\mathcal{I}$ but for the addition of the mapping from a to a given element of $C^{\mathcal{I}}$. As shown in Section 3.2.1 we build $\widehat{\mathcal{I}'}^+$ as the transitive closure of the unravelled interpretation of $\mathcal{I}'$.

The main idea is “extracting”, from $\widehat{\mathcal{I}'}^+$ the single tree rooted in $a^{\widehat{\mathcal{I}'}^+}$. Since in Σ_C there are not any role assertion, all the q.t. trees in $\widehat{\mathcal{I}'}^+$ are unconnected, and there cannot be any loop on the root $a^{\widehat{\mathcal{I}'}^+}$ (see Definition 3.7). Formally, we extract an interpretation $\mathcal{I}_t = (\Delta^{\mathcal{I}_t}, \cdot^{\mathcal{I}_t})$ by selecting all the elements of $\Delta^{\widehat{\mathcal{I}'}^+}$ connected to $a^{\widehat{\mathcal{I}'}^+}$:

$$\begin{aligned}\Delta^{\mathcal{I}_t} = \{v \mid\ v = a^{\widehat{\mathcal{I}'}^+}, &\text{ or there is a finite sequence } \{z_1, \ldots, z_n\} \subseteq \Delta^{\widehat{\mathcal{I}'}^+} \setminus \mathcal{O}^{\widehat{\mathcal{I}'}^+} \\ &\text{and roles } R_1, \ldots, R_{n-1} \text{ s.t.} \\ &z_1 = a^{\widehat{\mathcal{I}'}^+}, z_n = v \text{ and } (z_i, z_{i+1}) \in R_i \text{ for } i = 1, \ldots, n-1\},\end{aligned}$$

and then we define the new interpretation function $\cdot^{\mathcal{I}_t}$ restricting the original $\cdot^{\widehat{\mathcal{I}'}^+}$ to the new domain:

$$\begin{aligned}A^{\mathcal{I}_t} &= A^{\widehat{\mathcal{I}'}^+} \cap \Delta^{\mathcal{I}_t} \\ R^{\mathcal{I}_t} &= R^{\widehat{\mathcal{I}'}^+} \cap \Delta^{\mathcal{I}_t} \times \Delta^{\mathcal{I}_t}\end{aligned}$$

By construction $\mathcal{I}_t$ is connected, in the sense that there is a path from the root $a^{\widehat{\mathcal{I}'}^+}$ to any other individual. It is not difficult to verify that $\mathcal{I}_t$ is still a canonical quasi transitive shrub interpretation satisfying $\mathcal{T}$, and the root is in $C^{\mathcal{I}_t}$.

3.5 Remarks

At this point we need to clarify the reason why the complicate mechanism that has been put in place was really necessary. The alternative technique can be derived from the algorithm used for showing the satisfiability of DL formulae w.r.t. a terminology (see Horrocks [1998]). It can be shown that if the formula is satisfiable the algorithm produces a pseudo–model from which an interpretation can be trivially built. It is not difficult to show that the interpretations generated by the pseudo–models satisfy the properties we are interested in. This method is very good for showing the completeness of the class of interpretations w.r.t. the problem of kb satisfiability (Section 3.3), but it has the disadvantage that the connection between an arbitrary interpretation and one of the interpretations belonging to the restricted class cannot easily be established.

This connection is not necessary for kb satisfiability, but we need it for the problem of query answering (see Horrocks and Tessaris [2000]) when the query language is different from the assertional language (see Section 7.1). As we will see in Chapter 6, query answering is strongly related to logical deduction; which, roughly speaking, is the problem of deciding whether a formula is verified in all the interpretations satisfying a knowledge base (written as $\Sigma \models \varphi$).

In general, when the knowledge base and the formulae share a common language closed under negation the problem $\Sigma \models \varphi$ is simply tranformed into the kb (un) satisfiability problem $\Sigma \cup \{\neg\varphi\}$. In the case of our language, this is not directly possible because the formula φ is written in a language strictly more expressive than the one of Σ. However, our goal is to reduce the problem to KB (un) satisfiability anyway, by introducing a transformation of the original formula φ into a kind of negation $\overline{\varphi}$, such that $\Sigma \models \varphi$ iff $\Sigma \cup \overline{\varphi}$ is not satisfiable.

The required transformation is inspired by the "tree–model property" of the DLs we consider (see Section 6.2.2); therefore we show a restricted version of the logical deduction problem in which we consider only interpretations belonging to a given class Γ.[11] I.e. deciding whether a formula is verified in all the interpretations belonging to the class Γ and satisfying a knowledge base (written as $\Sigma \models_\Gamma \varphi$).

The class of interpretations cannot be chosen arbitrarly, in fact we are interested in a class Γ such that $\Sigma \models \varphi$ iff $\Sigma \models_\Gamma \varphi$. Our candidate class is the one presented in this chapter, therefore we must show that it satisfies the required property. The "only if" direction is the easy bit, since $\Sigma \models \varphi$ takes into account all the interpretations, we must show that if $\Sigma \models_\Gamma \varphi$ then $\Sigma \models \varphi$.

The technique we will use for this proof (which will be presented in detail in Chapter 7) relies on the interpretation transformation presented in Section 3.2; in particular, on the ability to relate the original and the transformed interpretations in both directions. If we were interested in kb satisfiability only, we needed only the first direction; therefore the mere existence of an arbitrary interpretation belonging to the required class would have been enough.

[11] In particular we consider the class of q.t. shrub canonical w.r.t. the kb (see Section 3.1).

Chapter 4

KB satisfiability algorithms

In this chapter we present a brief overview of the different methods for KB satisfiability presented in the literature, then we describe the precompletion technique, investigated in this thesis. In addition, we provide the motivation for our decision to concentrate on the precompletion method.

4.1 Alternative Abox reasoning techniques

In our analysis we concentrate on complete algorithms suitable for DLs whose expressivity is at least $\mathcal{ALC}$ with general axioms. This choice is dictated by the acknowledgement that the ability to express general axioms is essential for modern DL systems. For this reason, for example, we are not going to describe the structural algorithm used by the CLASSIC DL system.

4.1.1 Direct tableaux

In Section 2.2.2 we provided an example of verification for the satisfiability of a KB based on an extension of the tableaux–based technique presented in Section 2.2.1. In general, tableaux algorithms for terminological reasoning can be extended to hybrid reasoning without major changes.

The main idea is that the expansion starts with an initial set of constraints (see Equation 2.5) corresponding to the assertions in the Abox (see Buchheit et al. [1993], Haarslev and Möller [2000b], Horrocks et al. [2000b]).[1] Individual names are substituted by different variables in the constraint system; therefore if the unique name assumption holds for individuals, inequality constraints are added for imposing that two different individual names are never merged.

It is worth mentioning that up until now the fastest DL system providing Abox reasoning (described in Haarslev and Möller [2000a]) uses a direct tableaux technique.

We are interested to experiment with a technique which maintains the algorithmic distinction between Abox and Tbox. In addition we want to reuse terminological reasoners as they are (if possible). So we decided to not investigate the direct tableaux technique but a variation of it, called the precompletion technique, which will be described later on.

4.1.2 Encoding

The encoding technique reduces the KB satisfiability problem to the problem of verifying the satisfiability of a concept w.r.t. a terminology (see De Giacomo and Lenzerini [1996]). This is achieved by treating individuals as mutually disjoint newly introduced concept names, and encoding the Abox assertions as terminological axioms. For example the assertions $a{:}C$ and $\langle a, b\rangle{:}R$ are encoded as the two axioms $P_a \sqsubseteq C$ and $P_a \sqsubseteq \exists R.P_b$ respectively.

This idea alone does not provide a complete solution, since when concepts are used instead of individuals we are relaxing the fundamental property of an individual of being unique. In fact we cannot restrict the cardinality of the concept the individual has been substituted with, therefore it is possible that a knowledge base which is unsatisfiable because of this uniqueness restriction results in a satisfiable one once encoded.

A similar idea was exploited in Borgida and Patel-Schneider [1994] and Era and Donini [1992]; however in the first case a different semantics was adopted for the individual name,[2] while the second approach works only for not very expressive DLs.

In De Giacomo and Lenzerini [1996] the proposed algorithm deals with very expressive DLs, providing correct and complete reasoning. The trick they use is to ensure that, given an interpretation for the encoded KB, all the elements belonging to the same individual representative concept are indistinguishable. To guarantee this property they use terminology axioms to impose that if an element of a representative concept P_a has

[1]Note that terminological reasoning always starts with a single concept formula C to check, therefore corresponding to the single constraint $x{:}C$.

[2]In fact they relaxed the assumption that an individual is unique.

a property (i.e. belongs to a concept) C then all the elements of P_a have the same property.

Note that the concept C is not specified, and in principle there are infinitely many different expressions; therefore, infinitely many new axioms for each representative concept. However, they show that the number of different concept expressions they need is finite and polynomialy bound by the size of the original knowledge base.

Although the technique is a very powerful theoretical mechanism for investigating the computational properties of very expressive DLs, this method has not lead to any practical implementations. The encoding they suggests requires a very expressive underlying language (i.e. Converse PDL), for which no efficient reasoner is yet available;[3] it is also the case that the size of the generated terminology, although polynomial, soon becomes unmanageable when the number of individuals grows.

4.1.3 Resolution based methods

In spite of the fact that the DL community is strongly focused on tableaux–based methods, recently there have been some attempts to apply resolution–based methods to the KB satisfiability problem. We will not spend much time here on this approach since reasoning with Aboxes using resolution seems to be still at an early stage. However, some results have been obtained by translational methods[4] applied to terminological and Abox reasoning (see Hustadt and Schmidt [2000], Tammet [1995]). Note that a resolution–based terminological reasoner can be used in conjunction with the technique we are investigating (see Section 4.2).

Recently, a direct resolution–based method has been proposed for KB satisfiability (see Areces et al. [1999]). Although the DL they cover lacks some important features (for example the possibility of expressing general inclusion axioms) we think that more work in this subject can produce interesting results, since several modern computational logic tools are based on resolution methods rather than tableaux.

[3]In fact, no attempts have been made to investigate whether the encoding technique is applicable to different DLs.

[4]Modal or Description logics are translated into First Order Logic and then resolution methods are applied with an appropriate strategy for guaranteeing termination.

4.2 Precompletion

The main idea behind the precompletion technique is to split the KB satisfiability algorithm in two parts. In the first part all the information implicit in the role assertions is made explicit, generating what we call a "precompletion" of the knowledge base, then a terminological reasoner is used for verifying the consistency of the concept assertions for each individual name (see Hollunder [1996], Donini et al. [1994]).

The precompletion approach has been successfully used for both providing correct and complete algorithms, and analysing the complexity of the KB satisfiability problem. The works cited above focused on DL knowledge bases with empty terminologies, and languages not including transitive or functional roles.[5] With the work presented in this thesis we generalised the precompletion technique for KBs with general inclusion axioms, transitive roles, and functional roles (i.e. the DL $\mathcal{SH}f$).

Let us consider for example a very simple Abox containing only the assertions:

$$\mathcal{A} = \{a{:}\forall R.C, \langle a, b\rangle{:}R, b{:}\neg C\}\,.$$

The two first assertions can be used to derive the new assertion $b{:}C$; it is easy to realise that $\mathcal{A}$ is satisfiable iff $\mathcal{A}' = \mathcal{A} \cup \{b{:}C\}$ is satisfiable as well (which is not the case, because b cannot be in C and $\neg C$ at the same time). The interesting point is that when we check the satisfiability of $\mathcal{A}'$ we do not need to consider the role assertion, because its effects have been made "explicit" in the new assertion. Therefore we can verify the KB satisfiability by checking the satisfiability of the concepts $\forall R.C$ and $C \sqcap \neg C$ separately.

This technique consists of a correctness–preserving process which eliminates specific information regarding dependencies between individuals, while maintaining the consequences of such information. Once these dependencies are eliminated, the assertions about a single individual can be independently verified, ignoring the fact that an individual is involved. The precompletion, or elimination of the dependencies, is performed by adding new assertions using a set of nondeterministic syntactic rules. Because of the nondeterminism of the rules, many different precompletions can be derived from a single knowledge base, which is satisfiable if and only if at least one of these precompletions is satisfiable.

A subset of the tableaux rules for $\mathcal{SH}f$ (see Horrocks [1998]) is used to transform

[5]Even if the DLs included number restriction constructors, general axioms are needed for simulating functional restrictions.

the KB into one of its precompletions (see Figure 5.1). In the case of $\mathcal{SH}f$, the only "dropped" rule is the $\exists$-rule (see Figure 2.1), and this means that new variables are never generated.

The nondeterminism is introduced by the $\sqcup$-rule and possibly an exponential number of precompletions can be generated. On the other hand, since the number of variables is constant, the size of a precompletion is alway polynomial in the size of the original KB, and, most importantly, we do not need to worry about the possible non–termination of the algorithm (see Section 5.1).

The main advantage of this technique lies in the fact that the concept satisfiability tests can be performed by using any available terminological reasoner (providing a sufficiently expressive DL language); in particular, impressive results have been recently obtained by optimised tableau–based systems like FaCT (see Franconi et al. [1998b]). Indeed, one of the primary motivations for reexamining the precompletion approach is the availability of such optimised systems.

The formal proof of correctness and completeness of the algorithm is provided in Chapter 5, here we sketch the actual algorithm and the technique used for proving its completeness.[6]

4.2.1 KB satisfiability algorithm

The input consists of a set containing the constraints (see Formulae 2.5) corresponding to the assertions in the Abox. In this section we introduce the algorithm for the less expressive language $\mathcal{SH}$ (i.e. without functional roles). The rules for this language are simpler and more intuitive, the full details for $\mathcal{SH}f$ are presented in the next chapter. The rules in Figure 4.1 are repeatedly applied to the initial set until either no rule is applicable or a contradictory combination of constraints is detected (a so-called *clash*).

The $\sqcup$-rule generates several alternatives branches, these are exhaustively explored by means of *backtrack points* where the algorithm restarts in case of a failure.

Intuitively, a clash corresponds to an unsatisfiable combination of constraints in the constraint set. When the precompletion process encounters a clash there is no need to continue adding new constraints, the precompletion will be unsatisfiable anyway; therefore the algorithm backtracks to the most recent backtrack point and continues the precompletion from that state.

If none of the rules is applicable a precompletion has been generated, and can

[6] In this case the correctness is the easy part.

$\mathcal{A} \rightarrow_{\sqsubseteq} \{o{:}C\} \cup \mathcal{A}$
if o is in $\mathcal{O}$, $\top \sqsubseteq C$ is in $\mathcal{T}$
and $o{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\sqcap} \{o{:}C_1,\ o{:}C_2\} \cup \mathcal{A}$
if $o{:}C_1 \sqcap C_2$ in $\mathcal{A}$,
and either $o{:}C_1$ or $o{:}C_2$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\sqcup} \{o{:}D\} \cup \mathcal{A}$
if $o{:}C_1 \sqcup C_2$ in $\mathcal{A}$,
and $D = C_1$ or $D = C_2$
and $o{:}D$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\forall} \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\forall R.C$ in $\mathcal{A}$, and $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$, and $S \preceq R$,
and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\forall+} \{o'{:}\forall R.C\} \cup \mathcal{A}$
if $o{:}\forall T.C$ in $\mathcal{A}$, $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$,
and there is $R \in \mathcal{TRN}$ such that $S \preceq R \preceq T$,
and $o'{:}\forall R.C$ is not in $\mathcal{A}$.

Figure 4.1: Precompletion rules for $\mathcal{SH}$

be checked for its satisfiability. This is done by gathering all the concept assertions concerning an individual, the concepts are then conjoined generating a new single concept associated to each single individual. These concepts are then verified by using an external call to a terminological reasoner; if they are all satisfiable then the algorithm terminates with success, otherwise it backtracks as in the case of clash detection.

If the process fails to find a satisfiable precompletion after all the nondeterministic branches have been explored, then it terminates with failure.

As presented here the algorithm sounds rather naive and prone to inefficient exploration of the search space. In fact, we need a few “tricks” to avoid a hopelessly slow algorithm. In Chapter 8 we will come back to this point and show how to improve the algorithm to obtain acceptable behaviour in most of the cases. Note that the theoretical worst case complexity of reasoning is ExpTime, therefore there can be pathological KBs manifesting this complexity.

Example 4.1

We use a slightly modified version of Example 2.3 to illustrate the precompletion algorithm. Let us consider the problem of checking the satisfiability of the Abox

$$\left\{\begin{array}{l} \texttt{sarah}:(\forall\texttt{FRIEND}.(\neg\texttt{Female}\sqcup(\forall\texttt{LOVES}.\texttt{Female}))), \\ \texttt{susan}:\texttt{Female}, \\ \texttt{andrea}:\exists\texttt{LOVES}.\neg\texttt{Female}, \\ \langle\texttt{sarah},\texttt{susan}\rangle:\texttt{FRIEND}, \\ \langle\texttt{sarah},\texttt{andrea}\rangle:\texttt{FRIEND}, \\ \langle\texttt{susan},\texttt{andrea}\rangle:\texttt{LOVES} \end{array}\right\},$$

with an empty terminology. This Abox corresponds to the one of Example 2.3 where the two constraints $\texttt{bill}:\neg\texttt{Female}$ and $\langle\texttt{andrea},\texttt{bill}\rangle:\texttt{LOVES}$ have been replaced by the constraint $\texttt{andrea}:\exists\texttt{LOVES}.\neg\texttt{Female}$. Note that this Abox is still unsatisfiable, it can be easily verified by using similar arguments to those used for the previously seen example; however we want to show its unsatisfiability by using the precompletion algorithm.

Using the $\forall$–rule with the first concept constraint and the two role constraints $\langle\texttt{sarah},\texttt{susan}\rangle:\texttt{FRIEND}$ and $\langle\texttt{sarah},\texttt{andrea}\rangle:\texttt{FRIEND}$ we add the two constraints

$$\left\{\begin{array}{l} \texttt{susan}:(\neg\texttt{Female}\sqcup(\forall\texttt{LOVES}.\texttt{Female})), \\ \texttt{andrea}:(\neg\texttt{Female}\sqcup(\forall\texttt{LOVES}.\texttt{Female})) \end{array}\right\}.$$

Both the newly added constraints are possible branching points where the $\sqcup$–rule is applicable; however if we choose to add the concept constraint $\texttt{susan}:\neg\texttt{Female}$ we obtain a contradiction straight away. Therefore we ignore that branch and we add the constraint

$$\left\{\ \texttt{susan}:(\forall\texttt{LOVES}.\texttt{Female})\ \right\}.$$

We can carry on with the deterministic rules and apply the $\forall$–rule with the latest added constraint and the role constraint $\langle\texttt{susan},\texttt{andrea}\rangle:\texttt{LOVES}$; this results in the further addition of the constraint

$$\left\{\ \texttt{andrea}:\texttt{Female}\ \right\}.$$

At this point, the only option is applying the $\sqcup$–rule to the constraint

$$\texttt{andrea}:(\neg\texttt{Female}\sqcup(\forall\texttt{LOVES}.\texttt{Female})).$$

The choice of adding the constraint $\texttt{andrea}:\neg\texttt{Female}$ can be immediately ruled out by the fact that it is in contradiction with the previously added $\texttt{andrea}{:}\texttt{Female}$, therefore we select the other disjunct and we add the constraint

$$\left\{ \texttt{andrea}{:}(\forall \texttt{LOVES}.\texttt{Female}) \right\}.$$

It is easy to verify that we have obtained a precompletion, since there are no further applicable rules; in addition, all the search space has been explored. Therefore, the original Abox is satisfiable iff this precompletion is satisfiable. The satisfiability check is performed by verifying the satisfiability of the concept constraints associated to each individual:

Individual	Concept
$\texttt{sarah}$	$(\forall \texttt{FRIEND}.(\neg\texttt{Female} \sqcup (\forall \texttt{LOVES}.\texttt{Female})))$
$\texttt{susan}$	$(\texttt{Female} \sqcap (\neg\texttt{Female} \sqcup (\forall \texttt{LOVES}.\texttt{Female})) \sqcap (\forall \texttt{LOVES}.\texttt{Female}))$
$\texttt{andrea}$	$((\exists \texttt{LOVES}.\neg\texttt{Female}) \sqcap (\neg\texttt{Female} \sqcup (\forall \texttt{LOVES}.\texttt{Female}))$ $\sqcap\ \texttt{Female} \sqcap (\forall \texttt{LOVES}.\texttt{Female}))$.

The first two concepts are satisfiable, while the third one is not because of the conjunction of $(\exists \texttt{LOVES}.\neg\texttt{Female})$ and $(\forall \texttt{LOVES}.\texttt{Female})$. Therefore the initial Abox is unsatisfiable.

4.2.2 Correctness and completeness of the algorithm

This section presents an overview of the proof for correctness and completeness of the precompletion technique; the full formal proof is provided in the next chapter.

The proof is divided in two parts: in the first we show that the set of precompletions characterises the satisfiability of the original knowledge base. The second part gives a method for checking the satisfiability of such a precompletion using a terminological reasoner.

The definition of a precompletion for a knowledge base $\Sigma = (\mathcal{T}, \mathcal{A})$ is given in a procedural way as a new KB $\Sigma_{pc} = (\mathcal{T}, \mathcal{A}_{pc})$ where the ABox $\mathcal{A}_{pc}$ is obtained by extending $\mathcal{A}$ using the nondeterministic syntactic rules in Figure 5.1 as long as they are applicable.

The precompletion rules are designed in such a way that, whatever strategy of application is chosen, the process of completing a knowledge base always terminates, with the same set of precompletions. In fact the only rule that does not introduce a smaller assertion in the knowledge base is the $\forall^+$–rule, but its applicability is bounded by the number of role assertions, which is invariant. The number of precompletions of a KB can be exponential because of the presence of a nondeterministic rule, however the size of each precompletion is polynomial with respect to the size of the original KB.

The set of all precompletions of a knowledge base characterises its satisfiability. If one of the precompletions is satisfiable then the original KB is trivially satisfiable because the process never removes constraints, therefore the assertions in the Abox are still in the precompletion. The other direction is proved by using a technique similar to that used in Horrocks and Sattler [1998]. A model $\mathcal{I}$ of Σ which witnesses its satisfiability is used to deterministically guide the application of precompletion rules to a unique satisfiable precompletion. For this purpose the only nondeterministic rule is transformed into its deterministic counterpart:

$$\begin{array}{l} \mathcal{A} \rightarrow_{\sqcup} \{o{:}D\} \cup \mathcal{A} \\ \qquad \text{if } o{:}C_1 \sqcup C_2 \text{ in } \mathcal{A}, \\ \qquad \text{and } D = C_1 \text{ if } o^{\mathcal{I}} \in C_1^{\mathcal{I}},\ D = C_2 \text{ otherwise} \\ \qquad \text{and } o{:}D \text{ is not in } \mathcal{A}. \end{array}$$

Satisfiability is preserved by rule application since if $\mathcal{I}$ is a model for the ABox before the application of the rule, then it is a model for the extended ABox as well. Therefore, given the fact that the terminology does not change, $\mathcal{I}$ is a model for the generated precompletion.

Now we know that we can concentrate on precompleted KBs without loss of generality. In $\mathcal{SH}$ all the contradictions can be detected by the terminological reasoner, but in $\mathcal{SH}f$ there can be clashes involving role assertions (see Definition 5.7). For this reason in Chapter 5 we will assume clash–free precompletions, since the presence of a clash makes a precompletion trivially unsatisfiable.

Given a precompleted knowledge base Σ_{pc}, for each individual o in the KB we consider the *individual concept* $\bigsqcap \mathcal{L}(\Sigma_{pc}, o)$, which is defined as the conjunction of all the concept expressions in the set $\{C \mid o{:}C \in \mathcal{A}\}$, or $\top$ if there are no assertions about o. It is clear that a model for the knowledge base is a model for every individual concept. We now show how, given models for the individual concepts, we can build a model for the knowledge base.

If each individual concept $\bigsqcap \mathcal{L}(\Sigma_{pc}, o)$ is separately satisfiable with respect to the terminology, then for every individual name o there is an *individual model* $\mathcal{I}_o =$

$(\Delta_o, \cdot^{\mathcal{I}_o})$, which witnesses the satisfiability. Interpretations can be infinite, but without loss of generality it can be assumed that their interpretation domains are pairwise disjoint, as well as having tree structures,[7] whose root elements (indicated by $\widehat{\mathcal{I}_o}$) are in the extension of the corresponding individual concept $\bigsqcap \mathcal{L}(\Sigma_{pc}, o)$ (as shown in Section 3.4).

The idea is to build a *union interpretation* $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ by "gluing" together all the tree interpretations, and adding new pairs to the interpretation of roles according to the role assertions.

The fact that the union interpretation is a model for the precompleted KB is intuitive for the propositional part (concept conjunction, disjunction and negation), but not so for the modal part. In particular, trouble can arise from the interaction between the universal quantification and the newly added pairs. However, the domain disjointness and tree structure assumptions guarantee that the properties of role interpretations overcome this problem.

- First, new links are added only to root nodes (those associated to individual names); in fact for each pair in $(x, y) \in R^{\overline{\mathcal{I}}}$, $x \in (\Delta_o \setminus \left\{o^{\overline{\mathcal{I}}}\right\})$ implies that (x, y) is in $R^{\mathcal{I}_o}$. An important consequence of this property is that if a role connects two individuals of different individual domains, then the first element of the pair must be a root individual.

- Second, there cannot be a connection between elements from different individual domains without a path passing through a root node. That is, given two distinct individuals u and v, whenever there is a pair $(u^{\overline{\mathcal{I}}}, x) \in R^{\overline{\mathcal{I}}}$ with $x \in \Delta_v$, there should be a transitive role S included in R such that both pairs $(u^{\overline{\mathcal{I}}}, v^{\overline{\mathcal{I}}})$, $(v^{\overline{\mathcal{I}}}, x)$ are in $S^{\overline{\mathcal{I}}}$, or $x = v^{\overline{\mathcal{I}}}$.

  This property ensures that elements in different individual domains can interact only via individual names in the Abox (i.e. the roots). Therefore, restrictions like $\forall R.C$ holding inside an individual model cannot affect elements belonging to other individual domains.

It is easy to see that interpretation of roles satisfies the role assertions in the KB, because of their simple form; however concept assertions are more problematic. In fact, although the interpretation of a concept name is simply the union of the interpretations of individual models, the extension of some concept expressions can violate the

[7] Actually, the structure is more tree–like, as explained in Chapter 3, but here we can consider it simply as a tree.

semantics of concept forming constructors because of the presence of new pairs in the interpretation of roles.

Example 4.2
For example, given the simple precompleted knowledge base $\langle \emptyset, \{\langle a, a\rangle{:}R\}\rangle$, we can build an individual model $\mathcal{I}_a$ for the label of a. We choose the interpretation domain $\Delta_a = \{0\}$ with the interpretation function $a^{\mathcal{I}_a} = 0$, and $C^{\mathcal{I}_a} = R^{\mathcal{I}_a} = \emptyset$. The union interpretation maps R to $\{(0, 0)\}$ and a to 0.

Let us consider the concept expression $\forall R.C$, by the semantics of the universal constructor (see Table 2.1) $a^{\mathcal{I}_a} \in (\forall R.C)^{\mathcal{I}_a}$ because $a^{\mathcal{I}_a}$ is not related to any other element via R. However, in the union interpretation this is no longer true because $a^{\mathcal{I}_a}$ is related to itself via $R^{\mathcal{I}_a}$, but $a^{\mathcal{I}_a}$ cannot be in $C^{\mathcal{I}_a} = \emptyset$.

This problem is localised to root individuals, because the interpretation of roles restricted to non–root individuals does not change. In fact, the interpretation of arbitrary concept expressions, restricted to non–root individuals is a monotonic extension of the one in individual models; i.e. for any concept expression D, $D^{\mathcal{I}_o} \setminus \left\{o^{\overline{\mathcal{I}}}\right\} \subseteq D^{\overline{\mathcal{I}}}$ for each individual name o.

For root individuals this property does not hold for arbitrary concept expressions; however, the property is still valid for concept expressions which appear as assertions in the precompleted KB. In fact, for each assertion $o{:}D \in \mathcal{A}_{pc}$ the individual $a^{\overline{\mathcal{I}}}$ is in the extension $D^{\overline{\mathcal{I}}}$. This restricted property (together with the more general applicability to non–roots) is sufficient to prove that all the axioms and assertions in the precompleted knowledge base are satisfied.

4.2.3 Pros and cons of precompletion

The initial motivation of our research was exploring the feasibility of providing an Abox reasoner for the DL of the FaCT system ($\mathcal{SH}f$). In particular we are interested in the differences in developing DL systems with and without Abox. For this reason we decided to investigate the precompletion technique, since it enables a clear distinction between the hybrid reasoning and the terminological component of the algorithm.

The algorithm is completely independent from the terminological reasoner which is used for the concept satisfiability test. In this way, an Abox can be easily added to most existing DL systems without re-implementing the whole system. Moreover, optimisation strategies implemented at the terminological level do not adversely interact with the precompletion algorithm.

A further advantage of precompletion is that it confines the exponential blow of the complexity to the terminological reasoning. In fact, the size of precompletions is polynomial in the size of the KB. This may indicate that a system based on precompletion can run with a smaller memory footprint compared to a system based on an extension of the Tableaux method.[8]

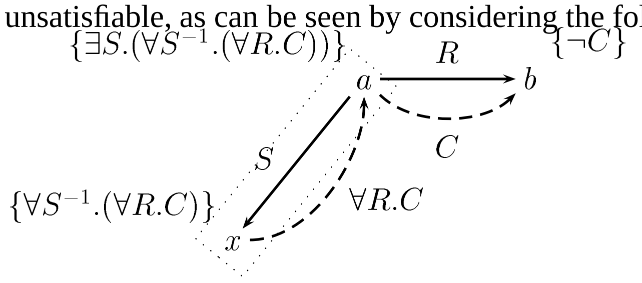
On the other hand, precompletion presents two big disadvantages w.r.t. other techniques. The first one is related to the fact that the precompletion process cannot obtain useful information from the terminological reasoner (just a binary yes/no). This means that in case of failure of a concept satisfiability test, the precompletion process cannot use additional information for improving heuristics in the exploration of the search space.

However, we think that the major drawback lies in the fact that the limits on the expressiveness of DLs suitable for this approach are not clear. For example, the extension of precompletion to the inverse role constructor is problematic.

Example 4.3
Consider for example the Abox

$$\left\{a{:}\exists S.(\forall S^{-1}.(\forall R.C)), b{:}\neg C, \langle a,b\rangle{:}R\right\}.$$

This Abox is unsatisfiable, as can be seen by considering the following diagram


$\{\exists S.(\forall S^{-1}.(\forall R.C))\}$
R
$\{\neg C\}$
a
b
C
S
$\{\forall S^{-1}.(\forall R.C)\}$
$\forall R.C$
x

The key point is the combination of the existential and universal quantification $\exists S.\forall S^{-1}.(\ldots)$ which pushes "new" restrictions on the individuals a and b (i.e. the concept $\forall R.C$ on a, and consequently the concept C on b). Note that the Abox is already precompleted, and the concept $\exists S.(\forall S^{-1}.(\forall R.C))$ is satisfiable; therefore a naive application of precompletion leads to a wrong result.

Precompletion is failing in this case because it does not take into account the fact that new concepts may have to be added to the label of an individual because of the terminological reasoning.

[8]This actually depends on the kind of optimisation implemented in the DL system. As we are going to suggest later on, the precompletion technique can be used as a testbed for improving strategies for Tableaux based systems.

In the next chapter we prove that the precompletion technique can be used with the DL $\mathcal{SH}f$. From the precise description of the technique an actual algorithm can be easily devised. We have implemented a prototype of an Abox reasoner based on precompletion, and we performed some evaluation experiments with it. The system and the results are described in Chapter 8.

Chapter 5

Precompletion algorithm for $\mathcal{SH}f$

The proof for correctness and completeness of the technique is in two parts: in Section 5.1 we present a method for deriving a set of simpler knowledge bases called *precompletions*, and we show that this set characterises the satisfiability of the original knowledge base. The second part, in Section 5.2 shows that the satisfiability of a precompletion can be verified using a terminological reasoner.

The purpose of the precompletion process is to generate a simpler (not smaller) knowledge base where the role assertions about individuals can be ignored. Precompletions of knowledge bases are built using a set of nondeterministic syntactic rules which extend the Abox of the original knowledge base. It will be proved that a knowledge base is satisfiable if and only if a satisfiable precompletion can be derived.

In Section 5.2 we show that for checking the satisfiability of a precompletion the role assertions can be ignored. In a precompletion the relevant elements are the concept assertions and the terminology. Each individual is associated to the conjunction of the concepts appearing in its concept assertions. If all those individual concepts are satisfiable with respect to the terminology, then we show that their models can be combined in an interpretation satisfying the precompleted knowledge base.

5.1 Precompletions of knowledge bases

Without loss of generality we assume that all the concept axioms in the Tbox are in the form $\top \sqsubseteq C$, where C is a concept expression. This assumption is not restrictive at all in DLs closed under negation. An arbitrary assertion $C_1 \sqsubseteq C_2$ can be transformed into the equivalent assertion $\top \sqsubseteq (\neg C_1 \sqcup C_2)$, which is in the required form.

A second assumption we adopt is that concept expressions are in *negation normal* *form*, where the $\neg\cdot$ constructor can appear only in front of concept names. Any concept expression can be transformed into an equivalent expression in normal form using the following rewriting rules.

$$\begin{array}{lll} \neg\neg C \equiv C & \neg(C \sqcap D) \equiv \neg C \sqcup \neg D & \neg\exists R.C \equiv \forall R.\neg C \\ & \neg(C \sqcup D) \equiv \neg C \sqcap \neg D & \neg\forall R.C \equiv \exists R.\neg C \end{array}$$

This assumption simplifies the description of the algorithm and the proofs detailed in this chapter.

Intuitively, a knowledge base is precompleted if all the information entailed by the presence of role assertions is exhibited in the form of concept assertions. The definition of a precompletion for a knowledge base is given in a procedural way as a new KB where the Abox is extended using the syntactic rules of Figure 5.1 as long as they are applicable.

We define a set of new binary operators $\cdot \approx_o \cdot$ (one for each individual name o), to simplify the formulae in the rules. Intuitively, they take into account the possible interaction between the role hierarchy and the functional restrictions (see Section 3.1.1). Two role names R and S are $\cdot \approx_o \cdot$ related if they are functional, and the Abox assertions force the R and S successors of the individual name o to be the same element. Strictly speaking, the operator depends on the KB as well, but for simplifying the notation we are omitting it. In places where the relation is used, it will be clear which is the KB we are referring to.

Definition 5.1. Given two roles R and S, an individual o, and a KB $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, the relation $R \approx_o S$ holds iff:

- there is a role $R_0 \preceq R$ s.t. either $o{:}\exists R_0.C_0$ or $\langle o, o' \rangle{:}R_0$ is in $\mathcal{A}$; and
- there is a role $S_0 \preceq S$ s.t. either $o{:}\exists S_0.D_0$ or $\langle o, o'' \rangle{:}S_0$ is in $\mathcal{A}$; and
- there is a set of roles $\{R_1, \dots R_{n-1}\}$, and a set of functional roles $\{F_1, \dots, F_n\}$, s.t.
  - either $o{:}\exists R_i.C_i$ or $\langle o, o' \rangle{:}R_i$ is in $\mathcal{A}$, for any $i = 1, \dots, n-1$ and
  - $R \preceq F_1$, $S \preceq F_n$, $R_1 \preceq F_1$, $R_1 \preceq F_2$, $R_2 \preceq F_2$, ..., $R_{n-1} \preceq F_n$.

The $\cdot \approx_o \cdot$ relation can be better understood by considering that it is describing a situation in which part of the role taxonomy looks like

$$\begin{tikzcd}
 & F_1 \arrow[dl, dash] \arrow[dr, dash] & & F_2 \arrow[dl, dash] \arrow[dr, dash] & & F_{n-1} \arrow[dl, dash] \arrow[dr, dash] & & F_n \arrow[dl, dash] \arrow[dr, dash] & \\
R \arrow[d, dash] & & R_1 & & \cdots & & R_{n-1} & & S \arrow[d, dash] \\
R_0 & & & & & & & & S_0
\end{tikzcd}$$

and the individual name o has a successor for each of the role names $R, R_1, \ldots, R_{n-1}, S$. In this case, the functional restrictions cause all these successors to be interpreted as the very same element. By the way it is defined, the relation $\cdot \approx_o \cdot$ is symmetric (i.e. $R \approx_o S \Rightarrow S \approx_o R$).

Proposition 5.2. *Let $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ be a KB, and $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation satisfying Σ. For any individual name o, role name R, S, and elements x, y of $\Delta^{\mathcal{I}}$. If $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$, $(o^{\mathcal{I}}, y) \in S^{\mathcal{I}}$, and $R \approx_o S$, then $x = y$.*

Proof. All the roles $R, S, R_1, \ldots, R_{n-1}$ of Definition 5.1 are functional because they are included in functional roles ($F_1, \ldots, F_n$); therefore $o^{\mathcal{I}}$ has at most one successor for any of these roles. We are going to show that all these successors are equal.

Note that for any R_i, the constraint $o{:}\exists R_i.C_i$ (or $\langle o, o' \rangle{:}R_i$) implies the existence of an element x_i in $\Delta^{\mathcal{I}}$ s.t. $(o^{\mathcal{I}}, x_i) \in {R_i}^{\mathcal{I}}$.

Let us consider $(o^{\mathcal{I}}, x_1) \in {R_1}^{\mathcal{I}}$, and $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$. Since ${R_1}^{\mathcal{I}} \subseteq {F_1}^{\mathcal{I}}$ and $R^{\mathcal{I}} \subseteq {F_1}^{\mathcal{I}}$, then $\left\{(o^{\mathcal{I}}, x_1), (o^{\mathcal{I}}, x)\right\} \subseteq {F_1}^{\mathcal{I}}$. From the functionality of ${F_1}^{\mathcal{I}}$ we can conclude that $x_1 = x$.

The very same arguments can be applied to all pair of roles R_i, R_{i+1}, including the last R_{n-1}, S; therefore $x = x_1 = x_2 = \ldots = x_{n-1} = y$. $\square$

Given this introductory definition, the transformation rules of Figure 5.1 should be clear, and we are going to introduce the formal definition of a precompletion of a KB.

Definition 5.3. A *precompletion* of a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is defined as a knowledge base $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$, where $\mathcal{A}_{pc}$ is an extension of $\mathcal{A}$ built by applying the syntactic rules in Figure 5.1 as long as their pre–conditions are satisfied.

The precompletion rules are designed in such a way that, whatever strategy of application is chosen, the process of completing a knowledge base always terminates (Proposition 5.4).

$\mathcal{A} \quad \rightarrow_{\sqsubseteq} \quad \{o{:}C\} \cup \mathcal{A}$
if o is in $\mathcal{O}$, $\top \sqsubseteq C$ is in $\mathcal{T}$
and $o{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\sqcap} \quad \{o{:}C_1,\ o{:}C_2\} \cup \mathcal{A}$
if $o{:}C_1 \sqcap C_2$ is in $\mathcal{A}$,
and either $o{:}C_1$ or $o{:}C_2$ if not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\sqcup} \quad \{o{:}D\} \cup \mathcal{A}$
if $o{:}C_1 \sqcup C_2$ is in $\mathcal{A}$,
and $D = C_1$ or $D = C_2$
and $o{:}D$ is not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\exists^1} \quad \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\exists R.C$ and $\langle o, o'\rangle{:}S$ are in $\mathcal{A}$,
$R \approx_o S$, and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\forall^1} \quad \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\forall R.C$ and $\langle o, o'\rangle{:}S$ are in $\mathcal{A}$,
there is $R' \preceq R$ s.t. $R' \approx_o S$
and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\forall} \quad \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\forall R.C$ is in $\mathcal{A}$, and $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$, and $S \preceq R$,
and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \quad \rightarrow_{\forall+} \quad \{o'{:}\forall R.C\} \cup \mathcal{A}$
if $o{:}\forall T.C$ in $\mathcal{A}$, $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$,
and there is $R \in \mathcal{TRN}$ such that $S \preceq R \preceq T$,
and $o'{:}\forall R.C$ is not in $\mathcal{A}$.

Figure 5.1: Precompletion rules for $\mathcal{SH}f$

Proposition 5.4. *The precompletion process always terminates, and any precompletion has a size which is polynomial w.r.t. the size of the knowledge base.*

(Sketched). The number of new assertions introduced by the terminology via the $\rightarrow_{\sqsubseteq}$ rule is equal to the number of individual names in the KB multiplied by the number of concept axioms in the Tbox. The rules $\rightarrow_{\sqcap}$, $\rightarrow_{\sqcup}$, $\rightarrow_{\exists^1}$, $\rightarrow_{\forall^1}$, and $\rightarrow_{\forall}$ always introduce assertions smaller than the original ones. The only rule that introduces non decreasing assertions is $\rightarrow_{\forall+}$; however, its applicability is bounded by the number of role assertions, which is invariant.

For estimating the size of each precompletion we can use the argument that the number of different concept expressions that can be generated is polynomial w.r.t. the size of the KB.[1] Therefore the size of a precompletion cannot exceed the number of individual names multiplied by the number of concept expressions, and this number is still polynomial w.r.t. the size of the KB. $\square$

The precompletion algorithm generates a finite, but possibly exponential, number of precompletions; which can be individually checked for their consistency. The advantage over the original knowledge base is that they are simpler, enabling the use of techniques based on terminological reasoning.

The satisfiability of a knowledge base and the satisfiability of its precompletions are strictly related. In fact, the knowledge base is satisfiable if and only if at least one of its precompletions is satisfiable (Proposition 5.5). Since Proposition 5.4 ensures that the precompletions of a given knowledge base can be enumerated, the satisfiability checking can be performed on precompleted knowledge bases.

Proposition 5.5. *A knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is satisfiable if and only if it has a satisfiable precompletion.*

Proof. $\Leftarrow$ Since Σ is included in all its precompletions, a model for a precompletion Σ_{pc} is a model for Σ as well.

$\Rightarrow$ Given a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ for Σ, a satisfiable precompletion of Σ can be built. This precompletion $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$ is built by extending $\mathcal{A}$ using a set of rules constrained by the model $\mathcal{I}$. The rules are the same as Figure 5.1 apart from the nondeterministic $\rightarrow_{\sqcup}$ rule, which is transformed into a deterministic one by using the model $\mathcal{I}$:

[1] Again the only problem may come from the $\rightarrow_{\forall+}$ rule, but the number of formulae that it can generate is limited by the number of transitive role names.

$\mathcal{A} \quad \rightarrow_{\sqcup} \quad \{o{:}D\} \cup \mathcal{A}$
if $o{:}C_1 \sqcup C_2$ in $\mathcal{A}$,
and $D = C_1$ if $o^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $D = C_2$ otherwise
and $o{:}D$ is not in $\mathcal{A}$.

All the rules preserve satisfiability, in the sense that if $\mathcal{I}$ is a model for the Abox before the application of the rule, then it is a model for the extended Abox as well:

$\rightarrow_{\sqsubseteq}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then every element of the domain is in $C^{\mathcal{I}}$ (including $o^{\mathcal{I}}$ for each $o \in \mathcal{O}$). Therefore the assertion $o{:}C$ is satisfied.

$\rightarrow_{\sqcap}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $o^{\mathcal{I}} \in (C_1 \sqcap C_2)^{\mathcal{I}}$. Therefore $o^{\mathcal{I}} \in C_1^{\mathcal{I}}$ and $o^{\mathcal{I}} \in C_2^{\mathcal{I}}$.

$\rightarrow_{\sqcup}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $o^{\mathcal{I}} \in (C_1 \sqcup C_2)^{\mathcal{I}}$; therefore either $o^{\mathcal{I}} \in C_1^{\mathcal{I}}$ or $o^{\mathcal{I}} \in C_2^{\mathcal{I}}$. Suppose that $o^{\mathcal{I}} \in C_1^{\mathcal{I}}$, then $\mathcal{A}$ is extended by adding the assertion $o{:}C_1$ which is satisfied by $\mathcal{I}$, and analogously for the case in which $o^{\mathcal{I}} \in C_2^{\mathcal{I}}$.

$\rightarrow_{\exists^1}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $(o^{\mathcal{I}}, o'^{\mathcal{I}}) \in S^{\mathcal{I}}$, and there is an element $x \in C^{\mathcal{I}}$ s.t. $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$. By Proposition 5.2 $x = o'^{\mathcal{I}}$; therefore the interpretation $\mathcal{I}$ satisfies the assertion $o'{:}C$ as well.

$\rightarrow_{\forall^1}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $(o^{\mathcal{I}}, o'^{\mathcal{I}}) \in S^{\mathcal{I}}$, and every element x s.t. $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$ must be in $C^{\mathcal{I}}$.

Since $R' \approx_o S$, there is a role $R_0 \preceq R'$ and an element x such that $(o^{\mathcal{I}}, x) \in {R_0}^{\mathcal{I}}$. In addition, $R' \preceq R$ therefore $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$, which means that $x \in C^{\mathcal{I}}$.

Finally, we can use Proposition 5.2 with R' and S for concluding that $x = o'^{\mathcal{I}}$, so $\mathcal{I}$ satisfies the assertion $o'{:}C$.

$\rightarrow_{\forall}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $o^{\mathcal{I}} \in (\forall R.C)^{\mathcal{I}}$, $(o^{\mathcal{I}}, o'^{\mathcal{I}}) \in S^{\mathcal{I}}$ and $S^{\mathcal{I}} \subseteq R^{\mathcal{I}}$. So $(o^{\mathcal{I}}, o'^{\mathcal{I}}) \in R^{\mathcal{I}}$ as well, therefore $o'^{\mathcal{I}} \in C^{\mathcal{I}}$.

$\rightarrow_{\forall^+}$ If $\mathcal{I}$ is a model for $\langle \mathcal{T}, \mathcal{A} \rangle$, then $o^{\mathcal{I}} \in (\forall T.C)^{\mathcal{I}}$, $(o^{\mathcal{I}}, o'^{\mathcal{I}}) \in S^{\mathcal{I}} \subseteq R^{\mathcal{I}} \subseteq T^{\mathcal{I}}$ and $R^{\mathcal{I}} = (R^{\mathcal{I}})^+$. If there is x in $\Delta^{\mathcal{I}}$ such that $(o'^{\mathcal{I}}, x) \in R^{\mathcal{I}}$, then $(o^{\mathcal{I}}, x) \in R^{\mathcal{I}}$ as well because $R^{\mathcal{I}}$ is transitive. The element x should be in $C^{\mathcal{I}}$ because $o^{\mathcal{I}} \in (\forall T.C)^{\mathcal{I}}$ and $R^{\mathcal{I}} \subseteq T^{\mathcal{I}}$; therefore $o'^{\mathcal{I}} \in (\forall R.C)^{\mathcal{I}}$.

Because of the preserved satisfiability, $\mathcal{I}$ must be a model for the precompleted knowledge base Σ_{pc} as well.

$\square$

The following Proposition 5.6 makes explicit the logical properties of precompletions implicit in the rules of Figure 5.1.

Proposition 5.6. *A precompleted $\mathcal{SH}f$ knowledge base $\Sigma_{pc} = \langle \mathcal{T}_{pc}, \mathcal{A}_{pc} \rangle$, containing assertions about the set of individual names $\mathcal{O}$, satisfies all the following conditions:*

$\rightarrow_{\sqsubseteq}$ *if $o \in \mathcal{O}$ and $\top \sqsubseteq C \in \mathcal{T}_{pc}$ then $o{:}C \in \mathcal{A}_{pc}$;*

$\rightarrow_{\sqcap}$ *if $a{:}C_1 \sqcap C_2 \in \mathcal{A}_{pc}$ then $a{:}C_1 \in \mathcal{A}_{pc}$ and $a{:}C_2 \in \mathcal{A}_{pc}$;*

$\rightarrow_{\sqcup}$ *if $a{:}C_1 \sqcup C_2 \in \mathcal{A}_{pc}$ then $a{:}C_1 \in \mathcal{A}_{pc}$ or $a{:}C_2 \in \mathcal{A}_{pc}$;*

$\rightarrow_{\exists^1}$ *if $\{o{:}\exists R.C, \langle a, b\rangle{:}S\} \subseteq \mathcal{A}_{pc}$, and $R \approx_o S$, then $b{:}C \in \mathcal{A}_{pc}$;*

$\rightarrow_{\forall^1}$ *if $\{o{:}\forall R.C, \langle a, b\rangle{:}S\} \subseteq \mathcal{A}_{pc}$, and there is $R' \preceq R$* s.t. *$R' \approx_o S$, then $b{:}C \in \mathcal{A}_{pc}$;*

$\rightarrow_{\forall}$ *if $\{a{:}\forall R.C, \langle a, b\rangle{:}S\} \subseteq \mathcal{A}_{pc}$ and $S \preceq R$ then $b{:}C \in \mathcal{A}_{pc}$;*

$\rightarrow_{\forall+}$ *if $\{a{:}\forall T.C, \langle a, b\rangle{:}S\} \subseteq \mathcal{A}_{pc}$ and there is a transitive role $R \in \mathcal{TRN}$ such that $S \preceq R \preceq T$, then $b{:}\forall R.C \in \mathcal{A}_{pc}$.*

Proof. Each property corresponds to the pre–conditions of one of the rules of Figure 5.1. So if the property is not satisfied, the corresponding rule is still applicable and Σ_{pc} is not yet a precompletion. $\square$

At this point we have reduced the problem of checking the satisfiability of a $\mathcal{SH}f$ knowledge base to the problem of verifying whether one of its precompletions is satisfiable. Now we turn our attention to precompleted KBs. The advantage in concentrating on such KBs is that the consistency can be verified using a terminological reasoner (Theorem 5.20).

5.2 Satisfiability of precompletions

In this section we show how the satisfiability of a precompletion can be reduced to the concept satisfiability problem. This reduction enables us to close the circle and leads to the main result for a general KB in Section 5.3.

Since we are going to ignore the role assertions of a precompleted KB, first we must make sure that a precompleted KB does not contain any contradiction caused by role assertions. In $\mathcal{SH}f$ this case is restricted to assertions involving functional roles.

Definition 5.7. A precompletion $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$ contains a *clash* iff there are two roles $R, R' \in \mathcal{RN}$, and individual names a, b, c with $b \neq c$, such that $R \approx_a R'$, and $\{\langle a, b\rangle{:}R, \langle a, c\rangle{:}R'\} \subseteq \mathcal{A}_{pc}$.

A precompletion containing a clash is not satisfiable because it violates some the functional restrictions; in fact, by Proposition 5.2 the interpretations of b and c must coincide, which is in contradiction with the unique name assumption. Precompletions not containing clashes are said to be *clash-free*.

Given a knowledge base, the *label* of an individual is the set of concept expressions in the assertions on the individual itself. This is formally defined in the next definition.

Definition 5.8. Given a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, the *label* of an individual $o \in \mathcal{O}$ with respect to Σ (written as $\mathcal{L}(\Sigma, o)$) is defined as the set of all concept expressions in the assertions about the individual name o:

$$\mathcal{L}(\Sigma, o) = \begin{cases} \{C \mid o{:}C \in \mathcal{A}\} & \text{if this set is not empty, or} \\ \{\top\} & \text{otherwise} \end{cases}$$

By using the label of an individual, the concept expression $\bigsqcap \mathcal{L}(\Sigma, o)$ is defined as the conjunction of all the concept expressions in $\mathcal{L}(\Sigma, o)$:

$$\bigsqcap \mathcal{L}(\Sigma, o) = C_1 \sqcap \ldots \sqcap C_n$$

where $\{C_i \mid i = 1, \ldots, n\} = \mathcal{L}(\Sigma, o)$.

Since in precompleted knowledge bases the information carried by role assertions is made explicit, all the relevant properties of an individual are in the form of concept assertions. The label of an individual completely characterises the properties of the individual, and it can be used to verify that these properties are not contradictory.

If each individual concept $\bigsqcap \mathcal{L}(\Sigma_{pc}, o)$ is separately satisfiable with respect to the terminology, then for every individual name o there is an *individual model* $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$, which witnesses the satisfiability. As seen in Section 3.4, it can be assumed that each individual model has a quasi transitive tree structure.

The domains of the models can be considered pairwise disjoint without loss of generality. Let $\mathcal{I}_u$ and $\mathcal{I}_v$ be two models of $\bigsqcap \mathcal{L}(\Sigma, u)$ and $\bigsqcap \mathcal{L}(\Sigma, v)$ respectively, whose domains Δ_u and Δ_v overlap. A new domain Δ'_u can be chosen having the same cardinality as Δ_u, and being disjoint from Δ_v, and a bijection can be defined between the sets Δ_u and Δ'_u. Moreover the defined bijection can be used to build a new interpretation in which every element of Δ_u is substituted by the mapped element in Δ'_u. It is easy to show that that new interpretation $\mathcal{I}'_u = (\Delta'_u, \cdot^{\mathcal{I}'_u})$ is a model for the corresponding concept $\bigsqcap \mathcal{L}(\Sigma, u)$ with respect to the given terminology.

Starting from the individual models $\mathcal{I}_o$, an interpretation for the overall knowledge base can be build by combining them. The domain of the union interpretation is the union of all the domains from each individual model. The interpretation function is defined in terms of the interpretation functions of each individual model:

- Each individual name is interpreted as the root of its corresponding model (i.e. $o^{\mathcal{I}_o}$), which is defined because to each individual name corresponds a label (see Definition 5.8).

- Concept names are interpreted as the union of their interpretations in the different models.

- Interpretation of role names is slightly more involved because the union of the single interpretations is not enough. In fact, the pairs corresponding to the role assertions in the Abox must be added as well (including those induced by both the role hierarchy, and transitive restriction on roles).

As anticipated, the interpretation of roles involves something more than the union of individual interpretations. First, the role assertions should be added; then, we must ensure that functional roles are still functional. The latter point can be understood by considering the precompletion of the KB $\Sigma = \langle \emptyset, \{\langle a, b\rangle{:}F, a{:}\exists F.C\}\rangle$, where F is a functional role. The precompletion of Σ is $\langle \emptyset, \{\langle a, b\rangle{:}F, a{:}\exists F.C, b{:}C\}\rangle$, which is satisfiable. From the precompletion, two individual models can be derived: $\mathcal{I}_a$ for the concept $\exists F.C$ and $\mathcal{I}_b$ for C. If we try to merge these two models, together with the pair generated by the role assertion $\langle a, b\rangle{:}F$, the resulting interpretation would not satisfy the functional restriction on F. The solution to this problem relies on a more careful definition of the union interpretation, which takes into account the functional restrictions on role extensions. For this purpose we introduce the notion of restricted interpretation of roles (Definition 5.9).

The idea is to remove the links that will be added later on by means of role assertions in the Abox. We perform this operation only on functional roles, because they are the problematic ones.

Definition 5.9. Let $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$ be a clash–free precompleted KB, and $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ be the individual model for the individual o in Σ_{pc}. The restricted individual interpretation function $\cdot^{\underline{\mathcal{I}_o}}$ for o is defined as:

$$o^{\underline{\mathcal{I}_o}} = o^{\mathcal{I}_o}$$

$$A^{\underline{\mathcal{I}_o}} = A^{\mathcal{I}_o}$$

$$R^{\underline{\mathcal{I}_o}} = \begin{cases} R^{\mathcal{I}_o} \setminus \left\{ (o^{\mathcal{I}_o}, x) \mid (o^{\mathcal{I}_o}, x) \in R^{\mathcal{I}_o} \right\} & \text{if there are } R', R'' \text{ s.t. } R \preceq R', \\ & \langle o, o' \rangle{:}R'' \in \mathcal{A}_{pc}, \text{ and } R' \approx_o R''; \\ R^{\mathcal{I}_o} & \text{otherwise.} \end{cases}$$

Since the restricted interpretation is different from the original one only for the role names, in the following we are going to use it only with role names.

Note that if R is transitive, then $R^{\underline{\mathcal{I}_o}} = R^{\mathcal{I}_o}$ because R does not have any functional super–role (see Definition 5.1).

Note that the restricted role interpretation depends on the knowledge base as well as on each individual model. It is used instead of the original interpretation function in Definition 5.10, where the extension of role names is defined in such a way that pairs removed by means of Definition 5.9 are substituted by pairs induced by Abox assertions. This is necessary only for non–transitive roles, because transitive roles cannot have functional super–roles by assumption, so the $\cdot \approx_o \cdot$ relations do not hold among transitive roles.

Then, the following Sections provide the formal proofs showing that this union interpretation is indeed a model for the precompleted KB.

Definition 5.10. Given a precompleted knowledge base $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$ and the individual models $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ for each individual $o \in \mathcal{O}$, then the *union interpretation*

$\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ is defined as:

$$\overline{\Delta} = \bigcup_{o \in \mathcal{O}} \Delta_o$$
$$o^{\overline{\mathcal{I}}} = o^{\mathcal{I}_o}$$
$$A^{\overline{\mathcal{I}}} = \bigcup_{o \in \mathcal{O}} A^{\mathcal{I}_o}$$
$$R^{\overline{\mathcal{I}}} \begin{cases} = \bigcup_{o\in\mathcal{O}} R^{\mathcal{I}_o} \cup \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\} & \text{if } R \text{ is not transitive,} \\ \qquad \cup \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\} & \\ \qquad \cup \bigcup_{S \preceq R, S \in \mathcal{TRN}} S^{\overline{\mathcal{I}}} & \\ = (\bigcup_{o\in\mathcal{O}} R^{\mathcal{I}_o} \cup \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\})^+ & \text{otherwise.} \end{cases}$$

For each $o \in \mathcal{O}$, $A \in \mathcal{CN}$, $R \in \mathcal{RN}$. The operator $\cdot^+$ builds the transitive closure of a relation. For an arbitrary binary relation ρ, it is defined as $\rho^+ = \bigcup_{n\geq 1} \rho^n$; where $\rho^1 = \rho$, and ρ^{n+1} is the composition of ρ and ρ^n, i.e. $\rho^{n+1} = \rho \circ \rho^n$.

Note that the definition is recursive because the union interpretation is used to build the interpretation for roles. Given the fact that we assume that the role hierarchy is acyclic (see Section 2.4), the interpretation of the roles is well defined.

The fact that the union interpretation is a model for the precompleted KB is intuitive for the propositional part (concept conjunction, disjunction and negation), but not so for the modal part. In particular, troubles can arise from the interaction between the universal quantification and the newly added pairs. However, we are going to prove that the domain disjointness and tree–like structure of the individual interpretations guarantee that the union interpretation is indeed a model for the precompleted KB.

5.2.1 Interpretation of roles

The disjointness of the domains guarantees that the individual model trees do not overlap, so the role interpretations $R^{\mathcal{I}_o}$ are pairwise disjoint. This property is crucial for the structure of the role extension in the union interpretation, since it ensures that new links are added only to root nodes (i.e. to those associated to individual names).

First we are going to show that restricted interpretations still satisfy the role hierarchy.

Proposition 5.11. *Let $\Sigma_{pc} = \langle\mathcal{T}, \mathcal{A}_{pc}\rangle$ be a clash–free precompleted KB, and $\mathcal{I}_o =$*

$(\Delta_o, \cdot^{\mathcal{I}_o})$ *be the individual model for the individual* o *in* Σ_{pc}. *For any pair of role names* S *and* R*, individual name* o*, if* $S \preceq R$ *then* $S^{\underline{\mathcal{I}_o}} \subseteq R^{\underline{\mathcal{I}_o}}$.

Proof. Since $\mathcal{I}_o$ satisfies the role hierarcy by hypothesis, $S^{\mathcal{I}_o} \subseteq R^{\mathcal{I}_o}$; therefore the only case in which $S^{\underline{\mathcal{I}_o}} \not\subseteq R^{\underline{\mathcal{I}_o}}$ is the one in which a pair is removed from $R^{\mathcal{I}_o}$ but not from $S^{\mathcal{I}_o}$. We are going to reason by contradiction, assuming that there is a pair $(o^{\mathcal{I}_o}, x) \in R^{\mathcal{I}_o} \cap S^{\underline{\mathcal{I}_o}}$ s.t. there are R', R'', $R \preceq R'$, $\langle o, o' \rangle{:}R'' \in \mathcal{A}_{pc}$, and $R' \approx_o R''$ (see Definition 5.9). This generates a contradiction because $S \preceq R \preceq R'$, therefore the pair should have been eliminated from $S^{\mathcal{I}_o}$ as well. □

Now we consider the relation between the union interpretation and the individual interpretations. We start by verifying that no pair of elements, coming from the same individual domain, is in the union interpretation of a role unless it is in the individual interpretation of the role itself. Naturally, we must exclude the case of a pair induced by an Abox interpretation like $\langle a, a \rangle{:}R$, which forces the pair $(a^{\mathcal{I}_a}, a^{\mathcal{I}_a})$ in $R^{\overline{\mathcal{I}}}$.

Proposition 5.12. *Given a precompleted knowledge base* $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$*, and the union interpretation* $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ *from the individual models* $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ *with* $o \in \mathcal{O}$. *For each role* $R \in \mathcal{RN}$*, individual name* $o \in \mathcal{O}$*, and elements* $\{x, y\} \subseteq \Delta_o$*:*

$$(x, y) \in R^{\overline{\mathcal{I}}} \text{ and } y \neq o^{\overline{\mathcal{I}}} \text{ implies } (x, y) \in R^{\underline{\mathcal{I}_o}}$$

Proof. We prove this proposition by induction using the partial order $\preceq$ over the set of roles $\mathcal{RN}$. By the structure of Definition 5.10 we can start from transitive roles (no induction is necessary in this case), and then prove the property for non–transitive roles.

If R is transitive, we consider the set

$$\overline{R} = \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}} \cup \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R \right\},$$

and we show that for any integer n, $\{x, y\} \subseteq \Delta_o$, $(x, y) \in \overline{R}^n$, and $y \neq o^{\overline{\mathcal{I}}}$ implies $(x, y) \in R^{\underline{\mathcal{I}_o}}$. We show this by induction on n.

- If $n = 1$ then either

$$(x, y) \in \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}}, \text{ or}$$
$$(x, y) \in \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R \right\}.$$

The second case can be ruled out because $y \notin \mathcal{O}^{\mathcal{I}}$. In addition, the sets $R^{\mathcal{I}_{o'}}_{\underline{\ }}$ and $R^{\mathcal{I}_o}_{\underline{\ }}$ are disjoint for any $o' \in \mathcal{O}$ different from o, because the individual domains are disjoint by assumption; therefore $(x, y) \in R^{\mathcal{I}_o}_{\underline{\ }}$.

- As an inductive hypothesis we assume that the property holds for any number less or equal than n, and we show that it holds for $n + 1$.

  If $(x, y) \in \overline{R}^{n+1}$ there is an element z such that $(x, z) \in \overline{R}$ and $(z, y) \in \overline{R}^{n}$. Note that z must be different from $o^{\overline{\mathcal{I}}}$, because we assumed that $\mathcal{I}_o$ is a quasi transitive shrub (see Section 3.1). We can use the inductive hypothesis to conclude that both (x, z) and (z, y) are in $R^{\mathcal{I}_o}_{\underline{\ }}$; therefore $(x, y) \in R^{\mathcal{I}_o}$, because $R^{\mathcal{I}_o}_{\underline{\ }} \subseteq R^{\mathcal{I}_o}$ and $R^{\mathcal{I}_o}$ is transitive by assumption. In addition, $R^{\mathcal{I}_o}_{\underline{\ }} = R^{\mathcal{I}_o}$ because R cannot have any functional super–role (see Definition 5.9 and Definition 5.1). Therefore $(x, y) \in R^{\mathcal{I}_o}_{\underline{\ }}$.

If R is not transitive then we have four cases:

$$(x, y) \in \bigcup_{o \in \mathcal{O}} R^{\mathcal{I}_o}_{\underline{\ }},$$
$$(x, y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\},$$
$$(x, y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\},$$
$$(x, y) \in \bigcup_{S \preceq R, S \in \mathcal{TRN}} S^{\overline{\mathcal{I}}}.$$

The three initial cases are analogous to the one for transitive roles and $n = 1$: the second and third cases can be ruled out because $y \notin \mathcal{O}^{\mathcal{I}}$, while the first case satisfies the proposition because of the disjointness of the individual domains.

In the fourth case there is a transitive role S included in R (and different, because R is not transitive) s.t. $(x, y) \in S^{\overline{\mathcal{I}}}$. Since we already proved the property for transitive roles, then $(x, y) \in S^{\mathcal{I}_o}_{\underline{\ }}$; therefore $(x, y) \in R^{\mathcal{I}_o}_{\underline{\ }}$ by Proposition 5.11. □

The following proposition shows that pairs of root elements come from Abox assertions only.

Proposition 5.13. *Given a precompleted knowledge base* $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc}\rangle$, *and the union interpretation* $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ *from the individual models* $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ *with* $o \in \mathcal{O}$.

For any role $R \in \mathcal{RN}$ and elements x, y of $\overline{\Delta}$, if $(x, y) \in R^{\overline{\mathcal{I}}}$ and $y \in \mathcal{O}^{\overline{\mathcal{I}}}$ then:

$(x,y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\}$, *or*
$(x,y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\}$, *or*
$(x,y) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq S\right\})^+$ *for some transitive role $S \preceq R$.*

Proof. If R is transitive, let us consider the set

$$\overline{R} = \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}} \cup \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\}.$$

We show by induction on n that if $(x, y) \in \overline{R}^n$ and $y \in \mathcal{O}^{\overline{\mathcal{I}}}$ then

$$(x,y) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\})^+.$$

For $n = 1$ it is true because $(x, y) \notin \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}}$ since $\mathcal{I}_o$ is a quasi transitive tree interpretation for any o (see Section 3.3). Let us assume the property for any integer less or equal than n, and we consider $\overline{R}^{n+1}$. If $(x, y) \in \overline{R}^{n+1}$, then there is an element z s.t. $(x, z) \in \overline{R}$ and $(z, y) \in \overline{R}^n$. By the inductive hypothesis applied to $(z, y) \in \overline{R}^n$ we conclude that

$$(z,y) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\})^+;$$

therefore $z \in \mathcal{O}^{\overline{\mathcal{I}}}$. We can then use the inductive hypothesis with $(x, z) \in \overline{R}$ concluding that

$$(x,z) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\})^+$$

as well; therefore

$$(x,y) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\})^+.$$

If R is not transitive and $(x, y) \in R^{\overline{\mathcal{I}}}$, then one of the following cases must be satisfied:

$$
\begin{aligned}
&(x,y) \in \bigcup_{o\in\mathcal{O}} R^{\mathcal{I}_o}, \\
&(x,y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\}, \\
&(x,y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\}, \\
&(x,y) \in \bigcup_{S\preceq R, S\in\mathcal{TRN}} S^{\overline{\mathcal{I}}}.
\end{aligned}
$$

We can rule out the first case because of the structure of the individual interpretations. The second and third cases trivially satisfy the proposition, while the fourth case falls into the previously proved property for transitive roles. □

The second important property of role extensions in union interpretation is that there cannot be a connection between elements from different individual domains without a path passing through a root node (Proposition 5.14). This property ensures that all the restrictions applying to a non–root element can only be induced through the root of its own individual domain, instead of by being directly propagated from different individual domains.

Proposition 5.14. *Given a precompleted knowledge base $\Sigma_{pc} = \langle\mathcal{T}, \mathcal{A}_{pc}\rangle$, and the union interpretation $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ from the individual models $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ with $o \in \mathcal{O}$. For any role $R \in \mathcal{RN}$, different individual names a, b, and elements $x \in \Delta_a$, $y \in \Delta_b$, if $(x,y) \in R^{\overline{\mathcal{I}}}$ then $x = a^{\overline{\mathcal{I}}}$, and $y = b^{\overline{\mathcal{I}}}$ or there is a role $S \in \mathcal{TRN}$ such that $S \preceq R$ and $\left\{(x, b^{\overline{\mathcal{I}}}), (b^{\overline{\mathcal{I}}}, y)\right\} \subseteq S^{\overline{\mathcal{I}}}$.*

Proof. The proof follows the same structure of the previous Proposition 5.12.

If R is transitive let us consider the set

$$
\overline{R} = \bigcup_{o\in\mathcal{O}} R^{\mathcal{I}_o} \cup \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\}.
$$

We show by induction on n that that for any integer, if $x \in \Delta_a$, $y \in \Delta_b$ with $a \neq b$ and (x,y) is in $\overline{R}^n$, then $x = a^{\overline{\mathcal{I}}}$, and $y = b^{\overline{\mathcal{I}}}$ or $(x, b^{\overline{\mathcal{I}}})$, $(b^{\overline{\mathcal{I}}}, y)$ are in $R^{\overline{\mathcal{I}}}$ (this satisfies the condition, because R is transitive and $R \preceq R$).

- If $n = 1$ then either

  $$(x, y) \in \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}}, \text{ or}$$
  $$(x, y) \in \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R \right\}.$$

  The first case can be ruled out because of the disjointness of the individual domains, so $x \in \mathcal{O}^{\mathcal{I}}$. In addition, $a^{\mathcal{I}}$ is the only element in both Δ_a and $\mathcal{O}^{\mathcal{I}}$; therefore $x = a^{\mathcal{I}}$ (and analogously $y = b^{\overline{\mathcal{I}}}$).

- As an inductive hypothesis we assume that the property holds for any number less or equal than n, and we show that it holds for $n + 1$.

  If $(x, y) \in \overline{R}^{n+1}$ there is an element z such that $(x, z) \in \overline{R}$ and $(z, y) \in \overline{R}^n$.

  If $z \in \Delta_b$, we can use the inductive hypothesis with $n = 1$ for concluding that $x = a^{\overline{\mathcal{I}}}$. We then distinguish the two cases $z = b^{\overline{\mathcal{I}}}$ and $z \neq b^{\overline{\mathcal{I}}}$. In the first case the condition is trivially satisfied (note that $\overline{R}^n \subseteq R^{\overline{\mathcal{I}}}$ for any n). In the second case, $\left\{(x, b^{\overline{\mathcal{I}}}), (b^{\overline{\mathcal{I}}}, z)\right\} \subseteq R^{\overline{\mathcal{I}}}$ by the inductive hypothesis; in addition the transitivity of $R^{\overline{\mathcal{I}}}$ and the fact that $\left\{(b^{\overline{\mathcal{I}}}, z), (z, y)\right\} \subseteq R^{\overline{\mathcal{I}}}$, guarantees the satisfiability of the condition.

  If $z \notin \Delta_b$, then there is an individual o different from b s.t. $z \in \Delta_o$. We can use the induction hypothesis with o and b for $(z, y) \in \overline{R}^n$, concluding that $z = o^{\overline{\mathcal{I}}}$ and either $y = b^{\overline{\mathcal{I}}}$ or $\left\{(z, b^{\overline{\mathcal{I}}}), (b^{\overline{\mathcal{I}}}, y)\right\} \subseteq R^{\overline{\mathcal{I}}}$. Since $(x, o^{\overline{\mathcal{I}}}) \in R^{\overline{\mathcal{I}}}$, $x \in \mathcal{O}^{\overline{\mathcal{I}}}$ by Proposition 5.13; therefore $x = a^{\overline{\mathcal{I}}}$. If $y = b^{\overline{\mathcal{I}}}$ the condition is trivially satisfied, while if $\left\{(z, b^{\overline{\mathcal{I}}}), (b^{\overline{\mathcal{I}}}, y)\right\} \subseteq R^{\overline{\mathcal{I}}}$, $\left\{(x, b^{\overline{\mathcal{I}}}), (b^{\overline{\mathcal{I}}}, y)\right\} \subseteq R^{\overline{\mathcal{I}}}$ because $\left\{(x, z), (z, b^{\overline{\mathcal{I}}})\right\} \subseteq R^{\overline{\mathcal{I}}}$ and $R^{\overline{\mathcal{I}}}$ is transitive.

If R is not transitive then have four cases:

$$(x, y) \in \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}},$$
$$(x, y) \in \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R' \right\},$$
$$(x, y) \in \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R \right\},$$
$$(x, y) \in \bigcup_{S \preceq R, S \in \mathcal{TRN}} S^{\overline{\mathcal{I}}}.$$

The first case can be ruled out because of the disjointness of the individual domains.

In the second and third case $x = a^{\mathcal{I}}$ because $a^{\mathcal{I}}$ is the only element in both Δ_a and $\mathcal{O}^{\mathcal{I}}$ (and analogously $y = b^{\overline{\mathcal{I}}}$). Finally, for the fourth case there is a transitive role S included in R s.t. $(x, y) \in S^{\overline{\mathcal{I}}}$, and we can conclude that the proposition is true because we have already shown that it holds for transitive roles. $\square$

Before considering the interpretation of concepts, we must verify that the functional restrictions, and the role hierarchy are still satisfied in the union interpretation.

Proposition 5.15. *Given a precompleted knowledge base $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc} \rangle$, and the union interpretation $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ from the individual models $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ with $o \in \mathcal{O}$. If a role R is included in a functional role F (i.e. $R \preceq F$), then $\sharp \left\{ y | (x, y) \in R^{\overline{\mathcal{I}}} \right\} \leq 1$ for any element x of $\overline{\Delta}$.*

Proof. Note that there are no transitive roles included in R (and $R \notin \mathcal{TRN}$), since this is forbidden by the language definition. First of all we are going to show that if R is included in a functional role F then the definition of $R^{\overline{\mathcal{I}}}$ (see Definition 5.10) can be simplified by the formula:

$$R^{\overline{\mathcal{I}}} = \bigcup_{o \in \mathcal{O}} R^{\underline{\mathcal{I}_o}} \cup \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R' \right\}. \qquad (*)$$

Since R is included in a functional role, then it is not transitive and it does not include any transitive role; therefore the component $\bigcup_{S \preceq R, S \in \mathcal{TRN}} S^{\overline{\mathcal{I}}}$ is empty. We just need to show that

$$\left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R \right\} \subseteq \left\{ (a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R' \right\},$$

by proving that if $S \preceq R$, then $S \approx_a R'$ implies $R \approx_a R'$ for any role R' (see Definition 5.1). But this is actually the case because $S \preceq R$, and by assumption there is a functional role (F) including both S and R. The sequence of roles in Definition 5.1 can be extended for R by using S and F, and if there is a constraint $a{:}\exists S_0.C$ (or $\langle a, o' \rangle{:}S_0$) with $S_0 \preceq S$, this is good for R as well because $S_0 \preceq S \preceq R$.

Now that we proved Equation $(*)$ we are going to show that the functional restriction on R is satisfied. We proceed by contradiction, assuming that there are (x, y), (x, z) s.t. $y \neq z$ and $\{(x, y), (x, z)\} \subseteq R^{\overline{\mathcal{I}}}$. For doing this we use the alternative definition of $R^{\overline{\mathcal{I}}}$ shown in Equation $(*)$.

Note that it cannot be the case that $\{(x,y),(x,z)\} \subseteq \bigcup_{o\in\mathcal{O}} R^{\underline{\mathcal{I}_o}}$ because $R^{\underline{\mathcal{I}_o}}$ is functional for any o, and the individual domains are disjoint. If

$$\{(x,y),(x,z)\} \subseteq \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\}$$

then there is $o \in \mathcal{O}$ s.t. $x = o^{\overline{\mathcal{I}}}$, and $R \approx_o R$ because R is included in a functional role (see Definition 5.1); therefore, there is a contradiction between the assumption that $y \neq z$ and Proposition 5.2.

Therefore

$$\begin{aligned}(x,y) &\in \bigcup_{o\in\mathcal{O}} R^{\underline{\mathcal{I}_o}}, \text{ and}\\ (x,z) &\in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a,b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\}\end{aligned}$$

(or the converse, since using y or z is symmetric). Note that there must be two individual names o and o', s.t. $x = o^{\overline{\mathcal{I}}}$, $z = o'^{\overline{\mathcal{I}}}$, $\langle o,o'\rangle{:}S$ is in $\mathcal{A}_{pc}$, and $R \approx_a S$.

Since $(o^{\overline{\mathcal{I}}}, y) \in \bigcup_{o\in\mathcal{O}} R^{\underline{\mathcal{I}_o}}$, then $(o^{\overline{\mathcal{I}}}, y) \in R^{\underline{\mathcal{I}_o}}$ because of the disjointness of the individual domains. This is in contradiction with the fact that $R \approx_o S$ (see Definition 5.9). □

Now we show that the inclusion restrictions among role names are satisfied as well.

Proposition 5.16. *Given a precompleted knowledge base $\Sigma_{pc} = \langle \mathcal{T}, \mathcal{A}_{pc}\rangle$, and the union interpretation $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ from the individual models $\mathcal{I}_o = (\Delta_o, \cdot^{\mathcal{I}_o})$ with $o \in \mathcal{O}$. For two arbitrary roles R and S, if $S \preceq R$ then $S^{\overline{\mathcal{I}}} \subseteq R^{\overline{\mathcal{I}}}$.*

Proof. Let be $(x,y) \in S^{\overline{\mathcal{I}}}$, we need to show that $(x,y) \in R^{\overline{\mathcal{I}}}$ as well.

If $y \in \mathcal{O}^{\overline{\mathcal{I}}}$, then $(x,y) \in R^{\overline{\mathcal{I}}}$ because of Proposition 5.13 and the way the union interpretation is constructed (see Definition 5.10).

We assume that $y \notin \mathcal{O}^{\overline{\mathcal{I}}}$, and we distinguish the two cases in which $\{x,y\} \in \Delta_o$ for same individual o, and in which $x \in \Delta_o$, $y \in \Delta_{o'}$ with $o \neq o'$. In the first case $(x,y) \in S^{\underline{\mathcal{I}_o}}$ by Proposition 5.12, therefore $(x,y) \in R^{\underline{\mathcal{I}_o}}$ by Proposition 5.11. If $x \in \Delta_o$ and $y \in \Delta_{o'}$ with $o \neq o'$, then by Proposition 5.14 either $y = o'^{\overline{\mathcal{I}}}$ (which is in contradiction with the assumption that $y \notin \mathcal{O}^{\overline{\mathcal{I}}}$), or there is a transitive role $S' \preceq S$ s.t. $\left\{(x, o'^{\overline{\mathcal{I}}}), (o'^{\overline{\mathcal{I}}}, y)\right\} \subseteq S'^{\overline{\mathcal{I}}}$. If R is not transitive, then $S'^{\overline{\mathcal{I}}} \preceq R^{\overline{\mathcal{I}}}$ by construction (see Definition 5.10); therefore we assume that R is transitive. In addition, by using Proposition 5.12 we can conclude that $(o'^{\overline{\mathcal{I}}}, y) \in S'^{\underline{\mathcal{I}_{o'}}}$, therefore $(o'^{\overline{\mathcal{I}}}, y) \in R^{\underline{\mathcal{I}_{o'}}}$ by

Proposition 5.11. By Proposition 5.13

$$(x, o'^{\overline{\mathcal{I}}}) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq S'\right\})^+,$$

because S' does not have any functional super–role; therefore $(x, o'^{\overline{\mathcal{I}}}) \in R^{\overline{\mathcal{I}}}$ as well. Finally, we can conclude that $(x, y) \in R^{\overline{\mathcal{I}}}$ because $R^{\overline{\mathcal{I}}}$ is transitive by construction (see Definition 5.10). □

Now we are going to look at the concepts, by considering the properties of the union interpretation.

5.2.2 Interpretation of concepts

It is easy to prove that the interpretation of roles satisfies the role assertions in the KB (because of their relatively low expressivity); however concept assertions are more problematic. In fact, the presence of new pairs in the interpretation of roles can modify the extension of concept forming constructors (see Example 4.2).

This "non-monotonicity" problem does not involve any element of the domain, but it is localised to root elements. Roughly speaking, the reason for this lies on the fact that the interpretation of roles, restricted to non–root elements does not change (see Proposition 5.12). Proposition 5.17 shows that, if we restrict to non–root elements, the extension of arbitrary concept expressions w.r.t. the union interpretation is monotonic w.r.t the individual models.

Proposition 5.17. *For any individual name $a \in \mathcal{O}$ and concept expression D, $D^{\mathcal{I}_a} \setminus \left\{a^{\overline{\mathcal{I}}}\right\} \subseteq D^{\overline{\mathcal{I}}}$.*

Proof. The proposition is proved by induction on the structure of the concept expression D. The basic case of a concept name is true by construction of $\overline{\mathcal{I}}$ (see Definition 5.10). If it is true for the concept expressions C_1 and C_2, then for each more complex expression:

$\neg A$ Let $x \in \Delta_a \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$ be such that $x \notin A^{\overline{\mathcal{I}_a}}$, then $x \notin A^{\overline{\mathcal{I}}}$ by construction (the individual domains are disjoint), so $x \in (\neg A)^{\overline{\mathcal{I}}}$.

$(C_1 \sqcap C_2)$ By definition $(C_1 \sqcap C_2)^{\overline{\mathcal{I}}} = C_1^{\overline{\mathcal{I}}} \cap C_2^{\overline{\mathcal{I}}}$ and $C_1^{\mathcal{I}_a} \subseteq C_1^{\overline{\mathcal{I}}}$, $C_2^{\mathcal{I}_a} \subseteq C_2^{\overline{\mathcal{I}}}$ for the induction hypothesis; therefore $(C_1 \sqcap C_2)^{\mathcal{I}_a} \subseteq (C_1 \sqcap C_2)^{\overline{\mathcal{I}}}$.

$(C_1 \sqcup C_2)$ It is analogous to the $\sqcap$ case.

$(\exists R.C_1)$ If $x \in (\exists R.C_1)^{\mathcal{I}_a} \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$, then there exists $y \in \Delta_a$ such that $(x,y) \in R^{\mathcal{I}_a}$ and $y \in C_1^{\mathcal{I}_a}$. Note that $(x,y) \in R^{\underline{\mathcal{I}_a}}$, because only pairs whose first element is the root element are removed in the restricted individual interpretation function (see Definition 5.9). Therefore $(x,y) \in R^{\overline{\mathcal{I}}}$ (see Definition 5.10). In addition, $y \neq a^{\overline{\mathcal{I}}}$ because $\mathcal{I}_a$ is a q.t. tree, so $y \in C_1^{\overline{\mathcal{I}}}$ by the induction hypothesis; therefore $x \in (\exists R.C_1)^{\overline{\mathcal{I}}}$.

$(\forall R.C_1)$ Let be x such that $x \in (\forall R.C_1)^{\mathcal{I}_a} \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$, and $(x,y) \in R^{\overline{\mathcal{I}}}$. By using Proposition 5.13 and Proposition 5.12 we can conclude that (x,y) is in $R^{\underline{\mathcal{I}_a}}$ so $y \in C_1^{\mathcal{I}_a}$, and $y \neq a^{\overline{\mathcal{I}}}$. Therefore $y \in C_1^{\overline{\mathcal{I}}}$ for the induction hypothesis.

□

As shown in Example 4.2, this monotonicity property does not hold for arbitrary concept expressions applied to root elements. However, it is still valid for concept expressions which appear explicitly as assertions in the precompleted KB (Proposition 5.18). In fact, the concept in Example 4.2 does not appear in the KB as an assertion involving the individual a. Moreover, this restricted property of the union interpretation (together with the more general one applying to non–roots) is sufficient to prove that all the axioms and assertions in the precompleted knowledge base are satisfied by the union interpretation (Proposition 5.19).

Proposition 5.18. *For any individual name $a \in \mathcal{O}$, $a{:}D \in \mathcal{A}_{pc}$ implies $a^{\overline{\mathcal{I}}} \in D^{\overline{\mathcal{I}}}$.*

Proof. The Proposition is proved by induction on the structure of the concept expression D.

- The basic case is a concept name: If $a{:}A$ is in $\mathcal{A}_{pc}$ then $A \in \mathcal{L}(\Sigma_{pc}, a)$ so $a \in A^{\mathcal{I}_a}$; therefore $a^{\overline{\mathcal{I}}} \in A^{\overline{\mathcal{I}}}$ because $A^{\mathcal{I}_a} \subseteq A^{\overline{\mathcal{I}}}$.

- If it is true for the concept expressions C_1 and C_2, then for each more complex expression:

  $\neg A$ If $a{:}\neg A \in \mathcal{A}_{pc}$, then $\neg A \in \mathcal{L}(\Sigma_{pc}, a)$ so $a^{\overline{\mathcal{I}}} \notin A^{\mathcal{I}_a}$. From the disjointness of the individual domains $a^{\overline{\mathcal{I}}} \notin A^{\overline{\mathcal{I}}}$ (see Definition 5.10), therefore $a^{\overline{\mathcal{I}}} \in (\neg A)^{\overline{\mathcal{I}}}$

  $(C_1 \sqcap C_2)$ If $a{:}(C_1 \sqcap C_2) \in \mathcal{A}_{pc}$, then both $a{:}C_1$ and $a{:}C_2$ are in $\mathcal{A}_{pc}$ (Proposition 5.6). For the induction hypothesis $a^{\overline{\mathcal{I}}}$ is in both $C_1^{\overline{\mathcal{I}}}$ and $C_2^{\overline{\mathcal{I}}}$, so it is in $(C_1 \sqcap C_2)^{\overline{\mathcal{I}}}$ as well.

$(C_1 \sqcup C_2)$ If $a{:}(C_1 \sqcup C_2) \in \mathcal{A}_{pc}$, then either $a{:}C_1$ or $a{:}C_2$ is in $\mathcal{A}_{pc}$ (Proposition 5.6). If $a{:}C_1$ is in $\mathcal{A}_{pc}$, then for induction hypothesis $a^{\overline{\mathcal{I}}} \in C_1^{\overline{\mathcal{I}}}$; therefore $a^{\overline{\mathcal{I}}} \in (C_1 \sqcup C_2)^{\overline{\mathcal{I}}}$ as well. The same if $a{:}C_2$ is in $\mathcal{A}_{pc}$.

$(\exists R.C_1)$ If $a{:}(\exists R.C_1)$ is in $\mathcal{A}_{pc}$ then $(\exists R.C_1) \in \mathcal{L}(\Sigma_{pc}, a)$; therefore there is an element y in Δ_a s.t. $(a^{\overline{\mathcal{I}}}, y) \in R^{\mathcal{I}_a}$ and $y \in {C_1}^{\mathcal{I}_a}$. The problem is that $R^{\underline{\mathcal{I}_a}} \subseteq R^{\mathcal{I}_a}$, so the pair $(a^{\overline{\mathcal{I}}}, y)$ can be removed in the restricted individual interpretation function (see Definition 5.9).

If $(a^{\overline{\mathcal{I}}}, y) \in R^{\underline{\mathcal{I}_a}}$, then $(a^{\overline{\mathcal{I}}}, y) \in R^{\overline{\mathcal{I}}}$ as well, and since $y \in \Delta_a \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$ because of the tree–like structure of $\mathcal{I}_a$, then $y \in {C_1}^{\overline{\mathcal{I}}}$ by Proposition 5.17. This concludes that $a^{\overline{\mathcal{I}}} \in (\exists R.C_1)^{\overline{\mathcal{I}}}$.

If $(a^{\overline{\mathcal{I}}}, y) \notin R^{\underline{\mathcal{I}_a}}$, then there are R', R'' s.t. $R \preceq R'$, $\langle a, o' \rangle{:}R'' \in \mathcal{A}_{pc}$ and $R' \approx_a R''$. Since $R' \approx_a R''$ and $R \preceq R'$, there is a functional role F such that $R \preceq F$ and $R' \preceq F$; in addition, the constraint $a{:}(\exists R.C_1)$ is in $\mathcal{A}_{pc}$, so $R \approx_a R''$ (see Definition 5.1). This means that

$$(a^{\overline{\mathcal{I}}}, y) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R'' \in \mathcal{A}_{pc}, R \approx_a R''\right\}$$

therefore there is an individual b s.t. $y = b^{\overline{\mathcal{I}}}$ and $(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \in R^{\overline{\mathcal{I}}}$ (see Definition 5.10).

The only remaining thing is to show that $b^{\overline{\mathcal{I}}} \in {C_1}^{\overline{\mathcal{I}}}$. However, the constraint $b{:}C$ must be in $\mathcal{A}_{pc}$ because of the $\rightarrow_{\exists^1}$ rule (by Proposition 5.6); therefore we can use the inductive hypothesis to conclude that $b^{\overline{\mathcal{I}}} \in {C_1}^{\overline{\mathcal{I}}}$.

$(\forall R.C_1)$ Suppose that $(a^{\overline{\mathcal{I}}}, x) \in R^{\overline{\mathcal{I}}}$, then we distinguish three different cases according to the element x: x is one of the root elements (i.e. $x \in \mathcal{O}^{\overline{\mathcal{I}}}$), $x \in \Delta_a \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$, or there is $b \neq a$ s.t. $x \in \Delta_b \setminus \left\{b^{\overline{\mathcal{I}}}\right\}$.

– If $x = o^{\overline{\mathcal{I}}}$ then the pair $(a^{\overline{\mathcal{I}}}, x)$ must be introduced by means of Abox assertions (see Proposition 5.13). Let us consider the three different cases.

  $*$ If
  $$(a^{\overline{\mathcal{I}}}, x) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b \rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq R\right\}$$
  there is a constraint $\langle a, b \rangle{:}R'$ with $R' \preceq R$ s.t. $x = b^{\overline{\mathcal{I}}}$. The constraint $b{:}C_1$ must be in $\mathcal{A}_{pc}$ because of the $\rightarrow_{\forall}$ rule (by Proposition 5.6). Therefore we can use the inductive hypothesis for concluding that $x \in {C_1}^{\overline{\mathcal{I}}}$.

* If
$$(a^{\overline{\mathcal{I}}}, x) \in \left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R \approx_a R'\right\}$$
there is a constraint $\langle a, b\rangle{:}R'$ with $R \approx_a R'$ s.t. $x = b^{\overline{\mathcal{I}}}$. The constraint $b{:}C_1$ must be in $\mathcal{A}_{pc}$ because of the $\rightarrow_{\forall^1}$ rule (by Proposition 5.6). Therefore we can use the inductive hypothesis for concluding that $x \in {C_1}^{\overline{\mathcal{I}}}$.

* If
$$(a^{\overline{\mathcal{I}}}, x) \in (\left\{(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \mid \langle a, b\rangle{:}R' \in \mathcal{A}_{pc}, R' \preceq S\right\})^+$$
there is a set of constraints
$$\{\langle a, o_1\rangle{:}S_1, \langle o_1, o_2\rangle{:}S_2, \ldots, \langle o_{n-1}, o_n\rangle{:}S_n\} \subseteq \mathcal{A}_{pc}$$
and a transitive role S s.t. $S_i \preceq S \preceq R$ for all $i = 1, \ldots, n$, and $x = {o_n}^{\overline{\mathcal{I}}}$. By Proposition 5.6 each $i = 1, \ldots, n$ the contraints $o_i{:}(\forall S.C_1)$ and $o_i{:}C_1$ are in $\mathcal{A}_{pc}$ (because of the $\rightarrow_{\forall^+}$ rule and the $\rightarrow_{\forall}$). Therefore $o_n{:}C_1$ is $\mathcal{A}_{pc}$ and $x \in {C_1}^{\overline{\mathcal{I}}}$ by the inductive hypothesis.

– If $x \in \Delta_a \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$ then $(a^{\overline{\mathcal{I}}}, x) \in R^{\mathcal{I}_a}$, because both $a^{\overline{\mathcal{I}}}$ and x are in the same individual domain (see Definition 5.10). In addition, $(\forall R.C_1) \in \mathcal{L}(\Sigma_{pc}, a)$ so $x \in {C_1}^{\mathcal{I}_a}$ because $\mathcal{I}_a$ is a model for the label $\mathcal{L}(\Sigma_{pc}, a)$. Using Proposition 5.17 we can conclude the $x \in {C_1}^{\overline{\mathcal{I}}}$.

– If there is $b \neq a$ s.t. $x \in \Delta_b \setminus \left\{b^{\overline{\mathcal{I}}}\right\}$, then by Proposition 5.14 there is a transitive role S s.t. $S \preceq R$ and both $(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}})$, $(b^{\overline{\mathcal{I}}}, x)$ are in $S^{\overline{\mathcal{I}}}$. By using the very same arguments as the first case we can show that the constraint $b{:}(\forall S.C_1)$ is in $\mathcal{A}_{pc}$; therefore we can conclude that $x \in {C_1}^{\overline{\mathcal{I}}}$ by using the same arguments as in the previous case.

□

5.2.3 Satisfiability of a precompleted KB

It is clear that a model for the knowledge base is a model for every individual concept. Given the results of the previous sections, Propositions 5.19 shows that the union interpretation is a model for the precompleted knowledge base.

Proposition 5.19. *A clash-free $\mathcal{SH}f$ precompleted knowledge base $\Sigma_{pc} = \langle \mathcal{T}_{pc}, \mathcal{A}_{pc} \rangle$ is satisfiable if and only if for each individual name $o \in \mathcal{O}$ the concept expression $\prod \mathcal{L}(\Sigma_{pc}, o)$ (see Definition 5.8) is satisfiable with respect the terminology $\mathcal{T}_{pc}$.*

Proof. $\Rightarrow$ Let $\mathcal{I}$ be a model for Σ_{pc}, then for any individual name o and for any concept C in $\mathcal{L}(\Sigma_{pc}, o)$ the individual $o^{\mathcal{I}}$ is in the extension $C^{\mathcal{I}}$ (see Definition 5.8). Therefore $o^{\mathcal{I}} \in (\prod \mathcal{L}(\Sigma_{pc}, o))^{\mathcal{I}}$ so its interpretation is non–empty with respect to the terminology $\mathcal{T}_{pc}$.

$\Leftarrow$ If for every named individual o the concept expression $\prod \mathcal{L}(\Sigma_{pc}, o)$ is satisfiable with respect to the terminology $\mathcal{T}_{pc}$, there is a quasi transitive tree interpretation $\mathcal{I}_o$ which satisfies $\mathcal{T}_{pc}$ and having the root element in $(\prod \mathcal{L}(\Sigma_{pc}, o))^{\mathcal{I}_o}$ (see Section 3.4). Therefore the union interpretation $\overline{\mathcal{I}} = (\overline{\Delta}, \cdot^{\overline{\mathcal{I}}})$ can be defined as Definition 5.10. Verifying that interpretation $\overline{\mathcal{I}}$ satisfies every assertion in Σ_{pc} it can be proved that it is a model for Σ_{pc}.

- Let $a{:}C$ be a concept assertion in $\mathcal{A}_{pc}$, then by Proposition 5.18 $a^{\overline{\mathcal{I}}} \in C^{\overline{\mathcal{I}}}$.
- Let $\langle a, b\rangle{:}R$ be a role assertion in $\mathcal{A}_{pc}$, then $(a^{\overline{\mathcal{I}}}, b^{\overline{\mathcal{I}}}) \in R^{\overline{\mathcal{I}}}$ by construction of $R^{\overline{\mathcal{I}}}$ in Definition 5.10.
- Let $\top \sqsubseteq C$ be an axiom in $\mathcal{T}_{pc}$, then $\overline{\mathcal{I}}$ satisfies the axiom if and only if every element in the domain $\overline{\Delta}$ is in the extension of C with respect to $\overline{\mathcal{I}}$. Let x be an element of $\overline{\Delta}$, then there exists $a \in \mathcal{O}$ such that $x \in \Delta_a$, and $x \in C^{\mathcal{I}_a}$ because the model $\mathcal{I}_a$ satisfies $\mathcal{T}_{pc}$. There are two cases: $x = a^{\overline{\mathcal{I}}}$ or $x \in \Delta_a \setminus \left\{a^{\overline{\mathcal{I}}}\right\}$
  - If $x = a^{\overline{\mathcal{I}}}$ then $x \in C^{\overline{\mathcal{I}}}$ because of Proposition 5.18 ($a{:}C$ is in the precompleted ABox $\mathcal{A}_{pc}$ by Proposition 5.6).
  - If $x \neq a^{\overline{\mathcal{I}}}$, then $x \in C^{\overline{\mathcal{I}}}$ by Proposition 5.17.
- If $R \in \mathcal{TRN}$ then $R^{\overline{\mathcal{I}}}$ is transitive by construction (see Definition 5.10).
- If the role R is functional (i.e. $R \in \mathcal{FRN}$, or there is a role $F \in \mathcal{FRN}$ such that $R \subseteq F$), then its functional restriction is satisfied by Proposition 5.15.
- If $S \sqsubseteq R$ is an axiom in $\mathcal{T}_{pc}$, then $S \preceq R$ and $S^{\overline{\mathcal{I}}} \subseteq R^{\overline{\mathcal{I}}}$ by Proposition 5.16.

□

5.3 Knowledge base satisfiability

As summarised in Theorem 5.20, the satisfiability of a general knowledge base can be verified by checking the satisfiability of its precompletions via a terminological reasoner.

Theorem 5.20. *The satisfiability checking problem for an $\mathcal{SH}f$ knowledge base can be reduced to concept satisfiability with respect to a terminology.*

Proof. A knowledge base Σ is satisfiable if and only if one of the precompletions $\Sigma_1, \ldots, \Sigma_n$ generated by the algorithm in Definition 5.3 is satisfiable (Proposition 5.5). Moreover, the problem of checking the satisfiability of each one of the precompletions can be solved by a concept satisfiability checking for each named individual in the knowledge base (Proposition 5.19). $\square$

This means that we now have a KB satisfiability algorithm: the precompletion rules of Figure 5.1 provide an algorithmic way to enumerate the precompletions of a knowledge base, while the satisfiability of each precompletion is verified using an external terminological system such as FaCT (see Horrocks [1998]).

Chapter 6

Querying Aboxes

In earlier chapters we have shown how to verify the satisfiability of a $\mathcal{SH}f$ KB (Chapter 5) and how to answer to simple DL queries (Section 2.2). This chapter introduces the reader to a novel query answering technique for DL KBs. This technique provides correct and complete answers for disjunctions of conjunctive queries. The aim of this chapter is to introduce the ideas underlying this technique for the simpler DL $\mathcal{ALC}$. The full formal discussion of the extension to the language $\mathcal{SH}f$ is in the next Chapter.

6.1 Introduction

Typically, the only kind of query DL systems support is instantiation (is an individual i an instance of a concept C?), realisation (what are the most specific concepts i is an instance of?) and retrieval (which individuals are instances of C?). The reason for this weakness is that, in these expressive logics, all reasoning tasks are reduced to that of determining KB satisfiability (consistency). In particular, instantiation is reduced to KB (un)satisfiability by transforming the query into a negated assertion; however, this technique cannot be used (directly) for queries involving roles because these logics do not support role negation.

For example, it can be inferred that `John` is an instance of `Animal` if and only if the KB is not satisfiable when an axiom is added to the Abox asserting that `John` is not an instance of `Animal` (i.e., that `John` is an instance of the negation of `Animal`). Realisation and retrieval can, in turn, be achieved through repeated application of instantiation tests. However, this technique cannot be used (directly) to infer from the above axioms that the pair $\langle$`John`, `Bill`$\rangle$ is an instance of the transitive role `Brother`, because these logics do not support role negation, i.e., it is not possible to assert that

$\langle\texttt{John}, \texttt{Bill}\rangle$ is an instance of the negation of $\texttt{Brother}$.

In this chapter we introduce a technique for answering such queries using a more sophisticated reduction to KB satisfiability. We then show how this technique can be extended to determine if an arbitrary tuple of individuals (i.e., not just a singleton or pair) satisfies a disjunction of conjunctions of concept and role membership assertions that can contain both constants (i.e., individual names) and variables. This provides a powerful query language, similar to the conjunctive queries typically supported by relational databases,[1] that allows complex Abox structures (e.g., cyclical structures) to be retrieved by using variables to enforce co-reference (see Chandra and Merlin [1977]). For example, the query

$$\{\langle x, y\rangle \mid \exists z(\langle z, \texttt{Bill}\rangle{:}\texttt{Parent} \wedge \langle z, x\rangle{:}\texttt{Parent} \wedge \langle z, y\rangle{:}\texttt{Parent} \wedge \langle x, y\rangle{:}\texttt{Hates})\} \quad (6.1)$$

would retrieve all the pairs of hostile siblings in $\texttt{Bill}$'s family.[2]

In our work we focus on answering boolean queries, i.e., determining if a query is true with respect to a KB. Retrieval can be easily (although inefficiently) turned into a set of boolean queries for all candidate tuples. Note that in available systems, the retrieval problem is similarly reduced to instantiation.[3] It is important to stress the fact that, given the expressivity of DLs, query answering cannot simply be reduced to model checking as in the database framework (we illustrate this point in Example 6.1 below). This is because KBs may contain nondeterminism and/or incompleteness, making it infeasible to use an approach based on minimal models. In fact, query answering in the DL setting requires the same reasoning machinery as logical derivation.

Example 6.1
For using model checking technique for query answering we must be able to associate a "preferred" model to a given KBs, and this is quite difficult for arbitrary DL KBs. Let us consider for example a simple KB containing the single axiom $\texttt{Elephant} \sqsubseteq \neg\texttt{Mouse}$, stating that elephants and mice are disjoint, and the Abox assertion $\texttt{hathi}{:}(\texttt{Elephant} \sqcup \texttt{Mouse})$.

We can identify two minimal (w.r.t. inclusion) interpretations satisfying the KB:

[1]The work is inspired by the use of Abox reasoning to decide conjunctive query containment (see Horrocks et al. [1999a], Calvanese et al. [1998a]).

[2]Note that a sound and complete KB satisfiability algorithm will guarantee sound and complete query answers.

[3]Apart from not very expressive DL languages (see for example Rousset [1999]).

in the first the element mapped from the individual `hathi` is in the extension of the concept `Elephant`, while in the second it is in the extension of the concept `Mouse`. Which of the two interpretations can be considered the "preferred" one? The point is that there is not any general mechanism for choosing one, even with this trivial KB.

An important advantage with the technique presented here is that it is quite generic, and can be used with any DL providing general inclusion axioms where instantiation can be reduced to KB satisfiability. It could therefore be used to significantly increase the utility of Abox reasoning in a wide range of existing (and future) DL implementations.

6.2 Conjunctive Queries in DL

In this chapter we will focus on conjunctive queries only (i.e. queries containing only conjunctions of terms), and the DL $\mathcal{ALC}$. This choice has been made for simplifying the description of the underlying ideas. In the next chapter we are going to formally present the technique for a wider range of queries and the more expressive DL $\mathcal{SH}f$.

A key feature of conjunctive queries is that they may contain variables, and we will assume the existence of a set of variables that is disjoint from the set of individual names. A conjunctive query is a formula $\exists x_1 \ldots x_k (q_1 \wedge \ldots \wedge q_n)$, where $q_1, \ldots, q_n$ are concept or role query terms. Concept and role terms are syntactically equal to Abox assertions of the form $x{:}C$ or $\langle x, y\rangle{:}R$ where C is a concept, R is a role and x, y are either individual names or variables.[4] A query formula generally contains free variables which correspond to the tuple the query is meant to retrieve; these variables are also called *distinguished*. Queries without free variables are said to be *boolean*, and the answer to these queries is not a set of tuples but a true or false value.

In this chapter we are going to introduce the semantics of query answering in a intuitive way; a formal definition will be presented in the next chapter. The semantics relies on the the definition of interpretations for a DL: an interpretation $\mathcal{I}$ satisfies a query Q iff the interpretation function can be extended to the variables in Q in such a way that $\mathcal{I}$ satisfies every term in Q. In addition, we require that free variables in Q are interpreted as element corresponding to individual names; this restriction ensure that the set corresponding to a query answer contains only individual names. Using this definition we can introduce the knowledge base into the framework: a query Q is true

[4]We decided to avoid the more common notation $C(x)$ and $R(x, y)$ because it is confusing when C and R are complex concept and role expressions.

w.r.t. Σ (written $\Sigma \models Q$) iff every interpretation that satisfies Σ also satisfies Q. The answer to a non–boolean query will be the set of tuples of individual names that, once substituted to the distinguished variables, make the query true w.r.t. the knowledge base. For example, the query

$$Q \equiv \langle \texttt{Bill}, y\rangle{:}\texttt{Parent} \land \langle y, z\rangle{:}\texttt{Parent} \land z{:}\texttt{Male} \tag{6.2}$$

is true w.r.t. a KB Σ iff it can be inferred from Σ that `Bill` has a grandson. Note that query truth value and the idea of logical consequence are strictly related. In fact, a boolean query is true w.r.t. a KB iff it is a logical consequence of the KB.

In the following, we will only consider how to answer boolean queries. Retrieving sets of tuples can be achieved by repeated application of boolean queries with different tuples of individual names substituted for variables. For example, the answer to the retrieval query $\{\langle y, z\rangle \mid Q\}$ w.r.t. a KB Σ is a set of tuples $\langle a, b\rangle$, where a, b are individual names occurring in Σ, such that $\Sigma \models Q'$ for the boolean query Q' obtained by substituting a, b for y, z in Q. Of course the naive evaluation of such a retrieval could be prohibitively expensive, but would clearly be amenable to optimisation. For example, let us consider the query (6.1) of the previous section. In many DLs the expressivity of the Abox for roles is very limited: in $\mathcal{ALC}$, for example, a KB implies a query term like $\langle z, \texttt{Bill}\rangle{:}\texttt{Parent}$, where the second argument is an individual name, only if there is an explicit assertion of this form in the Abox. Therefore we may use role assertions in the Abox to reduce the number of candidates among the individual names.

We will show how to answer boolean queries in two steps. Firstly, we will consider conjunctions of terms containing only individual names appearing in the KB; secondly, we will show how this basic technique can be extended to deal with variables. For simplifying the notation, we represent the knowledge bases as a single set containing both the Tbox and Abox assertions (instead of a pair as in Chapter 2).

6.2.1 Queries with multiple terms

In this section we will consider queries expressed as a conjunction of concept and role terms built using only names appearing in the KB (i.e. without variables); e.g., $\texttt{Tom}{:}\texttt{Student}$ or $\langle \texttt{Tom}, \texttt{CS710}\rangle{:}\texttt{Enrolled}$.

As we have already seen, logical consequence can easily be reduced to a KB satisfiability problem if the query contains only a single concept term (this is the standard instantiation problem). For example, $\texttt{Tom}{:}\texttt{Person}$ is a logical consequence of the KB $\{\texttt{Student} \sqsubseteq \texttt{Person}, \texttt{Tom}{:}\texttt{Student}\}$ iff the KB

$$\{\texttt{Student} \sqsubseteq \texttt{Person}, \texttt{Tom}{:}\texttt{Student}, \texttt{Tom}{:}\neg\texttt{Person}\}$$

is not satisfiable. This can be generalised to queries containing conjunctions of concept terms simply by transforming the query test into a set of (un)satisfiability problems: a conjunction $a_1{:}C_1 \wedge \ldots \wedge a_n{:}C_n$ is a logical consequence of a KB iff each $a_i{:}C_i$ is a logical consequence of the KB.

However, this simple approach cannot be used in our case since a query may also contain role terms. Instead, we will show how simple transformations can be used to convert every role term into a concept term. We call this procedure *rolling up* a query.

The rationale behind rolling up can easily be understood by imagining the availability of the DL `one-of` operator, which allows the construction of a concept containing only a single named individual (see Schaerf [1994]). The standard notation for such a concept is $\{\ a\ \}$, where a is an individual name, and the semantics is given by the equation $\{\ a\ \}^{\mathcal{I}} = \left\{a^{\mathcal{I}}\right\}$. For example, the expression $\{\ \texttt{Bill}\ \}$ represents a concept containing only the individual `Bill` (i.e., $\{\ \texttt{Bill}\ \}^{\mathcal{I}} = \left\{\texttt{Bill}^{\mathcal{I}}\right\}$).

Using the `one-of` constructor, the role term $\langle\texttt{John}, \texttt{Bill}\rangle{:}\texttt{Brother}$ can be transformed into the equivalent concept term $\texttt{John}{:}(\exists\texttt{Brother}.\{\ \texttt{Bill}\ \})$. Furthermore, other concept terms asserting additional facts about the individual being rolled up (`Bill` in this case) can be absorbed into the rolled up concept term. For example, the conjunction

$$\langle\texttt{John}, \texttt{Sally}\rangle{:}\texttt{Parent} \wedge \texttt{Sally}{:}\texttt{Female} \wedge \texttt{Sally}{:}\texttt{PhD}$$

can be transformed into $\texttt{John}{:}\exists\texttt{Parent}.(\{\ \texttt{Sally}\ \} \sqcap \texttt{Female} \sqcap \texttt{PhD})$. The absorption transformation is not strictly necessary for queries without variables, but it serves to reduce the number of satisfiability tests needed to answer the query (by reducing the number of conjuncts), and it will be required with queries containing variables. By applying rolling up to each role term, an arbitrary query can be reduced to an equivalent one which contains only concept terms, and which can be answered using a set of satisfiability tests as described above.

However, the logic we are using does not include the `one-of` constructor, and most of the state of the art DL systems do not provide the constructor (with the exception of the latest version of the DLP reasoner, see Patel-Schneider [1998]). Fortunately, we do not need the full expressivity of `one-of`, and in our case it can be "simulated" without the need of adding new axioms (see Section 4.1.2). The technique used is to substitute each occurrence of `one-of` with a new concept name not appearing in the knowledge base. These new concept names must be different for each individual in the query, and are called the *representative* concepts of the individuals (written P_a, where a is the individual name). In addition, assertions which ensure that each individual is an instance of its representative concept must be added to the knowledge base (e.g., $\texttt{Bill}{:}P_{\texttt{Bill}}$). In general, a representative concept cannot be used in place of `one-of` because it can have instances other than the individual which it represents (i.e., $P_a{}^{\mathcal{I}} \supseteq \{a^{\mathcal{I}}\}$). However, representative concepts can be used instead of `one-of` in our reduced setting. In particular, the conjunction $\langle a, b\rangle{:}R \wedge b{:}C$ is a logical consequence of a given knowledge base if and only if $a{:}\exists R.(P_b \sqcap C)$ is a logical consequence of the very same knowledge base augmented by the assertion $b{:}P_b$. Note that the trick of using a concept as representative for an individual has been used in a few other cases in the literature (see De Giacomo and Lenzerini [1996], Borgida and Patel-Schneider [1994], Era and Donini [1992]).

6.2.2 Queries with variables

In this section we show how variables can be introduced in this framework by using a more complex rolling up procedure in order to obtain a similar reduction to the KB (un)satisfiability problem. Variables can be used exactly as individual names, but their meaning is as "place-holders" for unknown elements of the domain. Because variables may be interpreted as any element of the domain, they cannot simply be considered as individual names to which the unique name assumption does not apply; nor can they be treated as referring only to named individuals, giving the possibility of nondeterministically substituting them with names in the KB. In fact the query

$$\langle \texttt{Bill}, y\rangle{:}\texttt{Parent} \wedge \langle y, z\rangle{:}\texttt{Parent} \wedge z{:}\texttt{Male} \tag{6.3}$$

is true w.r.t. both the KBs

$$\left\{\begin{array}{l}\langle \texttt{Bill}, \texttt{Mary}\rangle{:}\texttt{Parent},\\ \langle \texttt{Mary}, \texttt{Tom}\rangle{:}\texttt{Parent},\\ \texttt{Tom}{:}\texttt{Male}\end{array}\right\} \quad \text{and} \quad \{\texttt{Bill}{:}\exists\texttt{Parent}.(\exists\texttt{Parent}.\texttt{Male})\},$$

but for the first KB the variables can be substituted by the individual names `Mary` and `Tom`, while in the second case the variables may need to be interpreted as elements of the domain that are not the interpretations of any named individuals.

Answering queries containing variables involves a more sophisticated rolling up technique. For example, let us consider the terms $\langle y, z\rangle{:}\texttt{Parent}$ and $z{:}\texttt{Male}$ of query (6.3). If z were an individual name, then the terms could be rolled up as $y{:}\exists\texttt{Parent}.(P_z \sqcap \texttt{Male})$, but this is not an equivalent query when z is a variable name because z can be interpreted as any element of the domain, not just an element of ${P_z}^{\mathcal{I}}$. However, since in this case z is no longer referred to in any other place in the query, there is no other constraint on how an interpretation can be extended w.r.t. z, so the concept $\top$ (whose interpretation is always the whole domain) can be used instead of P_z. The resulting concept term is $y{:}\exists\texttt{Parent}.(\top \sqcap \texttt{Male})$, which can be simplified to $y{:}\exists\texttt{Parent}.\texttt{Male}$. The same procedure can now be applied to y, thereby reducing query (6.3) to the single concept term $\texttt{Bill}{:}\exists\texttt{Parent}.(\exists\texttt{Parent}.\texttt{Male})$.

In order to show how this procedure can be more generally applied, it will be useful to consider the directed graph induced by the query, i.e., a graph in which there is a node x for each individual or variable x in the query, and an edge R from node x to node y for each role term $\langle x, y\rangle{:}R$ in the query. For instance, Query (6.3) corresponds to the graph[5]

$$\begin{tikzcd} \texttt{Bill} \arrow[r, "\texttt{Parent}"] & y \arrow[r, "\texttt{Parent}"] & z^{\texttt{Male}} \end{tikzcd}$$

It is easy to see that the rolling up procedure can be used to eliminate variables from any tree-shaped part of a query by starting at the leaves and working back towards the root (this is similar to the notion of descriptive support described in Rousset [1999]). The fact that rolling up should start from leaves is essential for correctness: for example, rolling up query (6.3) in the reverse order would lead to the non-equivalent $\texttt{Bill}{:}\exists\texttt{Parent}.\top \wedge y{:}\exists\texttt{Parent}.\top \wedge z{:}\texttt{Male}$.

However, this simple procedure cannot be applied to parts of the query that contain cycles, or where more than one edge enters a node corresponding to a variable (i.e., with terms like $\langle x, z\rangle{:}R \wedge \langle y, z\rangle{:}S$).

Let us consider the case where a variable is involved in a cycle, e.g., the simple

[5]More details are provided in the next chapter.

query

$$\langle x, y\rangle{:}\texttt{Path} \wedge \langle y, z\rangle{:}\texttt{Path} \wedge \langle z, x\rangle{:}\texttt{Path} \tag{6.4}$$

$$\begin{tikzcd}
& y \arrow[dr, "\texttt{Path}"] & \\
x \arrow[ur, "\texttt{Path}"] & & z \arrow[ll, "\texttt{Path}"]
\end{tikzcd}$$

which tests the KB for the presence of a particular type of loop involving the role `Path`. Rolling up one of the terms does not help, because the resulting query

$$\langle x, y\rangle{:}\texttt{Path} \wedge \langle y, z\rangle{:}\texttt{Path} \wedge z{:}\exists\texttt{Path}.P_x$$

$$\begin{tikzcd}
& y \arrow[dr, "\texttt{Path}"] & & \\
x \arrow[ur, "\texttt{Path}"] & & z \arrow[r, phantom, "\exists\texttt{Path}.P_x"] &
\end{tikzcd}$$

still contains another reference to the variable x, and replacing P_x with $\top$ would result in a non-equivalent query that no longer contained a cycle. Moreover, it is obvious that there is no way to roll up the query in order to obtain a single occurrence of any of the three variables.

This problem can be solved by exploiting the tree model property of the logic. Given this property, we know that Tbox assertions alone cannot constrain all models to be cyclical (if there is a model, then there is a tree model), so any cycle that might satisfy a cyclical query must be explicitly asserted in the Abox. Moreover, given the restricted expressivity of role assertions (i.e., that they apply only to atomic role names), cycles enforced in every interpretation must be composed only of elements interpreting individual names occurring in the knowledge base. Therefore, before applying the rolling up procedure, a variable occurring in a cycle should be substituted by the individual names in the KB. This procedure generates a number of alternative queries corresponding to each individual in the KB. In the next chapter we will see how they are handled. The intuition behind this property can be understood by considering that, given an arbitrary interpretation satisfying the cycle only with elements not corresponding to individual names, a new interpretation can be build where the cycle is split by duplicating one or more of the involved elements. This new interpretation can be defined in such a way that it still satisfies the KB, but no longer contains the required cycle. This duplication can be performed only if the elements are not "fixed" individual names and by assertions in the Abox. For example, if in the query (6.4) the variable x is substituted by the individual name a, then it can be transformed into the query

$$\langle a, y\rangle{:}\texttt{Path} \wedge \langle y, z\rangle{:}\texttt{Path} \wedge z{:}(\exists\texttt{Path}.P_a),$$

which no longer contains a cycle composed only of variables. Consequently, it can be rolled up into the single concept term

$$a{:}\exists\texttt{Path}.(\exists\texttt{Path}.(\exists\texttt{Path}.P_a))$$

where the concept P_a is used to close the cycle.

For DLs which can enforce more restrictions on role structure (for example the transitivity or hierarchy in $\mathcal{SH}f$) the simple formulation of the tree model property is not longer valid and a more involved definition must be adopted (see Chapter 3). This fact complicates the basic algorithm we are sketching in this chapter as we show in Chapter 7.

A similar argument can be used when variables appear as the second argument of more than one role term, e.g., the variable z in the query $\langle x, z\rangle{:}R \wedge \langle y, z\rangle{:}S$. Such variables can also be dealt with by nondeterministically substituting them with individual names occurring in the Abox.

We have seen how role terms containing variables can be rolled up into concept terms, but these may still be of the form $x{:}C$, where x is a variable. For example, the query $\langle x, y\rangle{:}\texttt{Parent}$, where x and y are variables, can only be reduced to the single term $x{:}\exists\texttt{Parent}.\top$. In this case we need to verify that the interpretation of the concept $\exists\texttt{Parent}.\top$ is nonempty in every interpretation that satisfies the KB. In general, the interpretation of a concept C is nonempty in every interpretation that satisfies the KB Σ iff adding the assertion $\top \sqsubseteq \neg C$ to Σ makes it unsatisfiable.

Summarising, the procedure for answering an arbitrary boolean conjunctive query is divided into two phases. Firstly, the role terms are eliminated by repeatedly applying the following rules: (i) if the graph induced by the query contains a leaf node y, then the role term $\langle x, y\rangle{:}R$ is rolled up, and the edge $\langle x, y\rangle$ is removed from the graph; (ii) otherwise, if the graph contains a node y with multiple incoming edges, then all role terms $\langle x, y\rangle{:}R$ are rolled up,[6] and the corresponding edges are removed from the graph;

[6] If y is a variable, then it is first replaced with an individual name chosen nondeterministically from those occurring in the the KB.

(iii) if the graph still contains edges but no leaf nodes and no confluent nodes, then it must contain a cycle. In this case a node y in a cycle is chosen (preferably an individual as this reduces nondeterminism) and rolled up as in case (ii) above. Secondly, the query (which now contains only concept terms) evaluates to true iff there is at least one nondeterministic replacement of variables with individual names such that every term is a logical consequence of the KB.

6.2.3 Extending the framework

In the previous sections we sketched the very basic idea of the query answering algorithm. This intuitive procedure cannot be applied straight away to the $\mathcal{SH}f$ logic we are considering. In fact, role hierarchy and transitivity complicate the structure of interpretations (see Chapter 3). In addition, the presence of potential cycles introduces implicit disjunctions which we have not yet shown how to handle.

We need to add a further note about the `one-of` constructor (or the so called nominals in the Modal Logic community). From Section 6.2.1 the reader may have drawn the erroneous conclusion that this constructor can be used to represent variables as well as individual names. This is false, and providing a logic with `one-of` would not eliminate the creation of alternative queries in presence of cycles as described in Section 6.2.2. The reason for this is that, even though both variables and `one-of` represent a single element in the interpretation, the name of the individual in the `one-of` is more restrictive than the name of a variable.

Let us consider for example the query $\exists x, y(\langle x, y\rangle{:}R \wedge \langle x, y\rangle{:}S)$. We may be tempted to use the `one-of` constructor for representing the joining variable y by introducing a new individual name o_y and "rolling up" as in Section 6.2.1, giving the new query $\exists x(x{:}(\exists R.\{\ o_y\ \}) \wedge x{:}(\exists S.\{\ o_y\ \}))$. The semantics of the new query is completely different from the original one, because it is satisfied by an interpretation only if the interpretation function maps o_y to the appropriate element.[7] On the other hand, if the actual name o_y does not appear in the knowledge base, the element mapped from o_y by an interpretation function is arbitrary. Therefore, there is always at least an interpretation where o_y is mapped to a "wrong" element. This means that the new query would be never satisfied, even if the original one would be.[8]

[7] I.e. if the interpretation is $\mathcal{I}$, satisfying the condition that $(i, o_y^{\mathcal{I}}) \in R^{\mathcal{I}}$ and $(i, o_y^{\mathcal{I}}) \in S^{\mathcal{I}}$ for some $i \in \Delta^{\mathcal{I}}$.

[8] The query must be satisfied in every interpretation satisfying the KB.

6.3 Relation with other works

In the DL community soundness and completeness of reasoning services are considered essential; therefore we consider only approaches which satisfy this requirement. This assumption excludes for example the work done with LOOM which provides a first order query language with an incomplete algorithm (see MacGregor and Brill [1992]).

As we mentioned early on, our work on conjunctive query answering has been inspired by the application of Abox reasoning for solving the problem of query containment under constraints (see Horrocks et al. [2000a] and Calvanese et al. [1998a]). The similarity between the two problems comes from the fact that the query containment problem can be reformulated into a query answering problem. The trick, well known in the database community (see Chandra and Merlin [1977]), consists in transforming the less general query into a database by considering the variables as constants. If querying this simple database with the more general query provides an answer, then the inclusion is verified. This simple technique can be generalised to the case where the valid databases are restricted by a DL based constraint language (see Calvanese et al. [1998a]). The query containment problem is reduced to query answering in the DL setting where the constraints are transformed into the terminology.

The relation between the two problems is also exploited in Calvanese et al. [2000]. In their framework the query answering problem is reduced to satisfiability of Converse PDL formulae (see Chapter 4). Their technique is very similar to the one we are presenting, in fact we have been inspired by their work, particularly the idea of representing a query as a graph and using a "rolling-up" technique.

The difference lies on the underlying DL they use and the scope of their work. They rely heavily on Converse PDL logic for the transformation, therefore their approach is only applicable to systems providing a Converse PDL reasoner. In contrast, we want to present a general technique which is applicable to a wide range of DL systems with a minimum effort. For this reason, for example, we develop a mechanism for simulating the inverse role constructor which is provided by Converse PDL but not implemented in most of the available DL reasoners.

Even if Converse PDL is, for most aspects, more expressive than the logic $\mathcal{SH}f$ we are focusing on, the presence of transitivity and functional restrictions as well as role hierarchy in $\mathcal{SH}f$ makes the structure of its interpretations more convoluted (see Chapter 3). This complexity is reflected on the query answering algorithm, as shown in the next chapter.

The problem of enhancing the query language for Abox KBs has also been attacked in the context of the system CARIN (see Levy and Rousset [1996a]). This DL based system integrates a DL with a query language which is essentially (recursive) Datalog. Their combination provides a very powerful query language which subsumes conjunctive queries;[9] however, its applicability is limited by the fact that combining recursive Datalog with an expressive DL leads to undecidability (see Levy and Rousset [1996b]). In addition, the technique for query answering uses an ad hoc algorithm which is not easily portable across different DL reasoners.

On the same track as the CARIN approach, a proposal has been made to use backward-chaining for evaluating conjunctive queries based on query expansion (see Rousset [1999]). The problem with this approach is the extreme restriction on the expressiveness of the DL language ($\mathcal{ALN}$ with empty terminologies). The advantage of this technique is that the query answer is built bottom-up, leading to much better performance. The downside is that the the procedure relies on the fact that the underlying DL has the property of having some sort of "minimal model" against which the query can be evaluated. As soon that the DL expressivity is enriched by constructors enabling the representation of disjunctive information the underpinning idea behind the technique cannot no longer be applied.

In the next chapter we give a formal description of the technique outlined in this chapter. Formal proofs are provided for supporting the correctness and completeness of the described technique.

[9]Using non–recursive Datalog yields to a language which is essentially equivalent to the one presented in Calvanese et al. [2000].

Chapter 7

Answering conjunctive queries in $\mathcal{SH}f$

7.1 Query language

The query language we consider is an adaptation of the basic *conjunctive query* language as defined in the database setting (see Abiteboul et al. [1995], Chandra and Merlin [1977]). The main difference in the DL case lies in the fact that in DL there are only relations of arity one (concepts) or two (roles). Individual names can be viewed as the constants in the database case.

In the following we adopt the same notation as the one for the *conjunctive calculus* as described in [Abiteboul et al., 1995, section 4.2].

7.1.1 Syntax

Definition 7.1. Let $\mathcal{CL}$ be the set of valid concept expressions build over a set $\mathcal{CN}$ of atomic concept names, and a set $\mathcal{RN}$ of role names. We assume a set $\mathcal{O}$ of individual names, and a distinct set $\mathcal{V}$ of variables. The set $\mathcal{QL}(\mathcal{CL})$ of conjunctive formulae over $\mathcal{CL}$ is the set of all formulae defined by the following abstract syntax:

$$\begin{aligned} \varphi \leftarrow\ & x{:}C \mid \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n \mid x = o \mid x = y \mid \\ & \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \exists x \varphi \end{aligned}$$

where C is a concept expression in $\mathcal{CL}$, $R_1, \ldots, R_n$ are role names in $\mathcal{RN}$, o is an individual name in $\mathcal{O}$), and x, y are variable names in $\mathcal{V}$.

The atomic formulae of the form $x{:}C$, $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$, $x = o$, and $x = y$ are called *terms*. We distinguish the terms into concept terms ($x{:}C$), role terms

($\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$), and equality terms.

Given a formula of $\mathcal{QL}(\mathcal{CL})$ we use the natural definition of *free* variables as those not bound by an existential quantifier. We indicate the set of free variables of a given formula φ as $\mathrm{free}(\varphi)$. For the sake of simplicity we introduce the notation $\exists x_1 x_2 \ldots x_n \varphi$ as equivalent to the formula $\exists x_1(\exists x_2(\ldots(\exists x_n\varphi)\ldots))$.

Given the conjunctive set of formulae $\mathcal{QL}(\mathcal{CL})$ we define a query as a formula containing free variables. Intuitively, the free variables correspond to the "results" of the query. We distinguish a special form of query when the formula does not contain any free variable.

Definition 7.2. A conjunctive query over a DL $\mathcal{CL}$ is an expression

$$\{x_1, \ldots, x_n \mid \varphi\}$$

where φ is a conjunctive formula ($\varphi \in \mathcal{QL}(\mathcal{CL})$), and $x_1, \ldots, x_n$ are called *distinguished* variables and are exactly the free variable names of φ ($\{x_1, \ldots, x_n\} = \mathrm{free}(\varphi)$). Among the conjunctive queries we distinguish the *boolean conjunctive queries* as those without free variables.

7.1.2 Semantics

The semantics of the conjunctive formulae is given in the same way as for the DL by means of interpretations composed by a domain and an interpretation function (see Chapter 2).

Definition 7.3. Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an interpretation over the set of individual names $\mathcal{O}$, role names $\mathcal{RN}$, and concept names $\mathcal{CN}$. Given a formula $\varphi \in \mathcal{QL}(\mathcal{CL})$, and a mapping ψ from the names in $\mathrm{free}(\varphi)$ to the domain Δ, we say that the interpretation $\mathcal{I}$ *models* φ w.r.t. ψ ($\mathcal{I} \models_\psi \varphi$) when:

$\mathcal{I} \models_\psi x{:}C$	iff $\psi(x) \in C^{\mathcal{I}}$
$\mathcal{I} \models_\psi \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$	iff $(\psi(x), \psi(y)) \in {R_1}^{\mathcal{I}} \cap \ldots \cap {R_n}^{\mathcal{I}}$
$\mathcal{I} \models_\psi x = o$	iff $\psi(x) = o^{\mathcal{I}}$
$\mathcal{I} \models_\psi x = y$	iff $\psi(x) = \psi(y)$
$\mathcal{I} \models_\psi \varphi_1 \wedge \varphi_2$	iff $\mathcal{I} \models_\psi \varphi_1$ and $\mathcal{I} \models_\psi \varphi_2$
$\mathcal{I} \models_\psi \varphi_1 \vee \varphi_2$	iff $\mathcal{I} \models_\psi \varphi_1$ or $\mathcal{I} \models_\psi \varphi_2$
$\mathcal{I} \models_\psi \exists x\varphi$	iff there is $e \in \Delta^{\mathcal{I}}$ s.t. $\psi' = \psi[x/e]$ and $\mathcal{I} \models_{\psi'} \varphi$

Where the notation $\psi[x/e]$ represents the mapping ψ extended by the pair (x, e) if x is not in the domain of ψ, otherwise the original value for x is replaced by e (i.e. $\psi[x/e] = \{(x, e)\} \cup \{(y, e') \in \psi \mid y \not\equiv x\}$). With the simplified notation $\exists x_1 \ldots x_n \varphi$, we say that $\mathcal{I} \models_\psi \exists x_1 x_2 \ldots x_n \varphi$ iff there is an extension ψ' of ψ with a mapping for all the variable names $x_1, \ldots, x_n$ such that $\mathcal{I} \models_{\psi'} \varphi$.

When there is at least a mapping ψ such that $\mathcal{I} \models_\psi \varphi$, then the interpretation $\mathcal{I}$ *models* φ.

We can use the above defined semantics for specifying the meaning of answering a query w.r.t. a given knowledge base Σ. We start from boolean queries; then we are going to extend the definition to general queries.

Definition 7.4. Let Σ be a DL knowledge base, and φ a conjunctive formula in $\mathcal{QL}(\mathcal{CL})$ without free variables. We say that Σ answers (implies) φ (written as $\Sigma \models \varphi$) iff any interpretation satisfying Σ models φ: i.e. for any interpretation $\mathcal{I}$, $\mathcal{I} \models \Sigma$ implies $\mathcal{I} \models \varphi$.

The definition of logical implication for boolean queries provides the basis for the formal semantics of query answering. Analogously to the database setting, the answer to a query is a set of tuples of individual names (see Reiter [1984]). Each tuple corresponds to an instantiation of the conjunctive query into a boolean query where the free variables are bound to the corresponding names in the tuple. In other words, a tuple $\langle o_1, \ldots, o_n \rangle$ belongs to the answer of the query $\{x_1, \ldots, x_n \mid \varphi\}$ w.r.t. the kb Σ iff $\Sigma \models \exists x_1 \ldots x_n (x_1 = o_1 \wedge \ldots \wedge x_n = o_n \wedge \varphi)$.

It is obvious from the definition that query answering can be reduced to logical implication by repeated application of boolean queries with different tuples of individual names substituted for variables. Of course the naive evaluation of such a retrieval could be prohibitively expensive, but would clearly be amenable to optimisation (see Section 7.5 for more details). However, from now on we concentrate on boolean queries by showing how to decide if a query formula without free variables is a logical implication of a given knowledge base.

7.2 Completeness of quasi transitive shrub interpretations

As anticipated in Section 6.2.2 we need to show that cycles can be enforced only by means of Abox assertions (see Proposition 7.15 and Proposition 7.16). For this purpose we are going to use a result of completeness similar to the one already presented in Chapter 3. Canonical transitive shrub interpretations provides a formal characterisation of this "tree-like" structure of the parts of an interpretation which are not constrained by Abox assertions.

Note that we cannot just reuse the completeness result presented in Section 3.13 because the problem of logical implication is different from the one of kb satisfiability. In many logics these two problems can be trivially reduced to each other (for example using negation), but in our case the fact that the query language is different from the assertional language prevents us from taking this simple shortcut.

Our goal is to show that for providing the evidence that a given knowledge base implies a formula, we do not need to consider any arbitrary interpretation but only quasi transitive shrubs. For this purpose we are going to use the very same transformation for interpretations defined in Section 3.2.1.

Theorem 7.5. *Let Σ be a $\mathcal{SH}f$ knowledge base, and φ a conjunctive formula in $\mathcal{QL}(\mathcal{CL})$ without free variables. Then $\Sigma \models \varphi$ iff $\Sigma \models \varphi$ w.r.t. canonical quasi transitive shrub interpretations.*

Proof. The "only if" direction is trivial because quasi transitive shrub interpretations are interpretations as well.

For the "if" direction let us assume that $\Sigma \models \varphi$ w.r.t. canonical quasi transitive shrub interpretations. Let $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ be an arbitrary interpretation satisfying Σ (i.e. $\mathcal{I} \models \Sigma$); we have to show that $\mathcal{I}$ models φ as well (Definition 7.3).

Let $\widehat{\mathcal{I}}^{+} = (\Delta^{\widehat{\mathcal{I}}^{+}}, \cdot^{\widehat{\mathcal{I}}^{+}})$ be the transitive closure of the unravelled interpretation of $\mathcal{I}$; in addition, the mapping δ maps elements of $\Delta^{\widehat{\mathcal{I}}^{+}}$ to Δ (See Definition 3.7 and Definition 3.8). Since we assumed that $\Sigma \models \varphi$ w.r.t. canonical quasi transitive shrub interpretations, then $\widehat{\mathcal{I}}^{+}$ models φ ($\widehat{\mathcal{I}}^{+}$ is a canonical quasi transitive interpretation by Proposition 3.11).

Let ψ be an arbitrary mapping $\psi : \mathcal{V} \to \Delta^{\widehat{\mathcal{I}}^{+}}$; the mapping ψ' is a mapping from $\mathcal{V}$ to Δ defined by $\psi' = \{(v, \delta(u)) \mid (v, u) \in \psi\}$. We are going to show by induction on the structure of the formula φ that if $\widehat{\mathcal{I}}^{+} \models_{\psi} \varphi$ then $\mathcal{I} \models_{\psi'} \varphi$. Note that since $\mathcal{I}$ satisfies Σ, then $\widehat{\mathcal{I}}^{+}$ satisfies the properties in Proposition 3.9 (see Proposition 3.10).

First let us consider the basic cases.

$x{:}C$ If $\widehat{\mathcal{I}}^{+} \models_{\psi} x{:}C$ then $\psi(x) \in C^{\widehat{\mathcal{I}}^{+}}$, and $\psi'(x) = \delta(\psi(x)) \in C^{\mathcal{I}}$ by (3.9d); therefore $\mathcal{I} \models_{\psi'} x{:}C$.

$\langle x,y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$ If $\widehat{\mathcal{I}}^+ \models_\psi \langle x,y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$ then $(\psi(x),\psi(y)) \in R_i^{\widehat{\mathcal{I}}^+}$ for $i = 1,\ldots,n$ and $(\psi'(x),\psi'(y)) = (\delta(\psi(x)),\delta(\psi(y))) \in R_i^{\mathcal{I}}$ by (3.9b); therefore $\mathcal{I} \models_{\psi'} \langle x,y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$.

$x = o$ If $\widehat{\mathcal{I}}^+ \models_\psi x = o$ then $\psi(x) = o^{\widehat{\mathcal{I}}^+}$, and $\psi'(x) = \delta(\psi(x)) = o^{\mathcal{I}}$ by (3.9a); therefore $\mathcal{I} \models_{\psi'} x = o$.

$x = y$ If $\widehat{\mathcal{I}}^+ \models_\psi x = y$ then $\psi(x) = \psi(y)$, and $\psi'(x) = \delta(\psi(x)) = \delta(\psi(y)) = \psi'(y)$; therefore $\mathcal{I} \models_{\psi'} x = y$.

As inductive hypothesis let us assume that, for an arbitrary mapping $\omega : \mathcal{V} \to \Delta^{\widehat{\mathcal{I}}^+}$, if $\widehat{\mathcal{I}}^+ \models_\omega \varphi$ then $\mathcal{I} \models_{\omega'} \varphi$ (and the same holds for φ_1, φ_2). We are going to show that the very same property holds for the formulae $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, and $\exists x\varphi$.

$\varphi_1 \wedge \varphi_2$ If $\widehat{\mathcal{I}}^+ \models_\psi \varphi_1 \wedge \varphi_2$ then $\widehat{\mathcal{I}}^+ \models_\psi \varphi_1$ and $\widehat{\mathcal{I}}^+ \models_\psi \varphi_2$. By inductive hypothesis $\mathcal{I} \models_{\psi'} \varphi_1$ and $\mathcal{I} \models_{\psi'} \varphi_2$; therefore $\mathcal{I} \models_{\psi'} \varphi_1 \wedge \varphi_2$.

$\varphi_1 \vee \varphi_2$ If $\widehat{\mathcal{I}}^+ \models_\psi \varphi_1 \vee \varphi_2$ then $\widehat{\mathcal{I}}^+ \models_\psi \varphi_1$ or $\widehat{\mathcal{I}}^+ \models_\psi \varphi_2$. If $\widehat{\mathcal{I}}^+ \models_\psi \varphi_1$ then $\mathcal{I} \models_{\psi'} \varphi_1$ by inductive hypothesis; therefore $\mathcal{I} \models_{\psi'} \varphi_1 \vee \varphi_2$. Analogously, we come to the very same result if we assume that $\widehat{\mathcal{I}}^+ \models_\psi \varphi_2$.

$\exists x\varphi$ If $\widehat{\mathcal{I}}^+ \models_\psi \exists x\varphi$ there is an element u of $\Delta^{\widehat{\mathcal{I}}^+}$ such that $\widehat{\mathcal{I}}^+ \models_\omega \varphi$ with $\omega = \psi[x/u]$. For the inductive hypothesis $\mathcal{I} \models_{\omega'} \varphi$, where $\omega' = \{(v,\delta(u)) \mid (v,u) \in \omega\}$.

It is easy to show that $\omega' = \psi'[x/\delta(u)]$: if $(v,e) \in \omega'$, then either $v \equiv x$ and $e = \delta(u)$, therefore $(v,e) \in \psi'[x/\delta(u)]$; or $(v,w) \in \omega$ with $\delta(w) = e$, therefore $(v,e) \in \psi'[x/\delta(u)]$ because $\omega = \psi[x/u]$. Analogusly, it can be shown that if $(v,e) \in \psi'[x/\delta(u)]$ then $(v,e) \in \omega'$.

This means that $\mathcal{I} \models_{\psi'[x/\delta(u)]} \varphi$, therefore $\mathcal{I} \models_{\psi'} \exists x\varphi$.

By Definition 7.3 there is a mapping $\psi : \mathcal{V} \to \Delta^{\widehat{\mathcal{I}}^+}$ such that $\widehat{\mathcal{I}}^+ \models_\psi \varphi$. We just shown that there is a mapping $\psi' : \mathcal{V} \to \Delta$ such that $\mathcal{I} \models_{\psi'} \varphi$. Therefore from $\mathcal{I} \models \varphi$ for the arbitrariness of $\mathcal{I}$ we can conclude that $\Sigma \models \varphi$. □

7.3 Answering boolean queries

We are going to present a technique which enables us to decide whether a query formula without free variables is logical consequence of a given knowledge base. Our algorithm cannot answer arbitrary query formulae but only a restricted class representable in a normal form we are going to define. This normal form is general enough for answering the kind of queries we are focusing on: conjunctive queries and disjunctions of conjunctive queries.

First we are going to present the normal form, then we show how connected and unconnected conjunctive queries can be verified by using the normal form.[1] Finally, we show how disjunctions of conjunctive queries can be verified by a slightly more sophisticated reduction to the normal form.

7.3.1 Query normal form

We consider a disjunctive normal form

$$\exists z(\exists \overline{x_1}\varphi_1 \vee \ldots \vee \exists \overline{x_n}\varphi_n) \tag{7.1}$$

where each φ_i is a conjunction of terms of the form of $x{:}C$, $\langle x,y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$, $x = o$ and $x = y$ (i.e. it does not contain any disjunction or existential quantification). In addition, we require that its free variables are exactly $\overline{x_i}$ and z, and all the variables in each φ_i are connected. As the normal form suggests, we differentiate a special variable z whose purpose is "joining" all the different disjuncts, which otherwise do not share any variable.

It is easy to realise that this normal form is not general enough to cover all the valid query formulae as described in Section 7.1.1. The simple formula $\exists xy(\langle x,y\rangle{:}R \vee \langle x,y\rangle{:}S)$ for example cannot be transformed into the normal form. However, conjunctive queries (i.e. not containing any disjunction) or disjunctions of conjunctive queries (not sharing any variable) can be transformed into the normal form by renaming of variables.[2]

The reduction to boolean queries shown in Section 7.1.2 can generate queries that apparently cannot be transformed in the normal form; e.g. the query $\{x_1, x_2 \mid \varphi_1 \vee \varphi_2\}$, once instantiated into a boolean query, becomes $\exists x_1x_2(x_1 = o_1 \wedge x_2 = o_2 \wedge (\varphi_1 \vee \varphi_2)$ for some individual names o_1 and o_2. However, variables being identified with an individual name can be distributed among the disjuncts without altering the semantics, because the uniqueness of the individual name ensures that they will be identified with the

[1]Intuitively, being connected means that there is a path constituted by the role terms connecting each pair of variables (we provide a more formal definition at the end of next section).

[2]Actually the transformation is a little more involved in the case of non connected or disjunctive queries, as described in the next sections.

same element of the domain. Therefore the formula in the example can be transformed into the equivalent $\exists x_1 x_2(x_1 = o_1 \wedge x_2 = o_2 \wedge \varphi_1) \vee \exists x_1 x_2(x_1 = o_1 \wedge x_2 = o_2 \wedge \varphi_2)$, which is a disjunction of conjunctive queries.

For a better explanation of the algorithm we use an alternative equivalent representation of the formula (7.1) by using a set of sets of query terms. The idea behind this alternative representation is that since all the variables are existentially quantified we do not really need to specify the quantification operator for binding a variable name. Therefore we drop the quantification at the beginning of each φ_i and we use the set of terms in it instead of the formula itself; i.e. the formula $\exists z(\exists \overline{x_1}\varphi_1 \vee \ldots \vee \exists \overline{x_n}\varphi_n)$ is represented by $\{\phi_1, \ldots, \phi_n\}\,|_z$, where ϕ_i is the set of all the conjuncts in φ_i. The notation $\{\ldots\}\,|_z$ specifies that z is the special variable joining the disjuncts.

Given the one-to-one mapping between the set and explicit forms of the query formula, we use the same notation for describing the notion that an interpretation models a formula (see Definition 7.3). Given the formula $\{\phi_1, \ldots, \phi_n\}\,|_z$ we say that an interpretation $\mathcal{I}$ models the formula w.r.t a mapping ψ ($\mathcal{I} \models_\psi \{\phi_1, \ldots, \phi_n\}\,|_z$) iff there is an element (disjunct) ϕ_i such that the domain of ψ includes all the variables in ϕ_i and for each term t in ϕ_i, $\mathcal{I} \models_\psi t$. It is easy to see that, for the class of queries representable in the given normal form, the two notions of model for a formula coincide. To simplify the notation, when there is no ambiguity we are going to write formulae having a single element (disjunct) in the the form $\mathcal{I} \models_\psi \phi$ instead of $\mathcal{I} \models_\psi \{\phi\}\,|_z$.

The set representation enables us to provide a formal definition of the property of connectedness for a formula. First we introduce the notion of a *path* connecting two variables.

Definition 7.6. A *path* connects two variables and it is defined in the following recursive way:

- the set $\{\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n\}$ is a path connecting x to y;
- if the set φ is a path connecting x to y, the set φ' is a path connecting y to z, and $\varphi \cap \varphi'$ is empty, then $\varphi \cup \varphi'$ is a path connecting x to z;[3]
- if φ is a path from x to y, then it is a path connecting y to x as well.

A *cycle* is a path connecting a variable to itself. A set of terms contains a path (cycle) if a subset of it is a path (cycle).

[3]We require the disjointness of the two paths because we want to prevent cases like $\varphi = \varphi' = \{\langle x, y\rangle{:}R\}$ from being wrongly classified as paths starting and ending with the same variable.

By using the previous definition we can now provide a formal characterisation of a connected formula. A query formula in the normal form is connected iff in every element (disjunct) there is a path between any pair of variables.

7.3.2 Conjunctive queries

Since nested existential quantifiers can easily be prepended to the formula by renaming of the variables, a connected conjunctive boolean query is trivially in query normal containing a single element. In this section we consider boolean queries in the form

$$\exists {x_1}^{(1)} \ldots {x_{n_1}}^{(1)} \varphi^{(1)} \wedge \ldots \wedge \exists {x_1}^{(k)} \ldots {x_{n_k}}^{(k)} \varphi^{(k)},$$

where the formulae $\varphi^{(1)}, \ldots, \varphi^{(k)}$ are connected and do not contain any existential quantifier nor disjunctions. Without loss of generality we can assume that all the variable names are distinct in each formula.

It is intuitively the fact that, since formulae do not interact one to each other, such queries may be answered by considering one formula at a time. This is indeed the case, and we are going to show it formally in the following paragraphs.

Since we can prove a more general result about unconnected conjunction, which will be used in the next section, we are going to relax the constraint on the form of the query. Let us consider the query answering problem

$$\Sigma \models \exists {x_1}^{(1)} \ldots {x_{n_1}}^{(1)} \varphi^{(1)} \wedge \ldots \wedge \exists {x_1}^{(k)} \ldots {x_{n_k}}^{(k)} \varphi^{(k)}; \qquad (*)$$

where there is not any restriction on the form of $\varphi^{(1)}, \ldots, \varphi^{(k)}$ other than the fact that they do not share any free variables. We are going to show that the query answering problem $(*)$ is verified iff $\Sigma \models \exists {x_1}^{(i)} \ldots {x_{n_i}}^{(i)} \varphi^{(i)}$ for each $i = 1, \ldots, k$. The unconnected conjunctive queries we are considering in this section are a particular case of $(*)$.

Since the "only if" direction (i.e. $\Rightarrow$) is trivially satisfied by definition, we are going to concentrate on the other direction. Let us assume that the variable names are distinct in each formula $\exists {x_1}^{(i)} \ldots {x_{n_i}}^{(i)} \varphi^{(i)}$, and $\Sigma \models \exists {x_1}^{(i)} \ldots {x_{n_i}}^{(i)} \varphi^{(i)}$ for each $i = 1, \ldots, k$. Let $\mathcal{I}$ be an arbitrary interpretation satisfying Σ ($\mathcal{I} \models \Sigma$); we have to show that $\mathcal{I}$ models the query formula in $(*)$ as well.

Since $\Sigma \models \exists {x_1}^{(i)} \ldots {x_{n_i}}^{(i)} \varphi^{(i)}$ for each $i = 1, \ldots, k$, then $\mathcal{I} \models \exists {x_1}^{(i)} \ldots {x_{n_i}}^{(i)} \varphi^{(i)}$; therefore there are k mappings $\psi_1, \ldots, \psi_k$ for the variable names such that $\mathcal{I} \models_{\psi_i} \varphi^{(i)}$.

Let us consider the formula

$$\exists {x_1}^{(1)}\ldots {x_{n_1}}^{(1)}\ldots {x_1}^{(k)}\ldots {x_{n_k}}^{(k)}(\varphi^{(1)} \wedge \ldots \wedge \varphi^{(k)})$$

which is equivalent to the original query formula in ($*$), because the variable names are distinct. Since the variable names are distinct, making the union of the mappings does not alter the variables relevant for each single formula $\varphi^{(i)}$; therefore, the mapping $\psi = \psi_1 \cup \ldots \cup \psi_k$ is such that $\mathcal{I}$ models $(\varphi^{(1)} \wedge \ldots \wedge \varphi^{(k)})$ w.r.t. ψ. Therefore $\mathcal{I}$ models the query formula in ($*$) as well.

7.3.3 Disjunction of conjunctive queries

The last class of queries we consider is the disjunction of conjunctive queries. In this class, the query formula is a disjuntion of formulae like $\phi_1 \vee \ldots \vee \phi_n$, where each ϕ_i is in the form

$$\exists {x_1}^{(1)}\ldots {x_{n_1}}^{(1)}\varphi^{(1)} \wedge \ldots \wedge \exists {x_1}^{(k)}\ldots {x_{n_k}}^{(k)}\varphi^{(k)},$$

and the formulae $\varphi^{(1)}, \ldots, \varphi^{(k)}$ are conjunctive queries; i.e. do not contain any existential quantifier nor disjunctions. A disjunctive query answering problem can be formulated as

$$\begin{aligned}\Sigma \models\ & (\varphi_{1,1} \wedge \ldots \wedge \varphi_{1,n_1})\\ & \vee \ldots\\ & \vee (\varphi_{k,1} \wedge \ldots \wedge \varphi_{k,n_k})\end{aligned}$$

where each $\varphi_{i,j}$ is a connected conjunctive formula. We can assume without loss of generality that all the existential quantifiers appear at the beginning of each subformula $\varphi_{i,j}$. First the query formula is transformed into the conjunctive normal form by using the distributive property of boolean constructors (i.e. $(A \wedge B) \vee C$ is equivalent to $(A \vee C) \wedge (B \vee C)$). In this way we obtain an equivalent query formula where the conjunctions appear at the top level, like

$$\begin{aligned}\Sigma \models\ & (\varphi_{1,l_{(1,1)}} \vee \ldots \vee \varphi_{k,l_{(1,k)}})\\ & \wedge \ldots \qquad\qquad (*)\\ & \wedge (\varphi_{1,l_{(r,1)}} \vee \ldots \vee \varphi_{k,l_{(r,k)}}).\end{aligned}$$

By using the result shown in Section 7.3.2 we can consider each conjunct $(\varphi_{1,l_{(1,i)}} \vee \ldots \vee \varphi_{k,l_{(i,k)}})$ independently from the rest of the formula. Therefore, we need to show how to answer to a query in the form

$$\Sigma \models \varphi_1 \vee \ldots \vee \varphi_k$$

where $\varphi_1, \ldots, \varphi_k$ are connected conjunctive formulae. The formula $\varphi_1 \vee \ldots \vee \varphi_k$ has a close resemblance with the query normal form described in Section 7.3.1. What is missing is a common "joining" variable connecting the disjuncts (i.e. the variable z in Formula (7.1)).

The idea is to transform the query into the normal form by choosing arbitrarily one variable from each disjunct and "promoting" it as the joining variable name. In the next Proposition we are going to show that this approach is indeed correct and allows us to solve the query answering problem.

Proposition 7.7. *Let Σ be a knowledge base and $\exists z_1\varphi_1 \vee \ldots \vee \exists z_k\varphi_k$ a query formula such that $\exists z_1\varphi_1, \ldots, \exists z_k\varphi_k$ are connected conjunctive query formulae not containing the variable name z; then $\Sigma \models \exists z_1\varphi_1 \vee \ldots \vee \exists z_k\varphi_k$ iff $\Sigma \models \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$.*[4]

Proof. Let us assume that $\Sigma \models \exists z_1\varphi_1 \vee \ldots \vee \exists z_k\varphi_k$ and let $\mathcal{I}$ be an interpretation satisfying Σ. By assumption $\mathcal{I} \models \exists z_1\varphi_1 \vee \ldots \vee \exists z_k\varphi_k$; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \exists z_1\varphi_1 \vee \ldots \vee \exists z_k\varphi_k$. By definition there is one of the disjunct $\exists z_\ell\varphi_\ell$ such that $\mathcal{I} \models_\psi \exists z_\ell\varphi_\ell$; therefore $\mathcal{I} \models_{\psi'} \varphi_\ell$, where $\psi' = \psi[z_\ell/e]$ for some $e \in \Delta^{\mathcal{I}}$. Let ψ'' be a mapping such that $\psi'' = \psi[z/\psi'(z_\ell)]$. The mapping ψ'' satisfies $\mathcal{I} \models_{\psi''} \varphi_\ell[z_\ell/z]$ because φ_ℓ does not contain the variable z, so $\mathcal{I} \models_{\psi''} (\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_1/z])$, therefore $\mathcal{I} \models_{\psi''} \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$ by definition. This means that $\mathcal{I} \models \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$. By the arbitrariness of the choice of $\mathcal{I}$ we can conclude that $\Sigma \models \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$.

For the other direction, let us assume that $\Sigma \models \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$, and $\mathcal{I}$ be an interpretation satisfying Σ. By assumption there is a mapping ψ such that $\mathcal{I} \models_\psi \exists z(\varphi_1[z_1/z] \vee \ldots \vee \varphi_k[z_k/z])$; therefore there is a disjunct $\varphi_\ell[z_\ell/z]$ and a mapping $\psi' = \psi[z/e]$, for some $e \in \Delta^{\mathcal{I}}$, such that $\mathcal{I} \models_{\psi'} \varphi_\ell[z_\ell/z]$. Let ψ'' be a new mapping such that $\psi'' = \psi[z_\ell/\psi'(z)]$. Since z_ℓ is not appearing as a free variable in $\varphi_\ell[z_\ell/z]$, then $\mathcal{I} \models_{\psi''} (\varphi_\ell[z_\ell/z])[z/z_\ell]$. In addition, since z is not appearing in φ_ℓ,

[4]The formula $\varphi[x/y]$ indicates the formula φ in which all the free occurrences of the variable x, x itself is substituted by y.

$(\varphi_\ell[z_\ell/z])[z/z_\ell] \equiv \varphi_\ell$ so $\mathcal{I} \models_\psi \exists z_\ell \varphi_\ell$ (the only differences between ψ and ψ'' are the mapping of variables z and z_ℓ). Therefore $\mathcal{I} \models_\psi \exists z_1 \varphi_1 \vee \ldots \vee \exists z_k \varphi_k$ and consequently $\mathcal{I} \models \exists z_1 \varphi_1 \vee \ldots \vee \exists z_k \varphi_k$. By the arbitrariness of the choice of $\mathcal{I}$ we can conclude that $\Sigma \models \exists z_1 \varphi_1 \vee \ldots \vee \exists z_k \varphi_k$. □

7.4 Answering queries in normal form

In this section we describe the actual algorithm for answering queries in the normal form described in Section 7.3.1. The algorithm is described in terms of a set of non-deterministic rules which transform the initial implication by modifying the query formulae and possibly the KB on the left hand side. The goal of the rules is producing an implication test of the form $\Sigma \models \exists z(z{:}C_1 \sqcup \ldots \sqcup z{:}C_n)$; when such a form is reached the rules are no longer applied. Such a simple implication can be verified by checking whether the knowledge base Σ, extended with the axiom $\{\top \sqsubseteq (\neg C_1 \sqcap \ldots \sqcap \neg C_n)\}$ is unsatisfiable. In fact, the formula $\exists z(z{:}C)$ is implied by the KB Σ iff in every interpretation satisfying Σ the extension of the concept C is non-empty. We can rephrase the question by verifying whether there is an interpretation in which the extension is empty (i.e. the negation of the concept itself is equal to the whole domain).

We require that the algorithm is correct, complete, and terminating. We are going to build these transformation rules according to these requirements. In the next sections we prove that this is actually the case.

7.4.1 Query transformation rules

Rules transform query problems of the form $\Sigma \models \{\phi_1, \ldots, \phi_n\}\,|_z$ into "simpler"[5] problems of the same form. An element ϕ_i is selected such that it matches the preconditions of one or more rules. One of the matching rules is selected according to a strategy, and the query is transformed as described in the rule. The procedure is repeated until no elements of the query formula satisfy the preconditions of any rule. At this point the resulting query formula will be of the form

$$\{\{z{:}C_1\}, \ldots, \{z{:}C_n\}\}\,|_z$$

which is equivalent to the formula $\exists z(z{:}C_1 \sqcup \ldots \sqcup C_n)$.

[5]Simpler in the sense that the query after the application of a rule is closer to the target formula $\{\{z{:}C_1\}, \ldots, \{z{:}C_n\}\}\,|_z$.

In general, rules substitute the element (disjunct) satisfying the precondition with a new disjunct, and possibly they modify the KB Σ in the left hand side:

$$\begin{tikzcd}
\Sigma \arrow[dr, dash] & \models \{ & \phi \arrow[dl, dash] & , \phi_1 \arrow[dd] & , \ldots , & \phi_n \arrow[dd] & \}|_z \\
& \boxed{\textbf{rule}} \arrow[dl] \arrow[dr] & & & & & \\
\Sigma' & \models \{ & \phi' & , \phi_1 & , \ldots , & \phi_n & \}|_z
\end{tikzcd}$$

In some cases, the selected element ϕ may be substituted by more than one elements $\phi'_1, \ldots, \phi'_k$; but since this transformation is performed by the last two rules only, we maintain the simpler single element substitution for the rest of the rules.

Before presenting the rules we need to introduce some notation which is going to simplify the following exposition. As introduced in Chapter 6, we are considering knowledge bases as sets of Tbox and Abox assertions instead of a pair of sets. This simplifies the notation, when new axioms are added to the KB (see *nominal elimination* rule).

A further simplification is made to the role syntax in the role terms, which are represented in a set like notation instead of a conjunction; i.e. the expression $R_1 \sqcap \ldots \sqcap R_n$ is represented by $\sqcap\{R_1, \ldots, R_n\}$. In cases where the actual elements of the sets are not used, the set will be indicated by a calligraphic character like $\mathcal{R}$ or $\mathcal{S}$; i.e. the conjunction is $\sqcap\mathcal{R}$ (see for example the *role collapsing* rule).

New concept names are introduced in some of the rules; we assume that these new names do not appear in the knowledge base to which the rule is applied. Some of these new names are uniquely associated to individual names and are called *representative concepts*; they are denoted as P_o, where o is the associated individual name (see the *nominal elimination* rule for an example). It is important to note that the representative concepts are unique w.r.t. each individual name. Therefore, if the assertion $o{:}P_o$ is already in Σ, then the knowledge base $\Sigma' = \Sigma \cup \{o{:}P_o\}$ is not modified. New concept names, different from the representative concept, can be introduced by appropriate rules (e.g. the *simple inverse rolling up* rule); they are unique w.r.t. the actual rule application.

Finally, in some of the rules the presence a concept term of the form $y{:}C$ may be required among the other conditions (e.g. the rolling up ones). If such term is not in the formula, we assume an implicit $y{:}\top$ (see the *simple rolling up* rule).

The ordering in which the rules are presented is not arbitrary, since the strategy for their application gives the precedence according to this order. This assumption is essential for some of the proofs we are going to show.

1. **Equality elimination**: if $\{x = y\} \subseteq \phi$ (or $\{y = x\} \subseteq \phi$), x,y are variable names, $x \not\equiv z$, and $\phi[x/y]$ indicates the set ϕ in which all the variables x are substituted by y; then

$$\begin{aligned}\Sigma' &= \Sigma\\ \phi' &= \phi[x/y] \setminus \{y = y\}\end{aligned}$$

   Note that $y = y$ is used because x has been substituted by y in $\phi[x/y]$.

2. **Conjunction elimination**: if $\{x{:}C_1, x{:}C_2\} \subseteq \phi$, then

$$\begin{aligned}\Sigma' &= \Sigma\\ \phi' &= \{x{:}(C_1 \sqcap C_2)\} \cup (\phi \setminus \{x{:}C_1, x{:}C_2\}).\end{aligned}$$

3. **Role collapsing**: if $\{\langle x_1, x_2\rangle{:}\sqcap\mathcal{R}, \langle x_1, x_2\rangle{:}\sqcap\mathcal{S}\} \subseteq \phi$ then

$$\begin{aligned}\Sigma' &= \Sigma\\ \phi' &= \{\langle x_1, x_2\rangle{:}\sqcap(\mathcal{R} \cup \mathcal{S})\}\\ &\qquad \cup (\phi \setminus \{\langle x_1, x_2\rangle{:}\sqcap\mathcal{R}, \langle x_1, x_2\rangle{:}\sqcap\mathcal{S}\}).\end{aligned}$$

4. **Contradiction elimination**: if $\{x = o_1, x = o_2\} \subseteq \phi$, and o_1 is different from o_2; then

$$\begin{aligned}\Sigma' &= \Sigma\\ \phi' &= \{z{:}\bot\}\,.\end{aligned}$$

5. **Nominal elimination**: if $\{z = o\} \subseteq \phi$ and there is no term of the form $\langle x_1, x_2\rangle{:}\sqcap\mathcal{R}$ in ϕ; then

$$\begin{aligned}\Sigma' &= \Sigma \cup \{o{:}P_o\}\\ \phi' &= \{z{:}P_o\} \cup (\phi \setminus \{z = o\}).\end{aligned}$$

   Where P_o is the representative concept name associated to the individual o.

6. **Role elimination**: if $\{\langle x_1,x_2\rangle{:}\sqcap\{R_1,\dots,R_n\}\}\subseteq\phi$, $n>1$, and there are two indexes ℓ,j s.t. $R_j\preceq R_\ell$; then

$$\begin{aligned}\Sigma'&=\Sigma\\ \phi'&=\{\langle x_1,x_2\rangle{:}\sqcap(\{R_1,\dots,R_n\}\setminus\{R_\ell\})\}\\ &\quad\cup(\phi\setminus\{\langle x_1,x_2\rangle{:}\sqcap\{R_1,\dots,R_n\}\}).\end{aligned}$$

7. **Shortcut elimination**: If

$$\{\langle x_0,x_1\rangle{:}\sqcap\mathcal{R}_1,\dots,\langle x_{n-1},x_n\rangle{:}\sqcap\mathcal{R}_n,\langle x_0,x_n\rangle{:}\sqcap\{R_1,\dots,R_k\}\}\subseteq\phi,$$

and there is an index $\ell\le k$ and a transitive role S such that $S\preceq R_\ell$, and in every set of roles $\mathcal{R}_i$, there is at least a role S_i such that $S_i\preceq S$; then

$$\begin{aligned}\Sigma'&=\Sigma\\ \phi'&=\begin{cases}\{\langle x_0,x_n\rangle{:}\sqcap(\{R_1,\dots,R_k\}\setminus\{R_\ell\})\} & \text{if } k>1\text{, or}\\ \quad\cup(\phi\setminus\{\langle x_0,x_n\rangle{:}\sqcap\{R_1,\dots,R_k\}\}) & \\ \phi\setminus\{\langle x_0,x_n\rangle{:}R_1\} & \text{if } k=1.\end{cases}\end{aligned}$$

8. **Simple rolling up**: if $\{\langle x,y\rangle{:}R,y{:}C\}\subseteq\phi$, $y\not\equiv z$,[6] and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma'&=\Sigma\\ \phi'&=\{x{:}\exists R.C\}\cup(\phi\setminus\{\langle x,y\rangle{:}R,y{:}C\}).\end{aligned}$$

9. **Simple inverse rolling up**: if $\{\langle y,x\rangle{:}R,y{:}C\}\subseteq\phi$, $y\not\equiv z$, and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma'&=\Sigma\cup\{C\sqsubseteq\forall R.P_{R^-.C}\}\\ \phi'&=\{x{:}P_{R^-.C}\}\cup(\phi\setminus\{\langle y,x\rangle{:}R,y{:}C\}).\end{aligned}$$

Where $P_{R^-.C}$ is a new concept name not appearing in Σ.

10. **Functional rolling up**: if $\{\langle x,y\rangle{:}\sqcap\{R_1,\dots,R_n\},y{:}C\}\subseteq\phi$, $n>1$, $y\not\equiv z$,

[6]We are going to use the symbol $\equiv$, instead of $=$, to denote syntactic equivalence for avoiding confusion with the use of $=$ in query terms (see section 7.1.1).

there is a label L s.t. $\{R_1, \ldots, R_n\} \subseteq L$,[7] and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma' &= \Sigma \\ \phi' &= \{x{:}(\exists R_1.C \sqcap \exists R_2.\top \sqcap \ldots \sqcap \exists R_n.\top)\} \\ &\quad \cup (\phi \setminus \{\langle x,y\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\}).\end{aligned}$$

11. **Functional inverse rolling up**: if $\{\langle y,x\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\} \subseteq \phi$, $n > 1$, $y \not\equiv z$, there is a label L s.t. $\{R_1, \ldots, R_n\} \subseteq L$, and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma' &= \Sigma \\ \phi' &= \{y{:}(C \sqcap \exists R_2.\top \sqcap \ldots \sqcap \exists R_n.\top), \langle y,x\rangle{:}R_1\} \\ &\quad \cup (\phi \setminus \{\langle y,x\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\}).\end{aligned}$$

12. **Nominal rolling up**: if $\{y = o, \langle x,y\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\} \subseteq \phi$, $y \not\equiv z$, and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma' &= \Sigma \cup \{o{:}P_o\} \\ \phi' &= \{x{:}(\exists R_1.(C \sqcap P_o) \sqcap \exists R_2.P_o \sqcap \ldots \sqcap \exists R_n.P_o)\} \\ &\quad \cup (\phi \setminus \{y = o, \langle x,y\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\}).\end{aligned}$$

Where P_o is the representative concept name associated to the individual o.

13. **Nominal inverse rolling up**: if $\{y = o, \langle y,x\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\} \subseteq \phi$, $y \not\equiv z$, and ϕ does not contain any other term involving y; then

$$\begin{aligned}\Sigma' &= \Sigma \cup \big\{o{:}P_o, (C \sqcap P_o) \sqsubseteq (\forall R_1.P_{R_1^- y} \sqcap \ldots \sqcap \forall R_n.P_{R_n^- y})\big\} \\ \phi' &= \big\{x{:}(P_{R_1^- y} \sqcap \ldots \sqcap P_{R_n^- y})\big\} \\ &\quad \cup (\phi \setminus \{y = o, \langle y,x\rangle{:}\sqcap\{R_1,\ldots,R_n\}, y{:}C\}).\end{aligned}$$

Where P_o is the representative concept name associated to the individual o, and $P_{R_1^- y}, \ldots, P_{R_n^- y}$ are new concept names not appearing in Σ.

[7]The set of labels of a given KB are defined in Definition 3.4.

14. **Cycle breaking**: if $\{y = o, \langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\} \subseteq \phi$, and in
$$(\phi \setminus \{\langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\})$$
there is a path from x to y (see Definition 7.6), or $x \equiv y$; then
$$\begin{aligned}\Sigma' &= \Sigma \cup \{o{:}P_o\}\\ \phi' &= \{x{:}(\exists R_1.P_o \sqcap \dots \sqcap \exists R_n.P_o)\}\\ &\quad \cup (\phi \setminus \{\langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\}).\end{aligned}$$
Where P_o is the representative concept name associated to the individual o.

15. **Simple cycle breaking**: if $\{y = o, \langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\} \subseteq \phi$, and $n > 1$; then
$$\begin{aligned}\Sigma' &= \Sigma \cup \{o{:}P_o\}\\ \phi' &= \{\langle x, y\rangle{:}R_1, x{:}(\exists R_2.P_o \sqcap \dots \sqcap \exists R_n.P_o)\}\\ &\quad \cup (\phi \setminus \{\langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\}).\end{aligned}$$
Where P_o is the representative concept name associated to the individual o.[8]

16. **Inverse cycle breaking**: if $\{y = o, \langle y, x\rangle{:}\sqcap\{R_1, \dots, R_n\}\} \subseteq \phi$, and in $(\phi \setminus \{\langle y, x\rangle{:}\sqcap\{R_1, \dots, R_n\}\})$ there is a path from x to y, or $x \equiv y$; then
$$\begin{aligned}\Sigma' &= \Sigma \cup \left\{o{:}P_o, P_o \sqsubseteq (\forall R_1.P_{R_1\text{-}y} \sqcap \dots \sqcap \forall R_n.P_{R_n\text{-}y})\right\}\\ \phi' &= \left\{x{:}(P_{R_1\text{-}y} \sqcap \dots \sqcap P_{R_n\text{-}y})\right\} \cup (\phi \setminus \{\langle y, x\rangle{:}\sqcap\{R_1, \dots, R_n\}\}).\end{aligned}$$
Where P_o is the representative concept name associated to the individual o, and $P_{R_1\text{-}y}$,…,$P_{R_n\text{-}y}$ are new concept names not appearing in Σ.

17. **Simple nominal introduction**: if $\{\langle x, y\rangle{:}\sqcap\{R_1, \dots, R_n\}\} \subseteq \phi$, $n > 1$, there is not any label L such that $\{R_1, \dots, R_n\} \subseteq L$, and there is not any term like $y = o$ in ϕ, and $o_1, \dots, o_k$ are the individual names appearing in Σ; then k new disjuncts are added to the query formula in place of ϕ:
$$\Sigma \models \{\phi \cup \{y = o_1\}, \dots, \phi \cup \{y = o_k\}, \phi_1, \dots, \phi_n\}|_z$$

[8]The term $\langle x, y\rangle{:}R_1$ is left in the query for avoiding the problem of breaking the connectedness of the formula.

where $\phi_1, \ldots, \phi_n$ are the original disjuncts.

18. **Cyclic nominal introduction**: if ϕ contains a cycle involving the variable y, none of the variables y_i involved in the cycle appear in a term like $y_i = o$ in ϕ, and $o_1, \ldots, o_k$ are the individual names appearing in Σ, then new disjuncts are added to the query formula in place of ϕ:

$$\Sigma \models \{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \\ eq^{(1)} \cup \phi, \ldots, eq^{(l)} \cup \phi, \phi_1, \ldots, \phi_n\}|_z$$

where $\phi_1, \ldots, \phi_n$ are the original disjuncts, and $eq^{(1)}, \ldots, eq^{(l)}$ are all the possible combinations of equality terms among pairs of variable names y_i, y_j involved in the cycle; i.e. $eq^{(k)} = \{y_i = y_j\}$.

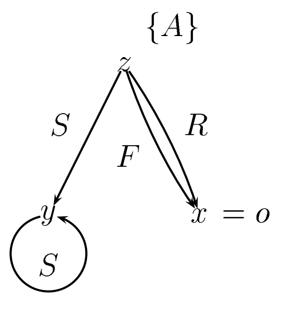
7.4.2 Notes on transformation rules

In this section we explain in detail the rationale behind the rules introduced in Section 7.4.1; we provide the intuition on their correctness, while the formal proofs are in the following sections.

The underlying idea is that a conjunctive formula, represented by a set of terms, can be visualised as a graph. The nodes of the graph are the variable names, labeled by the set of concept terms applying to the variable node. Nodes are connected by edges representing the role terms. In addition, equality terms are explicitly attached to the appropriate node. For example, the query expression

$$\{z{:}A, \langle z, x\rangle{:}R \sqcap F, x = o, \langle z, y\rangle{:}S, \langle y, y\rangle{:}S\}$$

is represented as the graph



A formula in normal form is a set of conjunctive formulae sharing a common variable; by the graph analogy it corresponds to a set of graphs "hooked" on the joining variable (the z in Formula (7.1)). Each graph corresponds to one of the elements in the normal form.

Note the similarity between the graph representation of a query and of an interpretation (see Chapter 2). The main difference is that, in the case of interpretations, nodes are elements of the domain and are labelled by concept names only (in interpretation graphs, edges are labelled by sets of names, but this is just a different notation for the conjunction constructor). The parallel between the two representations enables us to view query answering under the different perspective of graph matching. In fact, given an interpretation and a conjunctive query, we can see the question of whether the interpretation satisfies the query as a problem of verifying if the graph corresponding to the query can be matched against the graph representing the interpretation. The presence of disjunction in the query language slightly blurs this perspective; however, since we restrict ourselves to disjunctions of conjunctive queries we can consider the non-deterministic matching with one of the graphs in the disjunctive formula. The graph matching perspective is very useful for understanding the rationale behind the algorithm. In particular, it enables us to build and explain examples[9] in a very intuitive way.

The purpose of the transformation rules is to collapse each graph contained in a disjunctive formula into a graph composed of a single node and without any edges. The rules are grouped according to the kind of transformation they perform on the graph.

Normalisation rules

Due to the role conjunction constructor, more than one formula may correspond to the very same graph; e.g. both the formulae $\{\langle z, x\rangle{:}R \sqcap F\}$ and $\{\langle z, x\rangle{:}R, \langle z, x\rangle{:}F\}$ are represented by the very same graph. Moreover, there are valid formulae that do not fit nicely with the graph idea; e.g. formulae having multiple concept or equality terms applied to the very same variable name (like $\{z{:}A, z{:}B\}$ or $\{x = y, y = o\}$). For this reason, the first five rules are designed to transform the formula in a normal form which correspond univocally to a graph representation. We call these five rules *normalisation* rules.

The *equality elimination* rule eliminates equality terms by renaming the appropriate variables. Note that the rule is written in such a way that it never eliminates the joining

[9]And counterexamples as well. Most of the problems discovered in the development of the algorithm have been spotted by using the graph matching analogy.

variable z (for this reason, both the $x = y$ and $y = x$ cases must be considered).

The *conjunction elimination* and *role collapsing* rules serve the same purpose of gathering together concept (role) terms applying to the vary same variable (pair of variables) by means of the DL conjunction constructor.

The *contradiction elimination* rule takes care of formulae which cannot be satisfied by any interpretation because of two individual names. In fact, by the unique name assumption two different individual names cannot be interpreted by the very same element of the domain. The contradictory formula is substituted by the simpler $\{z{:}\bot\}$ which has no models either. The choice of the substituting formula is driven by the fact that the resulting formula must contain the connecting variable z, and should be connected (see the definition of normal form in Section 7.3.1). An alternative choice would have been the elimination of the element (i.e. the disjunct ϕ is simply eliminated from the query). However, this choice has been ruled out to avoid the necessity of taking into account cases in which the formula contains a single disjunct (e.g. like $\{\{z = a, z = b\}\}$).

The last normalisation rule, the *nominal elimination* rule, is needed for transforming a formula in which the "linking" variable is identified with an individual name (we will come back to this case later on, when we described the rest of the rules). It is easy to see that the number of times the normalisation rules can be sequentially applied to the same disjunct is bounded by the number of terms plus one (because of the *nominal elimination* rule).

Role elimination rules

The *role elimination* and *shortcut elimination* rules remove redundant edges from the graph. This is necessary because parts of the graph that may appear as a cycle (or as multiple edges connecting two nodes) may not really be cycles. If these "false" cycles are not correctly handled then the nominal introduction rules would be incomplete (see the proofs in Proposition 7.15 and Proposition 7.16). The following example shows why the absence of the *role elimination* rule would cause problems.

Example 7.1
Let us consider for example the query $\langle a, x\rangle{:}R \sqcap S$ against the knowledge base $\Sigma = \{S \sqsubseteq R, a{:}(\exists S.\top)\}$. Clearly, each interpretation satisfying Σ is a model for $\langle a, x\rangle{:}R \sqcap S$; therefore $\Sigma \models \exists x(\langle a, x\rangle{:}R \sqcap S)$ (i.e. $\Sigma \models \{\{\langle a, x\rangle{:}R \sqcap S\}\}$). On the other hand, the formula $\{\langle a, x\rangle{:}R \sqcap S\}$ is a candidate for the *simple nominal introduction* rule, which transforms the query into $\Sigma \models \exists x(\langle a, x\rangle{:}R \sqcap S \land x = a)$. However, this new query

formula is not implied by Σ. The problem is that, even though the query graph shows two entering edges into the node x, the fact that S is included in R implies that an edge labeled S is always an edge R as well.

The *shortcut elimination* rule is conceptually similar, but it covers the cases in which the transitivity restriction forces the existence of edges not explicitly appearing in the query formula itself.

In figure 7.4.2 these "phantom" edges are depicted as dotted lines. Solid lines denote explicit edges, while dashed lines represent implied edges. Finally, undirected edges represent arbitrary connection with the rest of the graph.

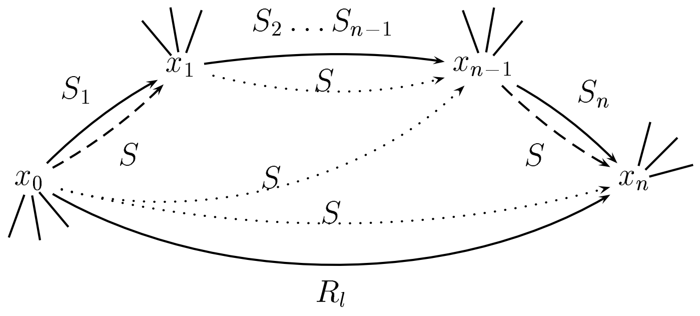


Figure 7.1: Prerequisites for *shortcut elimination* rule.

Since the proposed technique works only when in cycles there are no transitives role (or roles including transitive ones). The *shortcut elimination* rule can transform a query which is apparently not covered by the algorithm into a suitable one.

Rolling up rules

The next three pairs of rules (rolling up and its inverse version) are the core of the graph collapsing procedure. They act on "leaves" of the graph,[10] eliminating the leaf node and the edge(s) connecting it to the parent node. We call them "rolling up" because the information about the removed node and edge(s) is incorporated into the parent node via a concept term.

The *simple rolling up* rule is the more intuitive one; and the easiest way of understanding it is to consider the first order translation of the DL constructor $\exists R.C$, which is $\{x \mid \exists y(R(x, y) \wedge C(y))\}$ (see Borgida [1994]). The translation correspond exactly to the part of the query $\{\langle x, y\rangle{:}R, y{:}C\}$ involved in the rule.

[10]The condition of being leaves is enforced by the requirement "$\dots \phi$ does not contain any other term involving y".

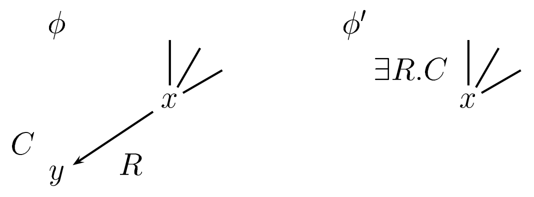


Figure 7.2: Application of *simple rolling up* rule.

Note that any reference of the fact that the edge was connecting x to y is lost in the application of the rule; therefore the rule can be applied only if there are not any other terms involving y itself (i.e. y is a leaf node). Rolling up a non–leaf node may produce a wrong aswer to the query as shown in the following example.

Example 7.2
To show why this can cause problems, let us consider the simple query

$$\{\{\langle w, x\rangle{:}R, \langle x, y\rangle{:}S\}\}$$

and the kb $\Sigma = \{a{:}(\exists R.\top \sqcap \exists S.\top)\}$. Clearly, Σ does not imply the given query because there is a model satisfying Σ and not having a path with the two roles in sequence (e.g. the interpretation $a^{\mathcal{I}} = 1$, $R^{\mathcal{I}} = \{(1, 2)\}$, and $S^{\mathcal{I}} = \{(1, 3)\}$ satisfies Σ). However, if the *simple rolling up* rule is applied to the term $\langle w, x\rangle{:}R$, the resulting query formula is $\{\{w{:}(\exists R.\top), \langle x, y\rangle{:}S\}\}$. It is easy to realise that the latter query is a logical implication of Σ (i.e. identifying both w and x with a); therefore the rule application would not be correct.

The *simple rolling up* rule is applied only if there is a single edge connecting the leaf node; e.g. the rule cannot be a applied to the term to the term $\langle x, y\rangle{:}R \sqcap S$. Again, the reason lies in the loss of information which would hide the fact that there is only one confluent node (i.e. y). However, there are cases in which this uniqueness is redundant because there are other contraints that guarantee it. This is the rationale behind the *functional rolling up* and *nominal rolling up* rules.

Let us consider the case when the leaf node y is connected to the "father" node by a set of roles (i.e. $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$) all being subroles of a common functional role F (see Figure 7.3 (a)). Variable y can be duplicated $n - 1$ times, because the common functional role F forces y and its copies to be interpreted as the very same value (Figure 7.3 (b)). All the generated nodes can then be rolled up as described for the *simple rolling up* rule.

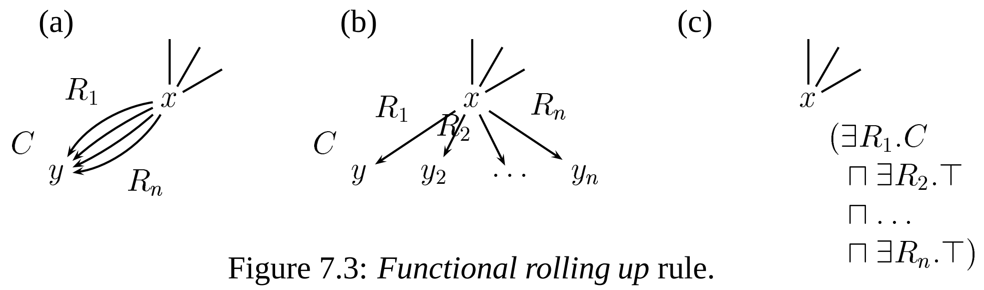


Figure 7.3: *Functional rolling up* rule.

The *nominal rolling up* rule works in the same way (Figure 7.4); but in this case the uniqueness is guaranteed by the equality term. Note that the basic rolling up procedure cannot be used directly in this case because there is an equality term (i.e. $y = o$). To cover these cases we use the representative concept of the individual name involved. As introduced in Section 6.2.1, the representative concept is used in place of the DL `one-of` constructor. In fact, using the `one-of` constructor, the terms $\{y = o, \langle x, y\rangle{:}R_1 \sqcap \dots \sqcap R_n, y{:}C\}$ (when y is a leaf node) can be substituted by the single term assertion $x{:}(\exists R_1.(C \sqcap \{\ o\ \}) \sqcap \exists R_2.\{\ o\ \} \sqcap \dots \sqcap \exists R_n.\{\ o\ \})$. The *nominal rolling up* rule uses the concept P_o in place of the concept expression $\{\ o\ \}$. In addition, an assertion which ensure that the individual is an instance of its representative concept is added to the knowledge base (e.g., $o{:}P_o$) if it is not already there.

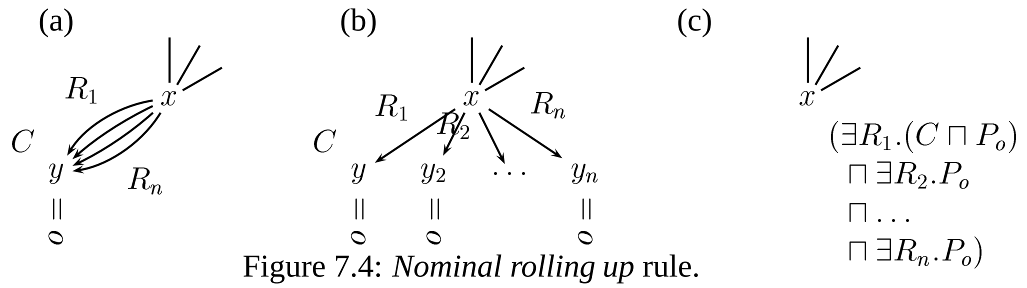


Figure 7.4: *Nominal rolling up* rule.

Since our underlying language does not allow the use of the inverse role constructor, all the rolling up rules have their inverse counterpart. This is necessary because we cannot guarantee that the graph can be collapsed by following the direction of the edges. In fact, is quite easy to come up with an example of graph in which all nodes have both entering and exiting edges. The idea is not dissimilar to the one already exploited for the representative concept: the introduction of a new concept representing the existance of the appropriate edge and node.

Consider the basic *simple inverse rolling up* rule for a disjunct having the terms $\langle y, x\rangle{:}R, y{:}C$ (node y is a leaf). If the underlying DL logic provided the inverse role constructor, the pair of term could have been substituted by the single concept term $x{:}\exists R^{-1}.C$ in the same way as the *simple rolling up* rule. In order to overcome the deficiency of the logic we are going to substitute the expression with a new concept name ($P_{R^-.C}$). The technique is similar to the one used for reducing the satisfiability problem of Converse PDL to the very same problem in PDL (see De Giacomo [1996]). In our case we are not interested in enriching the language with the inverse role constructor, but only in representing a particular "shape" in the interpretations. Therefore, we can localise the scope of the newly introduced concept name to the given expression C by using the axiom $C \sqsubseteq \forall R.P_{R^-.C}$ (see Proposition 7.18).

The *nominal inverse rolling up* rule is essentially a combination of the *simple inverse rolling up* rule, combined with the duplicating arguments we already presented in the case of the *nominal rolling up* rule. The *functional inverse rolling up* rule takes a different approach, exploiting the functional restriction on the role names. Consider the transformation shown in Figure 7.5, which is exactly the same pattern as seen in the *functional rolling up* rule (see Figure 7.3).

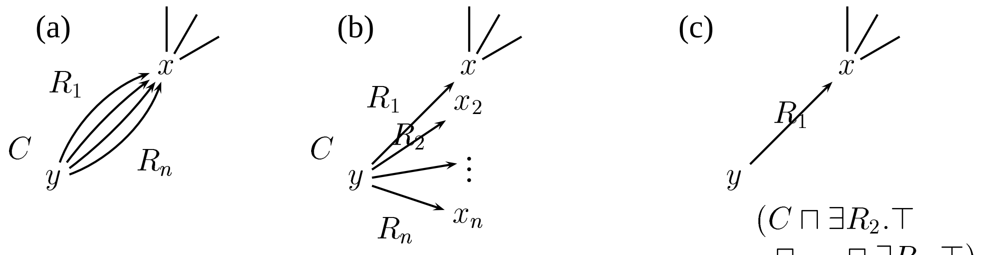


Figure 7.5: *Functional inverse rolling up* rule.

In this case we duplicate the variable x; the new nodes $x_2, \ldots, x_n$ can then be rolled up by using the same arguments as the *simple rolling up* rule. The resulting structure shown in Figure 7.5 (c) is then suitable for the application of a *simple inverse rolling up* rule in a subsequent step.

The rolling up rules eliminate nodes and edges from the graph, but we should make sure that the two conditions, maintaining the joining variable z and maintaining the connectedness, are still satisfied by the resulting graph. The second condition is guaranteed by the fact that rules only operate on leaves, while the first one is explicitly stated on each rule (with the condition $y \not\equiv z$). However, this latter requirement for the rules clashes with the necessity of eliminating the equality terms as well as the role terms. In fact, equalities with individual names are eliminated by the rolling up procedure (by the *nominal rolling up* and *nominal inverse rolling up* rules; but they never apply to the joining variable z. For this reason we have the *nominal elimination* rule which applies to z only. Note that this rule must be applied only when there are no role terms in the conjunctive formula, otherwise the fact that z must be identified with a particular individual name would be lost.

Cycles in query formulae

Until this point we have shown the rules which handle tree–like structures; in fact, when the graph is a tree we can simply roll it up starting from the leaves.[11] When the query graph contains cycles we need a means for "breaking" these cycles and rolling up the graph using the rules described above. This means is provided by the cycle breaking rules toghether with the nominal introduction rules.

Let us first examine the three cycle breaking rules, starting from the *cycle breaking* and *inverse cycle breaking* rules. They work using the very same principle which has been exploited for the nominal rolling up rules. The fact that the variable y is identified with an individual name allows us to introduce a new variable forced to have the very same interpretation as y (because the new variable is identified with the very same individual name). In addition, the path still connecting the two variables guarantees the connectedness of the resulting graph (see Figure 7.6). The *inverse cycle breaking* rule is similar, the only difference being the direction of the edges connecting the two variables. The two rules also cover the case in which the two variables coincide; in that case the graph contains a loop centered on the variable.

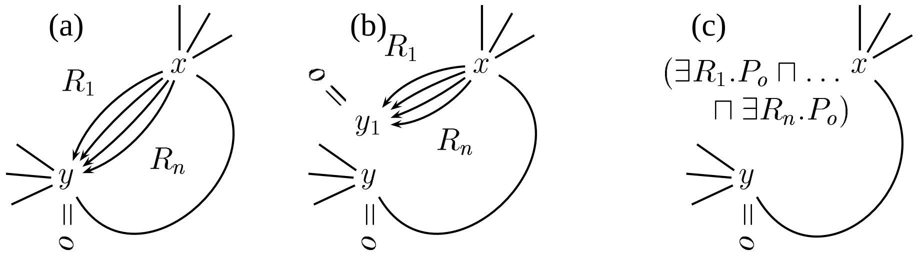


Figure 7.6: *Cycle breaking* rule.

The *simple cycle breaking* rule is similar, in principle, to the *cycle breaking* rule; but it does not assume that removing the edges will leave the graph connected; therefore,

[11] This is not completely precise because, even in tree–like graphs, we may still need to use the *simple nominal introduction* rule.

one of the role names (R_1) is left unchanged in order to maintain the connectedness. At a first sight, *simple cycle breaking* looks redundant because of the *nominal rolling up* rule. However, the rule is necessary when the rolling up is contrary to the direction of the edges (for example when $y \equiv z$, or the leaf node is x) because the *nominal inverse rolling up* rule covers only cases in which the starting node is identified with an individual name.

The rule is part of the ones handling cycles because, although the role term (i.e. $\langle x, y\rangle{:}\sqcap\{R_1, \ldots, R_n\}$) is not necessarily part of a cycle (according to Definition 7.6) when role names appearing in a role term are unrelated (see the role labels in Definition 3.4), the term itself can be considered a sort of a cycle. In fact, we can traverse one of the edges in one direction and a different edge in the other direction, coming back to the very same variable. This is not true for the "false cycles" as explained in the normalisation rules.

In the query there can be cycles (even the "cycles" composed by a single term) composed by variables only; i.e. in which none of the variables are identified with an individual name by an equality term. In these cases we cannot use the cycle breaking rules, and one of the nominal introduction rules must be used.

These two rules exploit the completeness of the quasi transitive shrub interpretations (Section 3.1), since the properties of these interpretations guarantee that cycles can only be enforced by assertions about the individual names. Consider the *simple nominal introduction* rule applied to a term $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$. The idea consists in identifying the second variable (y) with one of the individual names. The choice cannot simply be nondeterministic, as shown in Example 7.3; therefore we must be able to try all the possibilities at once. This is the reason why maintaining a connecting variable (the z variable in Section 7.3.1) across all the disjuncts is essential for the completeness of the algorithm.

The problem with the nondeterministic approach is that once the (nondeterministic) choice is made, it must be the correct choice for all the possible interpretations satisfying the knowledge base. In fact, we are going to show an example of knowledge base in which for some interpretations a variable must be identified with one individual, while for others a different individual must be chosen. Trying all the possibilities at once allows all the different cases to be covered.

Example 7.3

Let us consider the knowledge base

$$\Sigma = \{\langle a, b\rangle{:}R, \langle b, a\rangle{:}R, \langle a, b\rangle{:}S, \langle b, a\rangle{:}S, a{:}(A \sqcup \forall S.A)\}\,.$$

Σ forces two different cycles with the same role names:

$$\{\langle a, b\rangle{:}R, \langle b, a\rangle{:}S\}$$

and

$$\{\langle b, a\rangle{:}R, \langle a, b\rangle{:}S\}\,;$$

in addition, the assertion $a{:}(A \sqcup \forall S.A)$ forces either a or b to be an instance of A in every model. Therefore the query formula $\exists xy(x{:}A \wedge \langle x, y\rangle{:}R \wedge \langle y, x\rangle{:}S)$ is a logical consequence of Σ. On the other hand, we cannot nondeterministically choose one of the two individual names to be identified with x (or y). In fact, in this way all the possible choices will lead to query formulae which are not logical consequences of Σ.

When a true cycle is present in the query, the *cyclic nominal introduction* rule is used; its rationale relies on the very same arguments as for the *simple nominal introduction* rule. However, in this case we must be more careful in order to avoid a different kind of "false cycle", as shown in the next example.

Example 7.4
Let us consider the formula $\phi = \{\langle z, x_1\rangle{:}R, \langle z, x_2\rangle{:}R, \langle x_1, y\rangle{:}S, \langle x_2, y\rangle{:}S\}$ having a diamond shape:

$$\begin{tikzcd}
 & x_2 \arrow[dr, "S"] & \\
z \arrow[ur, "R"] \arrow[dr, "R"'] & & y \\
 & x_1 \arrow[ur, "S"'] &
\end{tikzcd}$$

and the knowledge base $\Sigma = \{a{:}\exists R'.(\exists R.(\exists S.\top))\}$. By interpreting both variables x_1 and x_2 as the very same element, it is easy to see that $\Sigma \models \{\phi\}$. In addition, the graph is cyclic and there are no applicable rules other than the *cyclic nominal introduction* one.

Suppose that we are going to apply the rule to the variable z (choosing a different variable would not make any real difference), without adding all the possible combinations of equality terms among the variables in the query. In this case we obtain the new formula

$$\phi' = \{z = a, \langle z, x_1\rangle{:}R, \langle z, x_2\rangle{:}R, \langle x_1, y\rangle{:}S, \langle x_2, y\rangle{:}S\}\,,$$

because a is the only individual name in Σ. It is easy to see that $\{\phi'\}$ is not a logical consequence of Σ (i.e. $\Sigma \not\models \phi'$).

Applying the rule as it is defined, we obtain several disjuncts and the formula $\{x_1 = x_2, \langle z, x_1\rangle{:}R, \langle z, x_2\rangle{:}R, \langle x_1, y\rangle{:}S, \langle x_2, y\rangle{:}S\}$ among them. This disjunct is simplified into the formula $\{\langle z, x_2\rangle{:}R, \langle x_2, y\rangle{:}S\}$ by the *equality elimination* rule, and subsequently rolled up into the formula $\phi'' = \{z{:}(\exists R.(\exists S.\top))\}$. It is easy to see that the query $\{\ldots, \phi'', \ldots\}$ is a logical consequence of Σ.

As shown by the example above, the cases of false cycles are covered by the introduction of the equalities in the *cyclic nominal introduction* rule. If two (or more) of the variable names in a cycle must be interpreted as the very some element, there is at least one of the disjuncts which has the required equality terms explicitly stated.

7.4.3 Limitation of the technique

Unfortunately the presented technique works only under some restrictions on the form of the queries and/or terminologies. We have two main sources of problems, namely the transitivity restrictions and the interaction between functionality and the role hierarchy. This is symptomatic of the fact that the restriction on role interpretation complicates the structure of the models as shown in Chapter 3. In the following two section we are going to describe the limitations of the query answering algorithm.

Functionality and hierarchy

As highlighted by the Definition 3.6 of a canonical interpretation, we can assume that if a pair is in the interpretation of two different roles these two roles must be in a common label (let us ignore Abox assertions). However, the contrary is not necessarly true; i.e. interpretations of roles do not have to coincide if two roles are in the very same label.

For the application of the *simple nominal introduction* rule it is essential to know whether a variable must be identified with an individual in the presence of two variables connected by more than one edge (i.e. the node y with a term like $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$). Trivial cases are covered by the *role elimination* rule, since the inclusion relation between two roles guarantees that every pair in the included role must be in the including role as well. However, the interaction between functional restrictions and role hierarchy may cause more subtle “terminological” coreferences.

For example, in the case in which two unrelated (via $\preceq$) roles have a common functional super–role. Let R_1 and R_2 be the two roles both included in a functional role F; in an arbitrary interpretation $\mathcal{I}$ satisfying the restrictions on the roles, whenever there is a pair (u, v) in ${R_1}^{\mathcal{I}}$ and a pair (u, w) in ${R_2}^{\mathcal{I}}$ then $v = w$ because they must be both in the interpretation of F. Note that this is different from saying that whenever a pair is in one of the two roles it must be in the second as well (as in the case if inclusion). This property allows us to “roll up” terms like $\langle x, y\rangle{:}R_1 \sqcap R_2$ into something like $x{:}(\exists R_1.\top \sqcap \exists R_2.\top)$ without loosing the information that the two successors must be the same element.

The cases shown until now have in common the fact that the “terminological” coreference can be discovered by looking at the relation among the role names. Unfortunately there are cases in which the role structure is not enough and the single interpretation makes the difference.

Example 7.5
Let F_1, F_2 be two unrelated functional roles and R, S, T three different roles s.t. $R \preceq F_1$, $S \preceq F_1$, $S \preceq F_2$, and $T \preceq F_2$ (assuming that the the converse elations do not hold). Given an arbitrary interpretation $\mathcal{I}$, and two pairs (u, v) in $R^{\mathcal{I}}$ and (u, w) in $T^{\mathcal{I}}$, whether or not v and w must be the same actually depends on the existance of an S–successor of u.

If there is an element u' s.t. $(u, u') \in S^{\mathcal{I}}$, then by using the same arguments as in the previous case we can show that $v = u'$ and $u' = w$; therefore $v = w$. Otherwise, nothing is forcing v and w to be the same. Here we have a “terminological” coreference that does not depend on the roles structure alone.

How this affects the presented algorithm can be shown by a simple example. Let us consider the following three Aboxes sharing the same role terminology described above

$$\{\langle a, b\rangle{:}R, a{:}\exists T.\top, \langle a, b\rangle{:}T\} \tag{7.5a}$$

$$\{\langle a, b\rangle{:}R, a{:}\exists T.\top, a{:}\exists S.\top\} \tag{7.5b}$$

$$\{\langle a, b\rangle{:}R, a{:}\exists T.\top\}\,. \tag{7.5c}$$

We want to decide whether the formula $\langle a, x\rangle{:}R \sqcap T$ is a logical consequence of the assertions.[12]

It is easy to realise that the given formula is a logical consequence of both the first two Aboxes (7.5a) and (7.5b), but not of the third one. In fact, an interpretation $\mathcal{I}$ where $\{(1,2)\} = R^{\mathcal{I}} = {F_1}^{\mathcal{I}}$, $\{(1,3)\} = T^{\mathcal{I}} = {F_2}^{\mathcal{I}}$, $a^{\mathcal{I}} = 1$, and $b^{\mathcal{I}} = 2$), satisfies the Abox assertions in (7.5c) but not the formula $\exists x(\langle a, x\rangle{:}R \sqcap T)$.

The problem in devising an algorithm to discern the different cases comes down to figuring out whether the variable x should or should not be identified with one of the individuals of the Abox. For the Abox (7.5a) we can use a transformation rule like *simple nominal introduction* which adds, (amongst others) the disjunct $\{\langle a, x\rangle{:}R \sqcap T, x = b\}$. However, this is not sufficient to detect the case of the Abox (7.5b); this last case can be covered by a rule like *functional rolling up*, which transforms the formula into $a{:}(\exists R.\top \sqcap \exists T.\top)$.

On the other end, if the latter transformation is indiscriminatevily used, the Abox (7.5c) would be wrongly recognised as satisfying $\langle a, x\rangle{:}R \sqcap T$. In this particular case, the right trasformation should be something that takes into account the necessity of an S–successor for a, like the term $a{:}(\exists R.\top \sqcap \exists T.\top \sqcap \exists S.\top)$.

The last example suggests that a combination of the two mentioned rules can be used to detect these tricky cases of interaction between role hierarcy and functional restrictions. However, we still have to generalise this intuition to the general case, and for the time being we are forced to restrict the expressivity of the role terminology.

Roughly speaking, we forbid cases like the described one by ensuring that for any pair of roles in a single label either they are included one in the other, or they share a common functional super–role. As shown in the proof of Proposition 7.14, this restriction is enough to prove correctness and completeness of the functional rolling up rules.

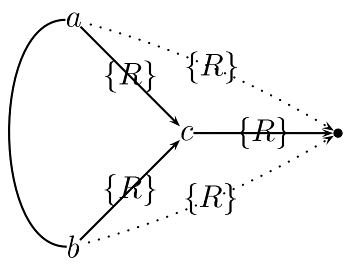
Transitivity and cycles

The kind of queries we are able to answer using this technique must not contain any cycle which includes a transitive role name.[13] The problem with cycles including transitive roles is that there may be variables in the cycle that are not identified with one of the individual names in the knowledge base. The reason is that transitivity provides a way of creating shortcuts for arbitrarily long directed paths. Let us consider for

[12]With an abuse of notation we used the individual a instead of a variable for simplifying the formula.

[13]In fact, the requirement can be less restrictive, as highlighted by the proof of Proposition 7.16. However, this formulation is much simpler and clearer.

example a knowledge base forcing interpretations having a structure like:



where a, b, c represent elements of the domain corresponding to individual names and the dot an “anonymous” element. When the role name R is transitive the dotted shortcuts are presents whenever the non–dotted structure is present (for example by means of an assertion like $c{:}\exists R.\top$). Therefore there is always a cycle which includes an element not corresponding to an individual name (the “anonymous” element). The scenario can be complicated by the role hierarchy, since these transitive shortcuts can be forced into non trasitive roles as well.[14]

7.4.4 Example of query answering

For a better undersanding of the algorithm we show the rule application process on a concrete example. Let $\Sigma = \{R \sqsubseteq F, \langle a, o\rangle{:}R, \langle a, b\rangle{:}S, a{:}A, b{:}\exists S.\top\}$ be a knowledge base and we like to know whether the answer to the boolean query

$$\exists zxy(z{:}A \wedge \langle z, x\rangle{:}R \sqcap F \wedge x = o \wedge \langle z, y\rangle{:}S \wedge \langle y, y\rangle{:}S)$$

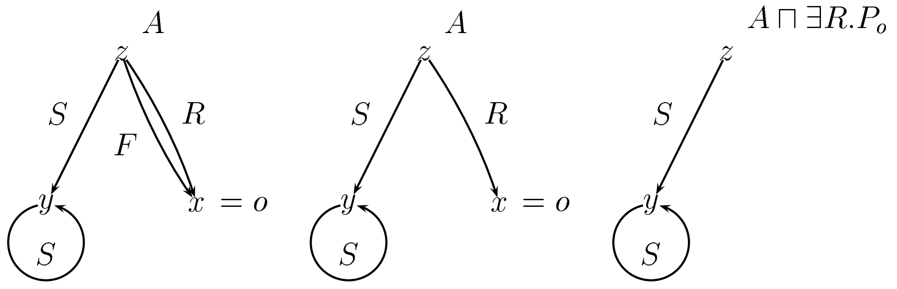
is positive or negative. We can guess that the answer must be negative (i.e. Σ does not imply the given formula). The reason for this is that the constraint $b{:}\exists S.\top$ does not force a loop on b but only the existence of a element related with b by role S (it may be b itself, a, o, or even a fourth one not specified).

Since the query formula is conjuntive and connected, it is already in normal form. We choose z as the joining variable:

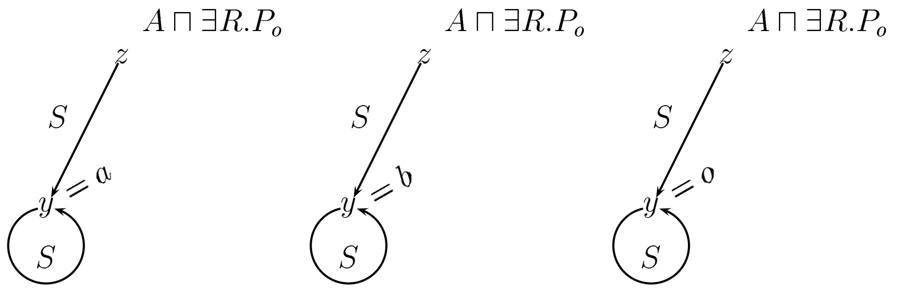
$$\{\{z{:}A, \langle z, x\rangle{:}R \sqcap F, x = o, \langle z, y\rangle{:}S, \langle y, y\rangle{:}S\}\} \mid_z.$$

[14]We have the conjecture that the problem can be solved by introducing more disjuncts with the *cyclic nominal introduction* rule. The idea is that, since the underlying DL does not provide the inverse role constructor, at least one of the variables in a cycle must be coreferenced with one of the individual names in the KB. However, this hypothesis is still being investigated.

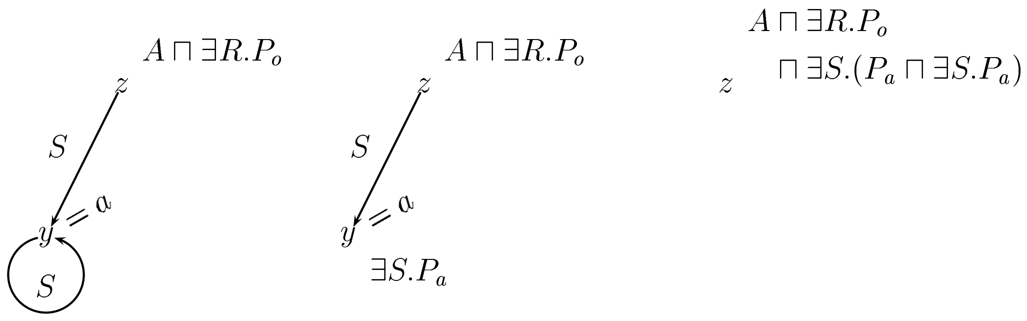
To simplify the exposition we start from a knowledge base Σ' already containing the assertions for the representative concepts (i.e. $\Sigma' = \Sigma \cup \{a{:}P_a, o{:}P_o, b{:}P_b\}$). In this way the knowledge base will be unmodified by the rules we are going to use. Initally the query formula contains a single graph, to which are applied first the *role elimination*, followed by the *nominal rolling up* rules:



At this point we can only apply the *cyclic nominal introduction* rule to the loop involving the variable y. The rule generates three different isomorphic graphs where the variable y is identified with each one of the three different individual names in the knowledge base. The query now is composed of the three graphs:



Since the graphs have the same shape, they are transformed in the very same way; with the only difference of the representative concept used for the variable y.First, the *cycle breaking* rule is applied to the loop on y, then the *nominal rolling up* rule concludes the collapsing:



The query formula we are now going to verify is composed by three disjuncts in the form of $\{z{:}(A \sqcap \exists R.P_o \sqcap \exists S.(P' \sqcap \exists S.P'))\}$; where P' is substituted by P_a, P_b, or P_o. Since the variable z is the same we can collapse the disjunction into a single concept expression by using the $\sqcup$ constructor.

$$\begin{aligned} z{:}(&(A \sqcap \exists R.P_o \sqcap \exists S.(P_a \sqcap \exists S.P_a)) \\ &\sqcup (A \sqcap \exists R.P_o \sqcap \exists S.(P_b \sqcap \exists S.P_b)) \\ &\sqcup (A \sqcap \exists R.P_o \sqcap \exists S.(P_o \sqcap \exists S.P_o))) \end{aligned}$$

By using simple transformations on the concept expressions we obtain the query answering problem

$$\Sigma' \models \exists z(z{:}(A \sqcap \exists R.P_o \sqcap (\exists S.(P_a \sqcap \exists S.P_a) \sqcup \\ \exists S.(P_b \sqcap \exists S.P_b) \sqcup \exists S.(P_o \sqcap \exists S.P_o)))),$$

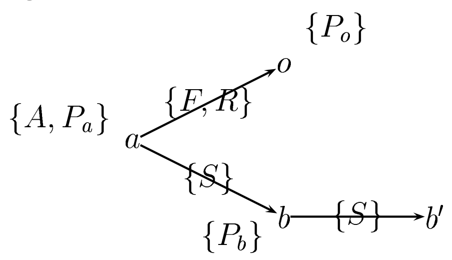
which can be verified by checking whether the knowledge base

$$\Sigma' \cup \{\top \sqsubseteq \neg(A \sqcap \exists R.P_o \sqcap (\exists S.(P_a \sqcap \exists S.P_a) \sqcup \\ \exists S.(P_b \sqcap \exists S.P_b) \sqcup \exists S.(P_o \sqcap \exists S.P_o)))\}$$

is unsatisfiable. We can push the negation down into the concept expression to obtain the kb

$$\Sigma' \cup \{\top \sqsubseteq (\neg A \sqcup \forall R.\neg P_o \sqcup \\ \forall S.((\neg P_a \sqcup \forall S.\neg P_a) \sqcap (\neg P_b \sqcup \forall S.\neg P_b) \sqcap (\neg P_o \sqcup \forall S.\neg P_o)))\}$$

This kb is indeed satisfiable, and this can be verified by using a KB satisfiability algorithm. However, here we demonstrate its satisfiability by exhibiting an example of interpretation satisfying the constraints. Let us consider the following interpretation $\mathcal{I}$:



Interpretation $\mathcal{I}$ has been constructed in such a way that it satisfies all the constraints in

Σ, and all the assertions for the representative concepts. To satisfy the added inclusion assertion derived from the query formula, every element of the domain must be in the interpretation of the concept. The elements o, b, and b' satisfy the constraint because they are not in $A^{\mathcal{I}}$ (therefore they are in $(\neg A)^{\mathcal{I}}$). The element a obviously is not in $(\neg A)^{\mathcal{I}}$, nor in $(\forall R.\neg P_o)^{\mathcal{I}}$; therefore we should verify that b is in $((\neg P_a \sqcup \forall S.\neg P_a) \sqcap (\neg P_b \sqcup \forall S.\neg P_b) \sqcap (\neg P_o \sqcup \forall S.\neg P_o))^{\mathcal{I}}$. Clearly, this is the case because it is in $(\neg P_a)^{\mathcal{I}}$, in $(\neg P_o)^{\mathcal{I}}$, and in $(\forall S.\neg P_b)^{\mathcal{I}}$.[15]

Since the transformed KB is satisfiable, the conclusion is that the query formula is not a logical consequence of the KB Σ.

7.4.5 Termination

Transformation rules are applied by following a simple "ordering" strategy: if more than a rule can be applied to the same disjunct, then the first one (in the order they have been presented) is applied.

Given this strategy the rule application process terminates; i.e. the transformation always leads to a formula where no rules are applicable, and the resulting formula has the form $\Sigma \models \{\{z{:}C_1\}, \ldots, \{z{:}C_n\}\} \,|_z$.

The proof of termination is divided in three parts. First we show that the application of any rule maintains the connectivity of the query formula. Secondly, we show that the variable z is never eliminated; finally, we prove that in a finite number of steps we are going to obtain a query formula without role terms. If an element (disjunct) of the query is connected, contains the variable z, and does not contain any role term it must be in the form $\{z{:}C_1, \ldots, z{:}C_n\}$ (possibly containing a term in the form $z = o$ as well). From that is collapsed into $\{z{:}C\}$ by the *conjunction elimination* rule (and *nominal elimination* rule if $z = o$ is present).

Proposition 7.8. *Let* $\Sigma \models \{\phi_1, \ldots, \phi_n\} \,|_z$ *be a query where all the elements* $\phi_1, \ldots, \phi_n$ *are connected, and* $\Sigma' \models \{\phi'_1, \ldots, \phi'_{n'}\} \,|_z$ *be the resulting query after the application of one of the rules. Then all the disjuncts* $\phi'_1, \ldots, \phi'_{n'}$ *are connected.*

Proof. Given the assumption that each one of the starting disjuncts is connected, we have to show that the connectivity is maintained by the rules which remove role terms.

[15]Note that, the interpretation $\mathcal{I}$ provides also an example of interpretation satisfying Σ, but not modelling the query formula. Therefore an alternative way of showing that the formula is not a logical consequence of the KB Σ.

- **Equality elimination**: even if all the occurrences of variable x are renamed to y, the disjunct is still connected because no role terms are removed.
- **Role collapsing**: two role terms are eliminated between x_1 and x_2, but a new role term is inserted connecting the very same variables. Therefore if the disjunct was connected before the rule application, then it is still connected after the application.
- **Contradiction elimination**: the new disjunct $\phi' = \{z{:}\bot\}$ is trivially connected.
- **Role elimination**: since $n > 1$, then the role term is substituted with a different role term between the very same variables.
- **Shortcut elimination**: even if a role term between variables x_0 and x_n is removed, there still is a path connecting the two of them by hypothesis.
- **Simple rolling up**: the term $\langle x, y\rangle{:}R$ is removed from ϕ; but since the variable y no longer appears in ϕ', then if the disjunct ϕ is connected, so must be the new ϕ'.
- **Simple inverse rolling up**: this case is analogous to the previous one.
- **Functional rolling up**: again, this case is analogous to the *simple rolling up*.
- **Functional inverse rolling up**: a role term connecting variables y and x is removed from ϕ; but a new role term connecting the very same variables is added.
- **Nominal rolling up**: this case is analogous to *simple rolling up*.
- **Nominal inverse rolling up**: this case is analogous to *simple rolling up*.
- **Cycle breaking**, **inverse cycle breaking**: a role term connecting x and y is removed from ϕ; however, in the remaining formula there is a path connecting x and y by hypothesis (or $y \equiv x$). Therefore ϕ' is still connected.
- **Simple cycle breaking**: a role term connecting x and y still remains in ϕ', so the connectedness is maintained.

$\square$

Proposition 7.9. *Let $\Sigma \models \{\phi_1, \dots, \phi_n\} |_z$ be a query answering problem where the variable z appears in all the disjuncts ϕ_1, and $\Sigma' \models \{\phi'_1, \dots, \phi'_{n'}\} |_z$ be the resulting query after the application of one of the rules. Then the variable z appears in all the disjuncts ϕ'_i.*

Proof. Analogously to the previous proposition we consider the rules that might delete terms.

- **Equality elimination**: z is never renamed by definition.
- **Conjunction elimination**: the variable x is still in ϕ' because of the fact that a new concept term is added.
- **Role collapsing**: both x_1 and x_2 are still kept in ϕ'.
- **Nominal elimination**: this rule does not affect z by definition.
- **Contradiction elimination**: variable z is still in ϕ' by definition.
- **Nominal elimination**: a concept term containing z is still in the disjunct ϕ'.
- **Role elimination**, **shortcut elimination**: both x_1 (x_0) and x_2 (x_n) are still in ϕ'.
- **Simple rolling up**, **simple inverse rolling up**, **functional rolling up**, **functional inverse rolling up**, **nominal inverse rolling up**, and **nominal rolling up**: these rules do not remove z by definition.
- **Cycle breaking**, **simple cycle breaking**, and **inverse cycle breaking**: even thought a role term containing y is removed, in ϕ' there is at least an equality term containing y.

□

Proposition 7.10. *Let $\Sigma \models \{\phi_1, \dots, \phi_n\} |_z$ be a query where all the disjuncts ϕ_1 are connected, and z is appearing in all of them. Then, there is a strategy for applying the rules such that a new query $\Sigma' \models \{\{z{:}C_1\}, \dots, \{z{:}C_k\}\} |_z$ containing only concept terms can be obtained in a finite number of application of the rules.*

Proof. The idea behind the proof is showing that we only eliminate role terms by using the rules, but never introduce new ones; therefore we obtain a query without role terms. Moreover each disjunct is connected and contains the variable z as shown in the previous propositions; therefore, if the query does not contain any role term, then each disjunct must contain only terms like $z = o$ or $z{:}C$. Elements containing only concept terms can be normalised into the form $\{z{:}C'\}$ by using the *nominal elimination* and *conjunction elimination* rules.

Given a set of terms ϕ, we define the *number of edges* of ϕ as the total number of different role names connecting each pair of variable names in role terms. Formally, this can be defined as the cardinality of the set

$$\{\langle x, y\rangle{:}R \mid \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n \in \phi, R = R_i\}\,.$$

Note that the rules never increase this number per disjunct, at most they multiply the number of disjuncts in the query (the nominal introduction rules). Given a query formula $\{\phi_1, \ldots, \phi_n\}\,|_z$, we consider the number of edges of the formula as the maximum among the number of edges of each disjunct ϕ_i. We are going to show that this number decrease in a finite number of applications of the rules.

A given disjunct in the query can be transformed (by the substitutions performed by the rules) into a form in which no rules are applicable or only the nominal introduction rules are applicable, in a number of steps which is linear w.r.t. the size of the disjunct itself.

Let us consider the query answering problem $\Sigma \models \{\phi_1, \ldots, \phi_n\}\,|_z$; we can assume that none of the normalisation rules can be applied to any of the ϕ_i (if it is not the case this can be achieved in a number of steps linear w.r.t. the sum $\sum_{i=1,\ldots,n} \sharp\phi_i$, where $\sharp\phi_i$ denotes the cardinality of the set ϕ_i). If the number of edges of $\{\phi_1, \ldots, \phi_n\}\,|_z$ is greater than 0, there should be a subset of the disjuncts having that number of edges. We are going to take one of those disjuncts and show that the number of edges of disjuncts "derived" from the chosen one can be decreased in a finite number of steps. In addition we are going to show that we can limit the number of "derivable" disjuncts. We can apply this arguments to all the "maximal" disjuncts in the original query; therefore in a finite number of steps we obtain a new query answering problem with a smaller number of edges.

First we must specify precisely what we intend by the fact that a disjunct is "derived" from another one in the query. As we described in Section 7.4.1, most of the rules take one of the disjuncts ϕ from the query and replace it with a modified version ϕ' of it; we say that ϕ' is "derived" from ϕ. The case of nominal introduction rules is different; the disjunct ϕ is substituted by a finite number of new disjuncts, and all these are said to be derived from ϕ.

Let ϕ be one of the disjuncts in the query $\{\phi_1, \ldots, \phi_n\} \mid_z$. Clearly all the derived disjuncts have a number of edges which is less or equal than the one of ϕ. We assumed that none of the normalisation rules is applicable; in addition none of the role elimination rules is applicable as well, otherwise we already obtained a derived disjunct with a smaller number of edges.

First we must show that, under these assumptions, when there is at least a role term in ϕ one of the nominal introduction rules is applicable. We distinguish two cases according to the fact that there is or there is not any cycle in ϕ.

- If there is no cycle in ϕ, then there is a role term $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ being the last term of a path not terminating with z; i.e. either there is not any different role term containing y and $y \not\equiv z$, or there is not any different role term containing x and $x \not\equiv z$.

  Let us assume that in ϕ there is not any different role term containing $y \not\equiv z$. Since none of the normalisation rule are applicable, there can be only two other terms containing y: $y = o$ for some individual name o and $y{:}C$. If $y = o$ is in ϕ, then the *nominal rolling up* would be applicable; which is not the case by assumption, therefore $y = o$ is not in ϕ. If $l = 1$ (i.e. $\langle x, y\rangle{:}R_1$), then the *simple rolling up* would be applicable, so $l > 1$. Finally, there is not any label L s.t. $\{R_1, \ldots, R_n\} \subseteq L$, otherwise the *functional rolling up* rule would be applicable. Therefore the *simple nominal introduction* rile is applicable because all the preconditions are met.

  Let us assume that in ϕ there is not any different role term containing $x \not\equiv z$. As before we can exclude the case in which $l = 1$ and all the roles are included in a single label (otherwise the *simple inverse rolling up* or the *Functional inverse rolling up* rule would be applicable). In the same way we should exclude the case in which either a term like $x = o$ or $y = o'$ are in ϕ, otherwise the *inverse cycle breaking* or *simple cycle breaking* rule would be applicable. Therefore the *simple nominal introduction* rule is again applicable.

- If there are cycles in ϕ, let us consider each role term $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ involved in at least a cycle. Since neither the *cycle breaking* nor the *inverse cycle breaking* rules are applicable, then there are no terms like $x = o$ or $y = o$ in ϕ; therefore the *cyclic nominal introduction* rule is applicable.

Now we consider the cases in which the nominal introduction rules must be applied. We show that in a finite number of steps a new query is obtained where all the disjuncts derived from ϕ have a number of edges less than that of ϕ (i.e. a rule which eliminates edges is applied). We proceed by induction on the number of variables, not identified with any individual name (by a term like $x = o$), which are involved in at least a cycle or in a term like $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ where $\{R_1, \ldots, R_l\}$ are not included in a single label.

- First let us consider the case in which there is only one variable y satisfying the above conditions. Then, either the *simple nominal introduction* rule is applicable or there is a loop on y and the *cyclic nominal introduction* is applied. In both cases the disjuncts introduced in place of ϕ are of the form $\phi \cup \{y = o_i\}$ (because adding an equality term between the same variable does not make any sense). In the case of the *simple nominal introduction* rule either the *nominal rolling up* or *simple cycle breaking* rules are then applicable. If the *cyclic nominal introduction* rule has been applied, then there is the possibility of applying the *cycle breaking* rule as well. Anyway, in all three cases the rules eliminate one or more edge from the derived disjunct $\phi \cup \{y = o_i\}$.

- If there is no cycle, then there must be a term like $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$, and we can apply the very same arguments we used in the basic case. So let us assume that there is a cycle involving more than one variable (not identified with any individual name) and one of the nominal introduction rules is applied to one of these variables y. We consider the two cases separately.

  Simple nominal introduction Let us consider one of the disjuncts $\phi' = \phi \cup \{y = o_i\}$ inserted in place of ϕ. Let $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ be the role term matching the rule application. In this case the *nominal rolling up* rule is not applicable because y is part of a cycle, but one of the *cycle breaking* or *simple cycle breaking* rules can be applied to any of the resulting disjuncts, diminushing their number of edges.

  Cyclic nominal introduction Since the cycle is longer than 1, either there is a role term $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ or $\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ in the path, with $x \not\equiv y$. In addition, the derived disjuncts are either in the form $\phi \cup \{y = o_i\}$ or $eq^{(i)} \cup \phi$, where $eq^{(i)}$ is a set of equalities between variables.

  Let us consider the disjunct $\phi' = \phi \cup \{y = o_i\}$. As we mentioned, either $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ or $\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ are in the path.

  If $\langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ is in the path then the preconditions of the *cycle breaking* rule are satisfied (and none of the preceding ones); therefore in the next step the (unique) disjunct derived from ϕ' will have a smaller number of edges. Let us now consider the case in which $\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_l$ is in the path. The *cycle breaking* rule is not applicable because by assumption ϕ' does not contain any term like $x = o$, and the same is true for the *simple cycle breaking* rule. However, the *inverse cycle breaking* rule is applicable, which is a role elimination rule.

Let us consider a disjunct in the form $\phi' = eq^{(i)} \cup \phi$, where $eq^{(i)}$ is a set of equalities between variables. The query answering problem can be transformed into a new problem in which ϕ' is substituted by a single new disjunct at each step, by applying the normalisation or role elimination rules. In a number of steps which is linear w.r.t. the size of ϕ' we obtain a single disjunct ϕ'' derived from ϕ' to which only the nominal introduction rules are applicable (or none of the rules).

Note that the number of variables involved in a cycle in ϕ'' is decreased because of the fact that two of them have been explicitly made equal by a term $x_1 = x_2$ (so one of them has been eliminated by the *nominal elimination* rule). Therefore, we can use the inductive hypothesis and conclude that in a finite number of steps we obtain a formula in which all the disjuncts that are derived from ϕ'' have a number of edges smaller than ϕ' (and consequently than ϕ). The arguments are valid for all the disjuncts $\phi' = eq^{(i)} \cup \phi$ derived from ϕ.

□

Worst case complexity analysis

The complexity[16] of the actual rolling up procedure is polynomial in the size of the query formula. This can be seen by considering the graph corresponding to each disjunct; the rolling up eliminates the leaves of the graph (after a linear normalisation by the first five rules). Even if new assertions are added to the knowledge base, these are linearly bounded by the sum of the number of edges of each graph representing a disjunct in the query formula.

Unfortunately, the nominal introduction rules multiply the number of disjuncts (graphs) in the query. As the reader can guess from the last section, the number of

[16]In this section, we use the word complexity intending the worst case complexity.

newly introduced disjuncts can be exponential in the size of the initial problem.[17] The worst case is related to the *cyclical nominal introduction* rule, because the other rule introduce a lesser number of elements (the equalities are not introduced). Therefore, the worst case complexity can be calculated by analysing the *cyclical nominal introduction* rule.

The proof for termination (see Proposition 7.10) provides the hints for the analysis of the complexity. The point of the proof to look at is where the inductive arguments are used for the cases in which the *cyclical nominal introduction* rule can possibly be applied recursively. At each application of this rule one of the variables involved in a cycle is “eliminated” from the derived new disjuncts.[18] Therefore, if at the next step the *cyclical nominal introduction* rule is applied again, the number of equalities to be introduced will be smaller than the one at the previous step.

The complexity is of the order of the number of graphs that can be generated by recursive application of the nominal introduction rules. This number can be calculated by considering the tree whose root is a disjunct and the nodes the new elements generated by successive applications of the rule from the root element. The number of all the generated graphs is equal to the number of leaves in this tree; so we need to estimate the maximum number of successors for each node.

By looking at the rule we can estimate the number of successors of each node by considering the sum of the number of individual names in the knowledge base (we denote this number by $\sharp\mathcal{O}$), with the number of equalities that can be generated by the number of variables in the cycle (we can consider this number as the total number of variables). Although the number of variables can be tightened to a more precise lower value, we assume that it is equal to the size of the single disjunct (we denote this number as n). The number of equalities generated from n variables can be calculated by considering a square matrix of size n where each row (column) is marked by a variable. Each element of this matrix denotes an equality between the correspondent two variables. We do not need the whole matrix because we are not interested in the principal diagonal (terms like $x = x$), and we need only half of the matrix (there is not point in having both $x = y$ and $y = x$); therefore the number of equalities is $\frac{n(n-1)}{2}$.

The maximum number of successor of the root are $\sharp\mathcal{O} + \frac{n(n-1)}{2}$, at the next level we have $(\sharp\mathcal{O} + \frac{n(n-1)}{2})(\sharp\mathcal{O} + \frac{(n-1)(n-2)}{2})$, and so on until the number of variables is

[17] Unfortunately it is not only on the size of the formula because of the identification of the variables with the individual names in the knowledge base.

[18] Or it is identified with an individual, which is the same from the perspective of the *cyclical nominal introduction* rule.

equal to one:

$$\underbrace{(\sharp\mathcal{O} + \frac{n(n-1)}{2})(\sharp\mathcal{O} + \frac{(n-1)(n-2)}{2})\dots(\sharp\mathcal{O} + \frac{2(2-1)}{2})\sharp\mathcal{O}}_{n \text{ times.}}$$

Therefore, the number of disjuncts is bounded by the formula $(\sharp\mathcal{O} + n^2)^n$. The size of each disjunct is bounded by the size of the original formula, and in addition the rolling up procedure can add to the knowledge base as many new assertions as the number of inverse rolling up rule applications. This does not alter the order, which is still $O(N^N)$ w.r.t. the size of the query problem.

At the end of the collapsing procedure the algorithm uses the knowledge base satisfiability procedure whose complexity is EXPTIME (see Chapter 4). Note that we already estimated that the size of the input formula is of the order of $O(N^N)$, therefore the worst case complexity for the whole problem is double exponential time (2EXPTIME).

This is more than intractable, the question at this point is—can we do better than that? The answer can only be found by the reduction of a different problem known to be 2EXPTIME–hard to our query answering problem. However, there is good evidence that this is the intrinsic complexity of the problem, since it is the same order as the results presented in Calvanese et al. [1998a] for a similar problem. Although the DL presented in that paper is more expressive than the one we are presenting here, we already know that the complexity of the satisfiability problem is the same for both the logics. The reader may object that in the same work they do not show hardness results, however the nondeterministic nature of the problem suggests that the addition of an exponential step to the basic satisfiability problem cannot be avoided.

7.4.6 Correctness and completeness

The correctness and completeness of a rule are defined in the usual way. A rule is correct if a positive answer to the query problem is still positive for the new query problem generated by the rule itself; i.e. if $\Sigma \models \{\phi, \phi_1, \dots, \phi_n\}\mid_z$ then $\Sigma' \models \{\phi', \phi_1, \dots, \phi_n\}\mid_z$. A rule is complete if negative answers are preserved; i.e. if $\Sigma' \models \{\phi', \phi_1, \dots, \phi_n\}\mid_z$ then $\Sigma \models \{\phi, \phi_1, \dots, \phi_n\}\mid_z$).

For rules that do not modify the KB on the left hand side ($\Sigma = \Sigma'$) the pattern of the proof is “simple” because the set of interpretations satisfying the KB before and after the rule application does not change. On the other hand, when a rule modifies the KB the set of satisfying interpretations is reduced because new constraints are added to the KB. In this case we must pay attention to the introduction of new concept names; in particular regarding the representative concept. In fact, a representative concept is an effective way of representing a single element only if the reasoning mechanism is unable to distinguish the case where the interpretation of a representative concept contains more than one element from the case in which it contains a single element. Therefore we want to show a completeness result for the class of interpretations in which the representative concepts are restricted to contain a single element (this is analogous to the completeness of a class of models as seen in Section 7.2).

We call an interpretation *nominal representative* (n.r.) if the interpretation function maps each representative concept into the set containing only the interpretation of the corresponding individual name (i.e. ${P_o}^{\mathcal{I}} = \left\{o^{\mathcal{I}}\right\}$). Using the definition of n.r. interpretations we introduce the notion of *nominal safe* queries, as those completely characterised by n.r. interpretations. A query answering problem $\Sigma \models \varphi$ is *nominal safe* when $\Sigma \models \varphi$ w.r.t. n.r. interpretations implies that $\Sigma \models \varphi$ (note that the converse is trivially true).

We have to show that the manipulations of the query answering problem done by the rules keep the result nominal safe. Otherwise, the use of representative concepts will can produce wrong answers. The initial query answering problem is trivially nominal safe because no representative concepts appear neither in the knowledge base nor in query itself.[19]

Some of the proofs we provide rely on the fact that the query problem remains nominal safe after the application of any rule; this is an additional requirement to the completeness and correctness. Let $\Sigma \models \varphi$ be the query answering problem matching a rule, and $\Sigma' \models \varphi'$ the problem resulting from the application of the rule itself. We want to show that:

1. the rule is complete: $\Sigma' \models \varphi'$ implies $\Sigma \models \varphi$;
2. the rule is correct: $\Sigma \models \varphi$ implies $\Sigma' \models \varphi'$;
3. the resulting problem $\Sigma' \models \varphi'$ is nominal safe: $\Sigma' \models \varphi'$ w.r.t. n.r. interpretations implies $\Sigma' \models \varphi'$.

[19]An interpretation satisfying the KB can just be extend by adding the representative concepts, and mapping them to the set containing the element of the domain corresponding to the appropriate individual (i.e. starting from the interpretation $\mathcal{I}$, we make a new $\mathcal{I}'$ from $\mathcal{I}$ by adding $P_o^{\mathcal{I}'} = \left\{o^{\mathcal{I}}\right\}$ for every individual name o).

If we start from the assumption that $\Sigma \models \varphi$ is nominal safe, we can simplify the proofs by showing that the two conditions

$$\begin{aligned} &\Sigma' \models \varphi' \text{ w.r.t. n.r. interpretations implies} \\ &\qquad \Sigma \models \varphi \text{ w.r.t. n.r. interpretations, and} \end{aligned} \tag{7.3a}$$

$$\Sigma \models \varphi \text{ implies } \Sigma' \models \varphi' \tag{7.3b}$$

hold for every rule. Note that the second condition is just the correctness as defined above; the real difference is the first restricted version of completeness. Firstly, we show that the two conditions are sufficient when the problem $\Sigma \models \varphi$ is nominal safe.

1. **Completeness**: if $\Sigma' \models \varphi'$ then $\Sigma' \models \varphi'$ w.r.t. n.r. interpretations. We can then use the Condition (7.3a) to conclude that $\Sigma \models \varphi$ w.r.t. n.r. interpretations. In addition $\Sigma \models \varphi$ is nominal safe by hypothesis, therefore $\Sigma \models \varphi$ w.r.t. n.r. interpretations implies that $\Sigma \models \varphi$.

2. **Correctness**: it is exactly the Condition (7.3b).

3. **Safeness**: if $\Sigma' \models \varphi'$ w.r.t. n.r. interpretations, then $\Sigma \models \varphi$ w.r.t. n.r. interpretations by Condition (7.3a). In addition, $\Sigma \models \varphi$ w.r.t. n.r. interpretations implies that $\Sigma \models \varphi$ because the problem is nominal safe by assumption. Therefore, we can use Condition (7.3b) to conclude that $\Sigma' \models \varphi'$.

In the following propositions we assume that the query problem before the application of the rules is nominal safe, and by completeness we intend the Condition (7.3a) instead of the classical definition. The proofs are organised by rules, first the rules which do not modify the knowledge base, and then the ones which do modify it.

The pattern of all the proofs is similar, so we are going to introduce it to help the reader in following the proofs. The rules transform the problem $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ into the new problem $\Sigma' \models \{\phi'_1, \ldots, \phi'_k, \phi_1, \ldots, \phi_n\}|_z$, where the element ϕ is substituted by one or more elements $\phi'_1, \ldots, \phi'_k$. Let us consider completeness (correctness is similar), and a rule that substitute the element with a single new element.

We have to show that assuming $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$, it is the case that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We choose an arbitrary n.r. interpretation $\mathcal{I}$ satisfying Σ and we show that it models the formula $\{\phi, \phi_1, \ldots, \phi_n\}|_z$ (see Definition 7.3). If necessary for a rule modifying the kb, we first extend $\mathcal{I}$ to a new interpretation $\mathcal{I}'$ satisfying Σ'. In general, this new interpretation is equal to the original one with mappings for the newly introduced concept names; we can then easily go back to $\mathcal{I}$ by discarding the appropriate mapping. Since $\mathcal{I}'$ satisfies Σ', then $\mathcal{I}' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ by hypothesis; so there is a mapping ψ such that $\mathcal{I}' \models_\psi \phi'$ or $\mathcal{I}' \models_\psi \{\phi_1, \ldots, \phi_n\}|_z$. In the latter case, we have that $\mathcal{I}' \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$ as well, and in general it is easy to show that if $\mathcal{I}' \models_\psi \{\phi_1, \ldots, \phi_n\}|_z$, then $\mathcal{I} \models_\psi \{\phi_1, \ldots, \phi_n\}|_z$. This is because either the new names in $\mathcal{I}'$ do not appear in any of the formulae $\phi_1, \ldots, \phi_n$, or they are already in $\mathcal{I}$ (representative concept) and $\mathcal{I}' = \mathcal{I}$. Therefore in the actual proof we show that if $\mathcal{I}' \models_\psi \phi'$ then there is a mapping ψ' such that $\mathcal{I} \models_{\psi'} \phi$.

Rules that do not modify the KB

Proposition 7.11. *The* ***equality elimination****,* ***conjunction elimination****,* ***role collapsing****,* ***role elimination****, and* ***shortcut elimination*** *rules are complete and correct.*

Proof. Correctness and completeness of these rules are trivially verified by the fact that for every interpretation $\mathcal{I}$ and mapping ψ, $\mathcal{I} \models_\psi \phi$ iff $\mathcal{I} \models_\psi \phi'$; therefore $\mathcal{I} \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ iff $\mathcal{I} \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$.

Let us consider the *shortcut elimination* rule as an example. Since the two query formulae differ only for the disjuncts ϕ and ϕ', we just need to show that $\mathcal{I} \models_\psi \phi$ iff $\mathcal{I} \models_\psi \phi'$ for an arbitrary mapping ψ.

If $\mathcal{I} \models_\psi \phi$ then $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}\sqcap\{R_1, \ldots, R_k\}$, therefore

$$\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}\sqcap(\{R_1, \ldots, R_k\} \setminus \{R_\ell\})$$

as well. The rest of the terms in ϕ' are the same of those in ϕ, therefore $\mathcal{I} \models_\psi \phi'$ as well.

Let us assume that $\mathcal{I} \models_\psi \phi'$, we need to show that $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}\sqcap\{R_1, \ldots, R_k\}$. We consider the case in which $k > 1$, the other is not really different. Since $\mathcal{I} \models_\psi \phi'$, then $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}\sqcap(\{R_1, \ldots, R_k\} \setminus \{R_\ell\})$; therefore we must verify that $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}R_\ell$. But this is true because

$$\{\langle x_0, x_1\rangle{:}\sqcap\mathcal{R}_1, \ldots, \langle x_{n-1}, x_n\rangle{:}\sqcap\mathcal{R}_n\} \subseteq \phi'$$

therefore in every set of roles $\mathcal{R}_i$, there is at least a role S_i such that $S_i \preceq S$, which means that $\mathcal{I} \models_\psi \langle x_i, x_{i+1}\rangle{:}S$ for $i = 1, \ldots, n-1$. For the transitivity restriction on S $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}S$, therefore $\mathcal{I} \models_\psi \langle x_0, x_n\rangle{:}R_\ell$ because $S \preceq R_\ell$. □

Proposition 7.12. *The* ***contradiction elimination*** *rule is correct and complete.*

Proof. If $\{x = o_1, x = o_2\} \subseteq \phi$, and o_1 is different from o_2, then there is no interpretation satisfying ϕ. The very same applies to $\phi' \equiv \{z{:}\bot\}$. If $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}\,|_z$, then an arbitrary interpretation satisfying Σ must model $\{\phi_1, \ldots, \phi_n\}\,|_z$ (without the disjunct ϕ); therefore it models $\{\phi', \phi_1, \ldots, \phi_n\}\,|_z$. The other direction is analogous. □

Proposition 7.13. *The* ***simple rolling up*** *rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *simple rolling up* rule, such that ϕ satisfies the rule conditions, and $\Sigma \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

completeness Let $\mathcal{I}$ be a n.r. interpretation satisfying Σ. We assume that $\Sigma \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$, so there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. If $\mathcal{I} \models_\psi \phi_i$ for $i = 1, \ldots, n$ then $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$ as well; so, we are going to show that if $\mathcal{I} \models_\psi \phi'$, then there is a mapping ψ' s.t. $\mathcal{I} \models_{\psi'} \phi$.

If $\mathcal{I} \models_\psi \phi'$ then $\mathcal{I} \models_\psi x{:}\exists R.C$, therefore there is an element $u \in \Delta^\mathcal{I}$ s.t. $(\psi(x), u) \in {R_1}^\mathcal{I}$ and $u \in C^\mathcal{I}$. The mapping $\psi' = \psi[y/u]$ in which y is mapped to u satisfies the required property that $\mathcal{I} \models_{\psi'} \langle x, y\rangle{:}R$ and $\mathcal{I} \models_{\psi'} y{:}C$. The rest of terms in ϕ are satisfied because y does not appear in any other term, therefore $\mathcal{I} \models_{\psi'} \phi$.

Correctness Let $\mathcal{I}$ be an interpretation satisfying Σ; for reasons analogous to those in the completeness proof, we are going to show that if $\mathcal{I} \models_\psi \phi$ then $\mathcal{I} \models_\psi \phi'$.

If $\mathcal{I} \models_\psi \phi$ then $\mathcal{I} \models_\psi \langle x, y\rangle{:}R$ and $\mathcal{I} \models_\psi y{:}C$, it is easy to see that this implies that $\mathcal{I} \models_\psi x{:}\exists R.C$ (the element $\psi(y)$ is the "witness" for the property). For the remaining terms in ϕ', since they are in ϕ as well they are modeled by $\mathcal{I}$ w.r.t. ψ; therefore $\mathcal{I} \models_\psi \phi'$. □

Proposition 7.14. *If the role hierarchy is such that for every unrelated pair of roles*[20] *in the same label there is a functional role that includes both of them, then the* ***functional rolling up*** *and* ***functional inverse rolling up*** *rules are correct and complete.*

Proof. The proof for these two rules are very similar to the previous one for the *simple rolling up* rule. In particular, the correctness part is almost identical and is left to the intuition of the reader. So we are going to concentrate on the completeness.

Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *functional rolling up* rule, such that ϕ satisfies the rule conditions, and $\Sigma \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

[20]They are not included one in each other.

completeness Again, the structure is the same as the previous completeness for the *simple rolling up* rule. The crucial point here is that although we "split" the role conjunction into several existential assertions, the interpretations are such that all the successors must be the very same element.

The point we need to show is that, in any interpretation satisfying Σ, if an element is related to two elements via two different role names being in the very same label and not being included one in the other, then the two related elements must coincide. Formally, let $\mathcal{I}$ be an interpretation satisfying Σ. Let u be an element of $\Delta^{\mathcal{I}}$, and R_1, R_2 two role names in the same label s.t. neither $R_1 \preceq R_2$ nor $R_2 \preceq R_1$. Then, if $(u, v_1) \in {R_1}^{\mathcal{I}}$ and $(u, v_2) \in {R_2}^{\mathcal{I}}$, $v_1 = v_2$.

Note that this is not true if $R_1 \preceq R_2$, because this implies that ${R_1}^{\mathcal{I}} \subseteq {R_2}^{\mathcal{I}}$ but not the converse. The intuition behind the proof is that if the two roles are not related, then there must be an interaction with a functional role that forces the collapsing of successors of u (i.e. there is a functional role F s.t. $R_1 \preceq F$ and $R_2 \preceq F$).

If R_1 and R_2 belongs to the very same label, then either $R_1 \lesssim R_2$ or $R_2 \lesssim R_1$ (by Definition 3.4b); let us assume that $R_1 \lesssim R_2$. We prove the claim by induction on the definition of $\lesssim$ (Definition 3.4a).

The two basic cases $R_1 \preceq R_2$ and $R_2 \preceq R_1$ are ruled out by assumption, so there must be a functional role F such that $R_1 \preceq F$ and $R_2 \preceq F$. This means that both (u, v_1) and (u, v_2) are in $F^{\mathcal{I}}$, and $v_1 = v_2$ because $\mathcal{I}$ satisfies the functional restriction on F.

The property we have just shown applies to the functional (inverse) rolling up rule because we assumed that the *role elimination* rule is not applicable. Therefore, if the set of role names $\{R_1, \ldots, R_n\}$ is included in a label and there is more than one role in it, they must be unrelated.

□

Proposition 7.15. *The* ***simple nominal introduction*** *rule is correct and complete.*

Proof. We assume that the query before the application rule is $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$, ϕ satisfies the conditions of *simple nominal introduction* rule, and none of the rules with a higher priority can be applied to ϕ.

completeness Let $\mathcal{I}$ be a canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ, which is n.r. as well. We assume that $\Sigma \models \{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z$, so there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z$. By definition there is one of the disjuncts such that $\mathcal{I} \models_\psi t$

for each term t contained in it. Moreover, each of the disjuncts in $\{\phi\cup\{y = o_1\}, \ldots, \phi\cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z$ includes at least one of the disjuncts in $\{\phi, \phi_1, \ldots, \phi_n\}|_z$; therefore $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$ as well.

Correctness Let $\mathcal{I}$ be a canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$, so there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We proceed by contradiction by assuming that $\mathcal{I}\not\models_\psi\{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z$.

If $\mathcal{I} \models_\psi \phi_i$ for some $i = 1, \ldots, n$, then

$$\mathcal{I}\models_\psi\{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z$$

as well; so we must assume only that $\mathcal{I} \models_\psi \phi$. If $\psi(y) \in \mathcal{O}^\mathcal{I}$, there is a contradiction with the assumption that none of the disjuncts $\phi \cup \{y = o_i\}$ is satisfied. If $\psi(y) \notin \mathcal{O}^\mathcal{I}$, and since we assumed than $n > 1$, then $(\psi(x), \psi(y)) \in {R_1}^\mathcal{I} \cap {R_2}^\mathcal{I} \cap \ldots \cap {R_n}^\mathcal{I}$. Therefore, by (3.6b) there is a label L s.t. $\{R_1, \ldots, R_n\} \subseteq L$, which is in contradiction with the conditions for the applicability of the rule itself.

Given the fact that we exhausted all the possibilities, finding only contradictions, we must reject the given assumption that

$$\mathcal{I}\not\models_\psi\{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z.$$

Therefore we conclude that

$$\mathcal{I} \models \{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \phi_1, \ldots, \phi_n\}|_z.$$

□

Proposition 7.16. *If the roles involved in the cycle are not transitive and do not have any transitive sub-role, then the* ***cyclic nominal introduction*** *rule is correct and complete.*

Proof. We assume that the query before the rule application is $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$, ϕ satisfies the conditions of *cyclic nominal introduction* rule, and none of the rules with an higher priority can be applied to ϕ. The query after the application of the rule is

$$\Sigma \models \{\phi \cup \{y = o_1\}, \ldots, \phi \cup \{y = o_k\}, \\ eq^{(1)} \cup \phi, \ldots, eq^{(l)} \cup \phi, \phi_1, \ldots, \phi_n\}|_z. \quad (*)$$

completeness We assume that $(*)$ is satisfied. Let $\mathcal{I}$ be a canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ, which is n.r. $\mathcal{I}$ models the query formula in $(*)$ by hypotheses; therefore there is a mapping ψ such that $\mathcal{I}$ models the query formula w.r.t. ψ. By definition there should be one of the disjuncts such that $\mathcal{I}$ is a model for it w.r.t. ψ. However, all the disjuncts in $(*)$ are supersets of at least one disjunct in $\{\phi, \phi_1, \ldots, \phi_n\}|_z$. Therefore $\mathcal{I}$ models $\{\phi, \phi_1, \ldots, \phi_n\}|_z$ w.r.t. ψ as well.

Correctness We must show that if $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$, then $(*)$ is satisfied. Let $\mathcal{I}$ be a canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. By hypothesis $\mathcal{I} \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$, therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}$. We proceed by contradiction assuming that $\mathcal{I}$ does not model the query formula in $(*)$ w.r.t. ψ.

By definition either $\mathcal{I} \models_\psi \{\phi\}$ or $\mathcal{I} \models_\psi \{\phi_i\}$ for $i = 1, \ldots, n$ (since no ϕ_i contains an existential quantifier, we can assume that ψ is a mapping for all the variables). If it is the latter case, then $\mathcal{I}$ models the query formula in $(*)$ w.r.t. ψ as well, leading to a contradiction; therefore we must assume that $\mathcal{I} \models_\psi \{\phi\}$.

Note that, since $\mathcal{I}$ is not a model for $(*)$, then the mapping ψ must not satisfy any term t in $\{y = o \mid o \in \mathcal{O}\}$ or in $\bigcup_i eq^{(i)}$, otherwise $\mathcal{I}$ would be a model for the corresponding disjunct $\phi \cup \{t\}$ in $(*)$. Therefore

- $\psi(y) \notin \mathcal{O}^\mathcal{I}$;
- $\psi(x_1) \neq \psi(x_2)$ for any pair of variables x_1, x_2 occurring in a cycle with y.

Let us consider a cycle $\phi' \subseteq \phi$ involving the variable y in ϕ. Obviously $\mathcal{I} \models_\psi \{\phi'\}$; we are going to show that this is in contradiction with the assumptions we made.

Before analysing the cases we show a property that will be used in the rest of the proof. Given the assumptions, it is the case that for each term $\langle x_1, x_2\rangle{:}R_1 \sqcap \ldots \sqcap R_n$ in ϕ' either $\{\psi(x_1), \psi(x_2)\} \subseteq \mathcal{O}^\mathcal{I}$ or $\psi(x_2) = \psi(x_1)e$ for some element e (see Definition 3.2). If $\psi(x_2) \in \mathcal{O}^\mathcal{I}$ and $(\psi(x_1), \psi(x_2)) \in {R_1}^\mathcal{I}$ then $\psi(x_1) \in \mathcal{O}^\mathcal{I}$ as well by (3.2c). Let us assume that $\psi(x_2) \notin \mathcal{O}^\mathcal{I}$, then $\psi(x_2) = ue$ for some element e of Δ and u of $\Delta^\mathcal{I}$ (See Definition 3.2). By (3.2d) either $u = \psi(x_1)$ or

$\{(\psi(x_1), u), (u, \psi(x_2))\} \subseteq {R_1}^{\mathcal{I}}$. The first case is exactly the result we want to prove; therefore we assume that $\{(\psi(x_1), u), (u, \psi(x_2))\} \subseteq {R_1}^{\mathcal{I}}$, and we show that this is a contradiction. Since $\mathcal{I}$ is canonical, then we can use the property (3.6a) to infer that there is a transitive role $S \preceq R_1$. This is in contradiction with the fact that we assumed that none of the role names involved in any cycle in ϕ is transitive or has a transitive subrole.

We consider three cases according to the length n of the cycle (cardinality of ϕ').

$n = 1$ If the cycle contains a single term then it must be like $\langle y, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$. This means that $(\psi(y), \psi(y)) \in {R_1}^{\mathcal{I}}$, which is impossible by Lemma 3.3 because we assumed that $\psi(y) \notin \mathcal{O}^{\mathcal{I}}$

$n = 2$ If the cycle contains two terms, then they must be of the form

$$\{\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_n, \langle x, y\rangle{:}S_1 \sqcap \ldots \sqcap S_k\}\,.$$

This is because if there is not any variable different from y, then there is a smaller cycle. On the other hand, with more than two different variables and two role terms, having a cycle is impossible. Moreover, we assumed that no other rules can be applied to ϕ, therefore there cannot be any equality like $x = y$ in ϕ; nor can the variables x and y appear in the same order in the two role terms in ϕ' (i.e. like $\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_n$, and $\langle y, x\rangle{:}S_1 \sqcap \ldots \sqcap S_k$), otherwise the *role collapsing* rule would be applicable. Since $\psi(y) \notin \mathcal{O}^{\mathcal{I}}$ we already proved that $\psi(y) = \psi(x)e$ and $\psi(x) = \psi(y)e'$, which is a contradiction.

$n > 2$ Let x_1 and x_2 be the two "neighbour" variables of y (i.e. there is a role term connecting y and x_1 and a different term connecting y and x_2). The two variables must be different, otherwise there will be a cycle of length 2. We can distinguish four cases according to the ordering of the variables in the terms; let us consider each one of them.

– Let us assume that the two terms are

$$\langle y, x_1\rangle{:}R_1 \sqcap \ldots \sqcap R_n,\ \text{and}\ \langle y, x_2\rangle{:}S_1 \sqcap \ldots \sqcap S_k.$$

We showed that $\psi(x_1) = \psi(y)e_{x_1}$ and $\psi(x_2) = \psi(y)e_{x_2}$ for some e_{x_1}, e_{x_2}; therefore $\psi(x_1)$ is not in $\mathcal{O}^{\mathcal{I}}$. This applies to the next variable x connected to x_1: either $\psi(x_1) = \psi(x)e_{x_1}$ or $\psi(x) = \psi(x_1)e_x$. The first case is ruled out because $\psi(x)$ cannot be equal to $\psi(y)$ (we assumed this at the beginning); therefore $\psi(x) = \psi(x_1)e_x$. The same arguments can be carried on for all the subsequent variables in the cycle, and this is clearly in contradiction with the fact that, for the last variable x_2, it must be the case that $\psi(x_2) = \psi(y)e_{x_2}$.

– Let us assume that the two terms are

$$\langle y, x_1\rangle{:}R_1 \sqcap \ldots \sqcap R_n, \text{ and } \langle x_2, y\rangle{:}S_1 \sqcap \ldots \sqcap S_k.$$

Therefore $\psi(x_1) = \psi(y)e_{x_1}$ and $\psi(y) = \psi(x_2)e_y$ for some e_{x_1}, e_y. Using the same argument of the previous case we can show that every variable in the cycle has the property of being in the form $\psi(x) = \psi(y)e_1\ldots e_k e_x$. This is in contradiction with the fact that $\psi(y) = \psi(x_2)e_y$.

– Let us assume that the two terms are

$$\langle x_1, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n, \text{ and } \langle y, x_2\rangle{:}S_1 \sqcap \ldots \sqcap S_k.$$

Therefore $\psi(x_2) = \psi(y)e_{x_2}$ and $\psi(y) = \psi(x_1)e_y$ for some e_{x_2}, e_y. Using the arguments of the previous case starting from x_2 instead of x_1 we get a similar contradiction.

– Let us assume that the two terms are

$$\langle x_1, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n, \text{ and } \langle x_2, y\rangle{:}S_1 \sqcap \ldots \sqcap S_k.$$

Therefore $\psi(y) = \psi(x_1)e_y$ and $\psi(y) = \psi(x_2)e_y$ for some e_y. This is in contradiction with the fact that $\psi(x_1)$ and $\psi(x_2)$ must be different.

At this point we exhausted all the possibilities, finding only contradictions. Therefore we must reject the original assumption that $\mathcal{I}$ does not model $(*)$ w.r.t. ψ. Given the arbitrariness of the choice of interpretation $\mathcal{I}$ we conclude that $(*)$ is satisfied. $\square$

Rules that modify the KB

Proposition 7.17. *The* ***nominal elimination*** *rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *nominal elimination* rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

completeness We assume that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ w.r.t. n.r. interpretations. Let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. We have to show that $\mathcal{I} \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We can distinguish the case in which $\{o{:}P_o\} \subseteq \Sigma$ from the case in which it is not.

- If $\{o{:}P_o\} \subseteq \Sigma$ then $\Sigma' = \Sigma$; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$ by the assumption that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$. As seen in the previous proofs we concentrate in the case in which $\mathcal{I} \models_\psi \phi'$; we just need to show that $\mathcal{I} \models_\psi z = o$. Since we assumed that $\mathcal{I}$ is n.r., then ${P_o}^{\mathcal{I}} = \left\{o^{\mathcal{I}}\right\}$ because $\{o{:}P_o\} \subseteq \Sigma$. Therefore $\psi(z) = o^{\mathcal{I}}$ because $\mathcal{I} \models_\psi z{:}P_o$, which proves that $\mathcal{I} \models_\psi z = o$.

- If $\{o{:}P_o\} \not\subseteq \Sigma$ then the concept name P_o does not appear in Σ, because the assertion $o{:}P_o$ is added to the left hand side whenever the representative concept P_o is introduced in the query formula. We define a new interpretation $\mathcal{I}'$ by adding (or redefining) the mapping ${P_o}^{\mathcal{I}'} = \left\{o^{\mathcal{I}}\right\}$ to $\mathcal{I}$.[21] $\mathcal{I}' \models \Sigma$ because Σ does not contain any reference to P_o; so $\mathcal{I}' \models \Sigma \cup \{o{:}P_o\}$. By assumption there is a mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. Let us consider the case in which $\mathcal{I}' \models_\psi \phi'$: as in the previous case $\psi(z) = o^{\mathcal{I}'} = o^{\mathcal{I}}$. In addition we can use the very same mapping ψ with $\mathcal{I}$, therefore $\mathcal{I} \models_\psi z = o$; this shows that $\mathcal{I} \models_\psi \phi$.

Correctness We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. Let $\mathcal{I}$ be a canonical (w.r.t. Σ') quasi transitive shrub interpretation satisfying Σ'. We have to show that $\mathcal{I} \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$. Again we distinguish the two cases according to whether $o{:}P_o$ is already in Σ. However, in both cases $\mathcal{I} \models \Sigma$ because $\Sigma \subseteq \Sigma'$; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We just need to show that if $\mathcal{I} \models_\psi \phi$, then $\mathcal{I} \models_\psi \phi'$, by showing that $\mathcal{I} \models_\psi z{:}P_o$. However, this is trivially verified because $\mathcal{I} \models_\psi z = o$ (by the assumption that $\mathcal{I} \models_\psi \phi$), and $o^{\mathcal{I}} \in {P_o}^{\mathcal{I}}$ because $\mathcal{I} \models \Sigma'$.

$\square$

Proposition 7.18. *The **simple inverse rolling up** rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *simple inverse rolling up* rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

[21] The domain of the interpretation function $\cdot^{\mathcal{I}}$ contains o by definition; therefore the value $o^{\mathcal{I}}$ is defined.

completeness We assume that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ w.r.t. n.r. interpretations. Let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. We build a new n.r. interpretation $\mathcal{I}'$ equal to $\mathcal{I}$ with the addition/substitution of the interpretation for the new concept name $P_{R^-.C}$ such that

$$P_{R^-.C}{}^{\mathcal{I}'} = \left\{u \in \Delta^{\mathcal{I}} \mid (v,u) \in R^{\mathcal{I}} \text{ and } v \in C^{\mathcal{I}}\right\}.$$

Note that since $P_{R^-.C}$ does not appear either in Σ or in $\{\phi, \phi_1, \ldots, \phi_n\}|_z$, $\mathcal{I}'$ still satisfies Σ, and if $\mathcal{I}' \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ then $\mathcal{I} \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. It is easy to see that $\mathcal{I}'$ satisfies $C \sqsubseteq \forall R.P_{R^-.C}$; i.e. $C^{\mathcal{I}} \subseteq \left\{u \in \Delta^{\mathcal{I}} \mid (v,u) \in R^{\mathcal{I}} \text{ and } v \in C^{\mathcal{I}}\right\}$ (note that $C^{\mathcal{I}'} = C^{\mathcal{I}}$ because $P_{R^-.C}$ does not appear in C). Therefore $\mathcal{I}'$ satisfies Σ', and by assumption there is a mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. We just need to show that if $\mathcal{I}' \models_\psi \phi'$, then there is a mapping ψ' s.t. $\mathcal{I}' \models_{\psi'} \phi$.

Let us assume that $\mathcal{I}' \models_\psi \phi'$, then $\psi(x) \in P_{R^-.C}{}^{\mathcal{I}'}$; therefore there is an element v of $\Delta^{\mathcal{I}}$ such that $(v, \psi(x)) \in R^{\mathcal{I}'}$ and $v \in C^{\mathcal{I}'}$. We define ψ' as $\psi[y/v]$; note that still $\mathcal{I}' \models_{\psi'} \phi'$ because y does not appear in ϕ' by hypothesis. By construction $\mathcal{I}' \models_{\psi'} \langle y, x\rangle{:}R$ and $\mathcal{I}' \models_{\psi'} y{:}C$; therefore $\mathcal{I}' \models_{\psi'} \phi$.

Correctness We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. Let $\mathcal{I}$ be a canonical (w.r.t. Σ') quasi transitive shrub interpretation satisfying Σ'. Obviously $\mathcal{I}$ satisfies Σ as well because it is a subset of Σ'; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We need to show that if $\mathcal{I} \models_\psi \phi$, then $\mathcal{I} \models_\psi \phi'$.

Let us assume that $\mathcal{I} \models_\psi \phi$, then $\psi(y) \in C^{\mathcal{I}}$; therefore $\psi(y) \in \forall R.P_{R^-.C}$ because $\mathcal{I}$ satisfies Σ'. In addition, $(\psi(y), \psi(x)) \in R^{\mathcal{I}}$ because $\mathcal{I} \models_\psi \phi$; therefore $\psi(x) \in P_{R^-.C}{}^{\mathcal{I}}$, which means that $\mathcal{I} \models_\psi \phi'$. □

Proposition 7.19. *The **nominal rolling up** rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *nominal rolling up* rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

completeness We assume that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ w.r.t. n.r. interpretations. Let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. We build a new interpretation $\mathcal{I}'$ by adding (or redefining) the mapping $P_o{}^{\mathcal{I}'} = \left\{o^{\mathcal{I}}\right\}$ to $\mathcal{I}$. The interpretation $\mathcal{I}'$ is still n.r. In addition, if $o{:}P_o$ is already in Σ, then $\mathcal{I}' = \mathcal{I}$ (see the proof of Proposition 7.17). If on the other hand $o{:}P_o$ is not in Σ, it is easy to see that if $\mathcal{I}' \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ then $\mathcal{I} \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ as well because P_o does not appear either in Σ or in $\{\phi, \phi_1, \ldots, \phi_n\}|_z$. The interpretation $\mathcal{I}'$ trivially satisfies Σ'; therefore there is a mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. We just need to show that if $\mathcal{I}' \models_\psi \phi'$, then there is a mapping ψ' s.t. $\mathcal{I} \models_{\psi'} \phi$.

Let us assume that $\mathcal{I}' \models_\psi \phi'$. Since $\mathcal{I}' \models_\psi x{:}(\exists R_1.(C \sqcap P_o) \sqcap \ldots \sqcap \exists R_n.(C \sqcap P_o))$, then $\psi(x) \in (\exists R_1.(C \sqcap P_o))^{\mathcal{I}'}$; therefore there is an element u of $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$ such that $(\psi(x), u) \in {R_1}^{\mathcal{I}'} = {R_1}^{\mathcal{I}}$, $u \in C^{\mathcal{I}'} = C^{\mathcal{I}}$ (because either C does not include P_o or ${P_o}^{\mathcal{I}'} = {P_o}^{\mathcal{I}}$) and $u \in {P_o}^{\mathcal{I}'}$. Note that ${P_o}^{\mathcal{I}'} = \{o^{\mathcal{I}}\}$, therefore $u = o^{\mathcal{I}} = o^{\mathcal{I}'}$. The very same arguments, applied to $\psi(x) \in (\exists R_i.P_o)^{\mathcal{I}'}$ for $i = 2, \ldots, n$, allow us to conclude that $(\psi(x), u) \in {R_i}^{\mathcal{I}'}$. We define ψ' as $\psi[y/u]$. By construction $\mathcal{I} \models_{\psi'} y = o$, $\mathcal{I} \models_{\psi'} y{:}C$, and $\mathcal{I} \models_{\psi'} \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$; therefore $\mathcal{I}' \models_{\psi'} \phi$.

Correctness We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. Let $\mathcal{I}$ be a canonical (w.r.t. Σ') quasi transitive shrub interpretation satisfying Σ'. Obviously $\mathcal{I}$ satisfies Σ as well because it is a subset of Σ'; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We need to show that if $\mathcal{I} \models_\psi \phi$ then $\mathcal{I} \models_\psi \phi'$.

Let us assume that $\mathcal{I} \models_\psi \phi$, therefore there is an element $\psi(y)$ of $\Delta^{\mathcal{I}}$ such that $\psi(y) = o^{\mathcal{I}}$ (because $\mathcal{I} \models_\psi y = o$), $\psi(y) \in C^{\mathcal{I}}$ (because $\mathcal{I} \models_\psi y{:}C$), and $(\psi(x), \psi(y)) \in {R_1}^{\mathcal{I}} \cap \ldots {R_n}^{\mathcal{I}}$ (because $\mathcal{I} \models_\psi \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$). In addition, $\psi(y) \in {P_o}^{\mathcal{I}}$ because $\mathcal{I}$ satisfies Σ' ($o{:}P_o$ is in Σ'). Therefore $\psi(y) \in (\exists R_1.(C \sqcap P_o) \sqcap \exists R_2.P_o \sqcap \ldots \sqcap \exists R_n.P_o)^{\mathcal{I}}$ and $\mathcal{I} \models_\psi \phi'$. $\square$

Proposition 7.20. *The **cycle breaking** and **simple cycle breaking** rules are correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query after the application of one of the two rules.

completeness We proceed as in the previous Proposition 7.19. Let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ, and $\mathcal{I}'$ defined as $\mathcal{I}$ with the mapping ${P_o}^{\mathcal{I}'} = \{o^{\mathcal{I}}\}$. As in the previous proposition, $\mathcal{I}'$ is still n.r. and it satisfies Σ'. Therefore by hypothesis there is a mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. As before we assume that $\mathcal{I}' \models_\psi \phi'$. We are going to show that $\mathcal{I} \models_\psi \phi$ as well. Note that in the application of both the rules the variables x and y are still in ϕ' (in the *cycle breaking* rule either because of the path or because $x \equiv y$); therefore both $\psi(x)$ and $\psi(y)$ are defined.

Let us consider the *simple cycle breaking* rule; since $\mathcal{I}' \models_\psi \phi'$, then $(\psi(x), \psi(y)) \in {R_1}^{\mathcal{I}'} = {R_1}^{\mathcal{I}}$ and there are $n-1$ elements $e_2, \ldots, e_n$ of $\Delta^{\mathcal{I}'} = \Delta^{\mathcal{I}}$ such that $e_i \in {P_o}^{\mathcal{I}'}$ and $(\psi(x), e_i) \in {R_i}^{\mathcal{I}'} = {R_i}^{\mathcal{I}}$. Since $\mathcal{I}'$ is n.r., then $e_2 = \ldots = e_n = \psi(y)$, so $(\psi(x), \psi(y)) \in {R_1}^{\mathcal{I}} \cap \ldots \cap {R_n}^{\mathcal{I}}$; therefore $\mathcal{I} \models_\psi \langle x, y\rangle{:}R_1 \sqcap \ldots \sqcap R_n$. The *cycle breaking* rule is the same.

Correctness The proof is exactly the same as for proposition 7.19 for the *nominal rolling up* rule. □

Proposition 7.21. *The **nominal inverse rolling up** rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *nominal inverse rolling up* rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

completeness We assume that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ w.r.t. n.r. interpretations. Let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. Again, we proceed as in Proposition 7.19 and Proposition 7.18 by considering $\mathcal{I}'$ as equal to $\mathcal{I}$ with the mapping ${P_o}^{\mathcal{I}'} = \{o^{\mathcal{I}}\}$, and for each $i = 1, \ldots, n$ ${P_{R_i\text{-}y}}^{\mathcal{I}'} = \{u \in \Delta^{\mathcal{I}} \mid (o^{\mathcal{I}}, u) \in {R_i}^{\mathcal{I}}\}$ if $o^{\mathcal{I}} \in C^{\mathcal{I}}$ or ${P_{R_i\text{-}y}}^{\mathcal{I}'} = \emptyset$ otherwise. This interpretation is still n.r.; in addition, it satisfies Σ because all the $P_{R_i\text{-}y}$ do not appear in Σ, and either P_o appears in Σ and ${P_o}^{\mathcal{I}'} = {P_o}^{\mathcal{I}}$ because $\mathcal{I}$ is n.r., or P_o does not appear in Σ. The interpretation $\mathcal{I}'$ satisfies the assertion $o{:}P_o$ by definition; now we are going to show that it satisfies $(C \sqcap P_o) \sqsubseteq (\forall R_1.P_{R_1\text{-}y} \sqcap \ldots \sqcap \forall R_n.P_{R_n\text{-}y})$ as well. Note that $C^{\mathcal{I}'} = C^{\mathcal{I}}$ because either P_o does not syntactically appear in C or ${P_o}^{\mathcal{I}'} = {P_o}^{\mathcal{I}}$; therefore either $(C \sqcap P_o)^{\mathcal{I}'} = \{o^{\mathcal{I}'}\}$ or $(C \sqcap P_o)^{\mathcal{I}'} = \emptyset$. In the first case, let us consider the role name R_i for some $i = i, \ldots, n$. If $e \in \Delta^{\mathcal{I}'}$ is such that $(o^{\mathcal{I}'}, e) \in {R_i}^{\mathcal{I}'}$ then $e \in \{u \in \Delta^{\mathcal{I}} \mid (o^{\mathcal{I}}, u) \in {R_i}^{\mathcal{I}}\}$; therefore $e \in {P_{R_i\text{-}y}}^{\mathcal{I}'}$ by definition, and $o^{\mathcal{I}'} \in (\forall R_i.P_{R_i\text{-}y})^{\mathcal{I}'}$ because of the arbitrariness of e. Given the fact that this is true for all $i = 1, \ldots, n$ then $(C \sqcap P_o)^{\mathcal{I}'} \subseteq (\forall R_1.P_{R_1\text{-}y} \sqcap \ldots \sqcap \forall R_n.P_{R_n\text{-}y})^{\mathcal{I}'}$. In the case that $(C \sqcap P_o)^{\mathcal{I}'} = \emptyset$ the inclusion constraint is trivially satisfied. This shows that $\mathcal{I}'$ satisfies Σ'. By assumption there is a mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. We need to show that if we assume $\mathcal{I}' \models_\psi \phi'$, then there is a mapping ψ' s.t. $\mathcal{I} \models_{\psi'} \phi$.

Since we assumed that $\mathcal{I}' \models_\psi \phi'$, then $\psi(x) \in (P_{R_1\text{-}y} \sqcap \ldots \sqcap P_{R_n\text{-}y})^{\mathcal{I}'}$. Therefore $(o^{\mathcal{I}}, \psi(x)) \in {R_i}^{\mathcal{I}'} = {R_i}^{\mathcal{I}}$ for all $i = 1, \ldots, n$ and $o^{\mathcal{I}} \in C^{\mathcal{I}'} = C^{\mathcal{I}}$ because ${P_{R_i\text{-}y}}^{\mathcal{I}'}$ is not empty. The new mapping $\psi' = \psi[y/o^{\mathcal{I}}]$ is clearly such that $\mathcal{I} \models_{\psi'} \phi$ because y does not appear in other terms in ϕ by hypothesis.

Correctness We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. Let $\mathcal{I}$ be a canonical (w.r.t. Σ') quasi transitive shrub interpretation satisfying Σ'. Obviously $\mathcal{I}$ satisfies Σ as well because is a subset of Σ'; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We need to show that if we assume that $\mathcal{I} \models_\psi \phi$ then $\mathcal{I} \models_\psi \phi'$.

Since $\mathcal{I} \models_\psi \phi$, then $\psi(y) = o^{\mathcal{I}}$, $\psi(y) \in C^{\mathcal{I}}$, and $(\psi(y), \psi(x)) \in {R_i}^{\mathcal{I}}$ for all $i = 1, \ldots, n$. In addition, $\psi(y) \in (\forall R_1.P_{R_1\text{-}y} \sqcap \ldots \sqcap \forall R_n.P_{R_n\text{-}y})^{\mathcal{I}}$ because $\mathcal{I}$ satisfies the inclusion $(C \sqcap P_o) \sqsubseteq (\forall R_1.P_{R_1\text{-}y} \sqcap \ldots \sqcap \forall R_n.P_{R_n\text{-}y})$, and $\psi(y) \in (C \sqcap P_o)^{\mathcal{I}}$. Therefore for all $i = 1, \ldots, n$ we have the combination $(\psi(y), \psi(x)) \in {R_i}^{\mathcal{I}}$ and $\psi(y) \in (\forall R_i.P_{R_i\text{-}y})^{\mathcal{I}}$, which implies that $\psi(x) \in {P_{R_i\text{-}y}}^{\mathcal{I}}$. This allows us to conclude that $\mathcal{I} \models_\psi \phi'$. □

Proposition 7.22. *The **inverse cycle breaking** rule is correct and complete.*

Proof. Let $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$ be the query before the application of the *inverse cycle breaking* rule, such that ϕ satisfies the rule conditions, and $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ is the resulting query.

completeness Let us assume that $\Sigma' \models \{\phi', \phi_1, \ldots, \phi_n\}|_z$ w.r.t. n.r. interpretations, and let $\mathcal{I}$ be a n.r. canonical (w.r.t. Σ) quasi transitive shrub interpretation satisfying Σ. We proceed as in proposition 7.21 by considering $\mathcal{I}'$ as equal to $\mathcal{I}$ with the mapping ${P_o}^{\mathcal{I}'} = \{o^{\mathcal{I}}\}$, and for each $i = 1, \ldots, n$ ${P_{R_i\text{-}y}}^{\mathcal{I}'} = \{u \in \Delta^{\mathcal{I}} \mid (o^{\mathcal{I}}, u) \in {R_i}^{\mathcal{I}}\}$. Again, interpretation $\mathcal{I}'$ is still n.r. and satisfies Σ. Using the same arguments as for Proposition 7.21 we can show that $\mathcal{I}'$ satisfies Σ' as well. By assumption there is a mapping mapping ψ such that $\mathcal{I}' \models_\psi \{\phi', \phi_1, \ldots, \phi_n\}|_z$. We are going to show that if we assume $\mathcal{I}' \models_\psi \phi'$, then $\mathcal{I} \models_\psi \phi$ (note that both the variables x and y maching the precondition are still in ϕ').

Since we assumed that $\mathcal{I}' \models_\psi \phi'$, then $\psi(x) \in (P_{R_1\text{-}y} \sqcap \ldots \sqcap P_{R_n\text{-}y})^{\mathcal{I}'}$, so $(o^{\mathcal{I}}, \psi(x)) \in {R_i}^{\mathcal{I}'} = {R_i}^{\mathcal{I}}$ for all $i = 1, \ldots, n$. In addition, $\psi(y) = o^{\mathcal{I}}$ because $y = o$ is still in ϕ'; therefore $\mathcal{I} \models_\psi \langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_n$ (i.e. $\mathcal{I} \models_\psi \phi$ because $\phi \subseteq \phi' \cup \{\langle y, x\rangle{:}R_1 \sqcap \ldots \sqcap R_n\}$).

Correctness We assume that $\Sigma \models \{\phi, \phi_1, \ldots, \phi_n\}|_z$. Let $\mathcal{I}$ be a canonical (w.r.t. Σ') quasi transitive shrub interpretation satisfying Σ'. Obviously $\mathcal{I}$ satisfies Σ as well because is a subset of Σ'; therefore there is a mapping ψ such that $\mathcal{I} \models_\psi \{\phi, \phi_1, \ldots, \phi_n\}|_z$. We need to show that if we assume that $\mathcal{I} \models_\psi \phi$, then $\mathcal{I} \models_\psi \phi'$.

Since $\mathcal{I} \models_\psi \phi$ then $\psi(y) = o^{\mathcal{I}}$, and $(\psi(y), \psi(x)) \in {R_i}^{\mathcal{I}}$ for all $i = 1, \ldots, n$. In addition, since $\psi(y) = o^{\mathcal{I}} \in {P_o}^{\mathcal{I}}$ we derive that $\psi(y) \in (\forall R_i.P_{R_i\text{-}y})^{\mathcal{I}}$ for all $i = 1, \ldots, n$, because $\mathcal{I}$ satisfies the inclusion $P_o \sqsubseteq (\forall R_1.P_{R_1\text{-}y} \sqcap \ldots \sqcap \forall R_n.P_{R_n\text{-}y})$. This means that $\psi(x) \in {P_{R_i\text{-}y}}^{\mathcal{I}}$ for all $i = 1, \ldots, n$; therefore $\mathcal{I} \models_\psi x{:}(P_{R_1\text{-}y} \sqcap \ldots \sqcap P_{R_n\text{-}y})$ (i.e. $\mathcal{I} \models_\psi \phi'$). □

7.5 Speeding up the answer

The major problem for the performances of the query algorithm lies in the nondeterministic substitution of the variables with the individual names. Note this is true for both the transformation of a "retrieval" query into a boolean query (see Section 7.1.2), and the actual boolean query answering (see Section 7.4). Therefore, reducing the number variable substitutions to perform, and the number of individuals which take part in each substitution is the way to go for a better performing algorithm.

The variables which need to be substituted by individual names cannot be reduced in the phase of transformation of a query into a boolean query, but there are a lot of possibilities in the application of the transformation rules.

Example 7.6
Let us consider a cyclical query like

$$\{\langle x, y\rangle{:}R_1, \langle y, z\rangle{:}R_2, \langle z, x\rangle{:}R_3, \langle x, x\rangle{:}S\}\,.$$

This query contains two cycles: one involving the variables x, y, and z, and a second constituted by the loop in x (i.e. $\langle x, x\rangle{:}S$). The *cyclic nominal introduction* rule can be applied to any of the tree variables. However, if it is applied to x, both the cycles are "eliminated" at once; while applying the rule to either y or z leaves the loop in x in each one of the newly generated disjuncts.

Note that, in this example, choosing x reduces the size of the transformed query of a logarithmic factor.

For reducing the number of candidate individuals for a variable name, we are considering two different techniques. The first one relies on the concept terms contained in the query, while the second on the structure of role terms.

They both rely on the observation that if a subset of the terms in a disjunct are never satisfied, then neither the disjunct will be satisfied. Therefore, if in a query contains a concept term like $x{:}C$, and the variable x must be substituted by an individual name, we can just consider the subset of individual names which can be instance of C (verifiable by instance checking tests like $\Sigma \models a{:}\neg C$, see Section 2.2). Note that for variable name different from the joining variable z (see Section 7.3.1) we can even consider the more restricted set of individual names which are instance of C.

We can make an analogous consideration for role terms, with the difference that the kind of role assertions in a $\mathcal{SH}f$ KB are not very expressive. In particular, they do not allow to express incomplete information as the concept assertions do. Using concept assertions we can states something like $a{:}C \sqcup D$ which leave a degree of uncertainty about the properties of a.[22] This is impossible to do for role assertions, which are usually limited to simple statements like $\langle a, b\rangle{:}R$. This limited expressivity for roles is shared by most of the DLs studied and/or implemented.

This limitation can be used in order to restrict the number of candidate individuals. Let us consider for example a term like $\langle x, y\rangle{:}R$, where the variable y must be substituted by an individual name. In DL like $\mathcal{SH}f$ we can restrict to names which appear explicitly in assertions like $\langle a, b\rangle{:}R$ as second element of the pair.[23] Note that this argument is no longer valid for DLs providing the inverse role constructor; on the other hand we are confident that effective optimisations can be devised in most of the cases.

We did not investigate these ideas yet, and all these optimisations must be carefully verified in order to avoid incomplete or incorrect answers. This will be subject of our future work.

[22]Note that the incomplete information may be stated in more subtle ways and/or implied by the terminology.

[23]Note that the role hierarchy must be taken into account as well, therefore considering role assertions with sub–roles of R as well.

Chapter 8

Implementation and testing

The main goal of our DL system implementation is not the delivery of a complete and robust system, but the development of a prototype to be used as test bed for verifying the validity of our approach and for suggesting new directions to investigate.

The first objective was the analysis of the overall performances of the system compared to other state of the art DL reasoners. This point is important because we know that the naive implementation of the tableaux–like method for DLs leads to unacceptable performance (see Horrocks [1997]). Modern DL systems, on the other hand, adopt a series of optimization techniques which provide exceptional results (see Horrocks and Patel-Schneider [1999], Haarslev and Möller [1999, 2000c]). Unsurprisingly, these techniques provide better results if the algorithm has control over the whole reasoning process. In our case, on the other hand, we have a sharp separation between the precompletion phase (dealing with Abox assertions) and the invocation of the terminological reasoner. The other goal of the experimentation was to understand the impact of different optimisation techniques applied at the Abox precompletion level.

The focus of this chapter is not to describe the actual software artifact we have developed, but rather on the underlying ideas and techniques used.

8.1 Description of the system

The DL system has been written in Common Lisp because it is an excellent language for symbolic manipulation, and it enables the programmer to easily extend and modify the code to experiment with different approaches.

Most available DL systems have been written using Lisp, mainly for the above mentioned reasons, but also because we are dealing with highly intractable algorithms;

therefore, twiddling with a low level programming language for gaining an order of magnitude (which is not that easy against modern Lisp compilers) is pointless without first investigating higher level optimisations, which can lead to gains of several order of magnitude.

Interaction with the system is performed by means of Lisp function and macros designed to be conform to the KRSS standard (see KRSS), as well as compatible with the syntax used by the FaCT system (see Horrocks [1997]).

The terminological reasoner used is FaCT. Since it is written in Common Lisp as well, its integration in the code as a library was very easy, and a natural choice since it is the system used in our research group. However, in principle, any DL terminological reasoner can be integrated as a library.

8.2 Optimising the algorithm

The experience with previous DL systems shows that the direct implementation of the tableaux–based satisfiability algorithms provides very poor performance, unacceptable for any real application. Even if the experiments with other DL systems have been mainly at the terminological level, we can try to extract some lessons from those experiences (see Haarslev and Möller [1999], Horrocks and Patel-Schneider [1999]).

One of the advantages of the precompletion technique is that the precompletion phase is completely separated from the terminological reasoning, so optimisation techniques implemented at the two levels do not interact adversely. The terminological reasoner we are using (i.e. FaCT) is already optimised, therefore we focus on the optimisation we can perform during the precompletion.

Among the answers we are looking for from the experiments is whether optimisations made at the precompletion level make a real difference, and which ones produce the best results. Given the fact that during the precompletion no new individuals are created, the source of complexity must be related to the nondeterminism in the $\sqcup$–rule.

8.2.1 Evaluation strategy

According to the formal description of the algorithm in Chapter 5, the order in which the rules are selected is irrelevant to the correctness and completeness of the algorithm.[1] However, for specifying the actual algorithm we decided to consider all the

[1]Note that with $\mathcal{SH}f$ this is no longer true for terminological reasoning. The difference is the absence of the $\exists$–rule in the generation of precompletions.

constraints associated to a given individual before moving to a different individual. Note that there is always the possibility that constraints associated with different individuals cause the addition of new constraints for the individual which has been previously considered. In this case, eventually the individual in question will be selected again.

Intuitively, this strategy may give good results when there is not a strong interdependency between individuals (i.e. role assertions). On the other hand, it may cause an unnecessary overhead if there are “easy” contradictions in individuals not yet considered.

8.2.2 Axiom absorption and lazy expansion

General axioms are one of the major sources of nondeterminism; in fact, in every axiom $C \sqsubseteq D$ is hidden the disjunctive formula $D \sqcup \neg C$ applied to every individual name. One the most effective ways of reducing the effects of axioms is the so called *absorption* technique used in conjunction with lazy expansion of concept names (see Horrocks and Tobies [2000]).

Roughly speaking, the idea behind this technique is to transform a general axiom into the special form $A \sqsubseteq C$ where A is a concept name. Then the axiom is treated as a sort of definition for the name A and ignored until a concept constraint of the form $o{:}A$ is examined; at this point the “definition” C of A is added to the label of o (i.e. the new constraint $o{:}C$). This basic idea can be extended to negated concept names as well; i.e. having definitions of the form $\neg A \sqsubseteq C$. However, their combination must be used carefully to avoid incorrect results. We implemented the absorption algorithm described in Horrocks and Tobies [2000].

8.2.3 Lexical normalisation

Concept expressions are normalised and encoded according to the transformation rules described in Horrocks and Patel-Schneider [1999]. In the normal form, concept expressions can be concept names, conjunctions of normal form concepts, universal quantification constructors, and the negation of a normal form. Conjunctions are represented as sets, so the order of the conjuncts does not affect the syntactic equivalence of different expressions; in addition, nested conjunctions are flattened (e.g. expressions like $((C_1 \sqcap C_2) \sqcap D)$ are transformed into $(C_1 \sqcap C_2 \sqcap D)$).

Normal form expressions are uniquely associated with an identifier which is used

Concept expression	Normal form
$\bot$	$\neg\top$
$C_1 \sqcup \ldots \sqcup C_n$	$\neg(\sqcap\{\neg C_1, \ldots, \neg C_n\})$
$\exists R.C$	$\neg(\forall R.\neg C)$
$\neg\neg C$	C
$C_1 \sqcap \ldots \sqcap C_n$	$\sqcap\{C_1, \ldots, C_n\}$
$\sqcap\{\sqcap\{C_1, \ldots, C_n\}, D_1, \ldots, D_k\}$	$\sqcap\{C_1, \ldots, C_n, D_1, \ldots, D_k\}$
$\sqcap\{C\}$	C
$\forall R.\top$	$\top$
$\sqcap\{\top, C_1, \ldots, C_n\}$	$\sqcap\{C_1, \ldots, C_n\}$
$\sqcap\{\neg\top, C_1, \ldots, C_n\}$	$\neg\top$
$\sqcap\{C, \neg C, C_1, \ldots, C_n\}$	$\neg\top$

Table 8.1: Lexical normalisation rules

in place of the complex expression itself. A table is used to relate the normalised expressions to their respective identifiers. In this way, every time an expression is encountered a second time during the parsing, the very same identifier is used to represent it. This mechanism enables the possibility of detecting trivial inconsistencies at an early stage, without the necessity of expanding the concept expressions.

For example, let us consider the expression $(\exists R.(C \sqcup D) \sqcap \forall R.(\neg C \sqcap \neg D))$. The parser is recursively invoked over its subexpressions, first the expression $\exists R.(C \sqcup D)$ is transformed into the negation of an universal quantification $\neg(\forall R.\sqcap\{\neg C, \neg D\})$. During the transformation each subexpression is associated to an identifier, and the identifier table will contain the mappings $id_1 \mapsto \sqcap\{\neg C, \neg D\}$, and $id_2 \mapsto \forall R.id_1$.[2] During the parsing of the second top level conjunct $\forall R.(\neg C \sqcap \neg D)$ the parsed subexpressions are recognised as stored in the table and substituted by their identifiers. So the resulting normal form is $\sqcap\{\neg id_2, id_2\}$ which is immediately normalised as $\neg\top$.

8.2.4 Backjumping

Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search known as thrashing.

Example 8.1

[2]There will also be identifiers for the other subexpressions, but these are sufficient to show our point.

Consider the set of constraints

$$\left\{ \begin{array}{l} a{:}(C_1 \sqcup D_1), \dots, a{:}(C_n \sqcup D_n), a{:}\forall S.\forall R.\neg C, \\ b{:}\exists R.(C \sqcap D), \langle a, b \rangle{:}S \end{array} \right\},$$

which may cause the exploration of 2^n alternative combinations of constraints on a deriving from the concepts $(C_1 \sqcup D_1), \dots, (C_n \sqcup D_n)$, while the true cause for the failure is related to the individual b. For example, this will happen if the rule application strategy forces the evaluation of all the constraints associated with an individual before considering a different individual.

This problem is addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems (see Baker [1995]). Backjumping works by labeling concept constraints with a dependency set indicating the branching points on which they depend. A concept constraint $a{:}C$ depends on a branching point if $a{:}C$ was added to the label by the $\sqcup$–rule generating the branching point or if $a{:}C$ was generated by a different rule and the concept constraints involved in the rule depends on the branching point.[3]

For example, the constraints $\{a{:}\forall R.C, \langle a, b\rangle{:}R\}$ generate the new constraint $b{:}C$ by means of the $\forall$–rule. The constraint $b{:}C$ inherits the very same dependency set as the constraint $a{:}\forall R.C$.

When a clash is discovered, the dependency sets of the clashing concept constraints can be used to identify the most recent branching point where exploring the other branch might alleviate the cause of the clash. The algorithm can then jump back over intervening branching points without exploring alternative branches.

In more detail, when a clash is detected the union of the dependency sets of the clashing concepts is taken, and backtracking is performed. During backtracking, each branching point encountered is checked against the dependency set to see if it is a member. If it is not in the dependency set, then the other branch is ignored and backtracking continues. If the branching point is in the dependency set, and the other branch has not been explored, then backtracking stops and the algorithm proceeds with the exploration of the second branch. If both branches have already been explored, then the dependency sets from the two branches are unioned and backtracking continues.

Note that when a contradiction is discovered by the verification of a label, there is no way for our algorithm of knowing the source of the contradiction; therefore, is

[3]Since new role assertions are never introduced, the dependency is only related to concept expressions.

returned the union of the dependency set of all constraints of the label. As we are going to show in Section 8.2.6, we can modify the algorithm in order to make the backjumping more effective.

With respect to the given example, the precompletion algorithm generates the first precompletion

$$\left\{\begin{array}{l} a{:}(C_1 \sqcup D_1), \ldots, a{:}(C_n \sqcup D_n), a{:}\forall S.\forall R.\neg C, \\ a{:}C_1, \ldots, a{:}C_n \\ b{:}\exists R.(C \sqcap D), b{:}\forall R.\neg C, \langle a, b\rangle{:}S \end{array}\right\},$$

by choosing the first concept for each constraint containing the disjunction, and applying the $\forall$–rule to the constraints $a{:}\forall S.\forall R.\neg C$ and $\langle a, b\rangle{:}S$. In this process, the algorithm generates n different branching points, and every constraint $a{:}C_1, \ldots, a{:}C_n$ is labelled with a different branching point. The constraints associated with b have no branching point associated with them (this indicate a dependency with the top level), so when the terminological reasoner discovers the inconsistency of the label associated with b there is no reason to explore the alternative options $a{:}D_n, \ldots, a{:}D_1$.

8.2.5 Abox partitioning

Analysing the precompletion algorithm, it is easy to realise that individuals can influence each other only by means of role assertions. In addition, these assertions are "static", in the sense that they do not change during the precompletion process. This guarantees that unconnected parts of the Abox can be precompleted independently. The whole KB is satisfiable iff each connected component is independently satisfiable.

This property makes little difference in case of KB satisfiability, since the only advantage is on the memory occupation.[4] However, it can be a significant improvement in case of the instance checking problem (i.e. verifying whether an individual is member of a concept in every model of the KB, see 2.2). In fact, this problem is usually reduced to KB (un)satisfiability: given a KB $\langle \mathcal{T}, \mathcal{A}\rangle$, an individual a, and a concept C, $\langle \mathcal{T}, \mathcal{A}\rangle \models a{:}C$ iff the KB $\langle \mathcal{T}, \mathcal{A} \cup \{a{:}\neg C\}\rangle$ is unsatisfiable. Knowing that the initial KB $\langle \mathcal{T}, \mathcal{A}\rangle$ is satisfiable (this is usually the case) allows us to verify the satisfiability of the KB by considering only the assertions about individuals "connected" with the name in question (the individual a). The rest of the Abox can be ignored.

[4]Even though one can use the property for loading into the main memory only the part of the Abox relevant for the current precompletion (see for example Elhaik and Rousset [1998]).

8.2.6 Using the terminological reasoner

According to the described algorithms the terminological reasoner is used only when a precompletion is generated. However, it can be used in different phases of the algorithm in a more sophisticated way. The starting assumption is that the terminological reasoner is more efficient than the precompletion phase; therefore it can be used not only for verifying the consistency of a precompletion but also for guiding the algorithm. In addition, is reasonable to expect that the Abox is much bigger than the Tbox.

Result caching As pointed out in Donini et al. [1996a], caching the satisfiability results for the tested concept expressions is essential for maintaining the complexity of the actual algorithm in the EXPTIME theoretical complexity. In addition, in our case it can allow us to avoid the overhead of converting a concept expression from the internal representation to a format suitable for the terminological reasoner.

As described in Section 8.2.3, the system keeps a table of all the normalised concept expressions the system comes across during the precompletion. When the terminological reasoner is invoked for checking the satisfiability of a concept expression (the conjunction of one or more concepts in a label), the result is stored in the table as well. If a concept expression, having a cached satisfiability value, must be checked again, its cached value is returned without invoking the terminological reasoner again.

Early inconsistency detection The first alternative use of the terminological reasoner is as an early inconsistency detector. This is based on the observation that if the constraints applied to an individual are inconsistent they will be inconsistent in any generated precompletion. Therefore the label of an individual can be verified even before an actual precompletion is generated. If an individual is found having an inconsistent label, the algorithm stops exploring the current search branch and backtracks to the appropriate saved state of the precompletion process.

Taking this approach too far can be damaging if the terminological reasoner is called too often. We decided to use the focusing/un-focusing of the precompletion process on an individual as a trigger for the label verification. The label is verified when all the constraints associated to an individual have been considered, before considering a different individual. If during subsequent precompletion of the rest of the KB no new constraints are added to an individual which as been already precompleted, there is no reason to verify its label again.

Modal verification Backjumping is an essential technique for pruning the search space, however it works properly only if we have precise knowledge of the constraints which generate a clash. Our problem is that in case of a clash detected by the terminological reasoner, we cannot get the information about the constraints responsible for the clash. We could assume that any concept in the label can be the cause of the clash, but this would make the backjumping technique much less effective.

For example, the label $\{C, \neg D, \exists S.C, \exists R.C, \forall R.D\}$ is inconsistent if the terminology contains an axiom like $C \sqsubseteq \neg D$. However, the inconsistency can only be detected by the terminological reasoner. Once the unsatisfiability of the label is discovered, the system cannot tell which element of the label caused the inconsistency; in principle it can even be among the set $\{C, \neg D\}$.

We can improve the algorithm observing that if during the precompletion a clash has not been detected, the only possible cause for an inconsistency must be associated with an "anonymous" element whose existence is enforced by an existential quantification. This is because the tree–like model property of the logic guarantees that no constraints can be "pushed back" from these "anonymous" elements, and contradictions generated in the propositional part of the labels (e.g. like C and $\neg C$) are detected during the precompletion process.

When there is the necessity of verifying the satisfiability of a label, the standard algorithm builds a new concept by conjoining all the concepts in the label. With the modal verification technique, the only concepts considered are the universal and existential quantifications (i.e. $\exists R.C$ and $\forall R.C$). The algorithm selects the smallest subsets of the label which can be independently verified without compromising the completeness of the algorithm. In the previous example the terminological reasoner is used to verify the two concepts $(\exists R.C \sqcap \forall R.D)$ and $\{\exists S.C\}$ independently.

The basic idea for selecting these subsets is to consider each existential concept in a different subset, and adding the universal restrictions which can apply to it (i.e. having the same or a more general role name). However, this simple approach does not work with $\mathcal{SH}f$, because of the interaction between functional role and role hierachy. In fact, two (or more) of those "anonymous" elements may be forced to co-refer by a common functional super-role (see Section 3.1.1, and Example 7.5). For example, a label containing the two concepts $\exists R.C$ and $\exists S.\neg C$, is not satisfiable if there is a functional role F such that $R \preceq F$ and $S \preceq F$. This contradiction would not be detected by considering the two existential concepts separately.

The solution for this problem is to check existential concepts whose role share common functional super–roles together. This must be extended to “chains” of roles like $R \preceq F_1$, $S \preceq F_1$, $S \preceq F_2$, and $T \preceq F_2$ as well (see Example 7.5). Universal concepts alone do not generate contradictions without the actual existence of a successor; i.e. two concepts like $\forall R.C$ and $\forall R.\neg C$ are not contradictory, unless they are combined with an corresponding existential concept (or Abox assertion). Therefore they do not need to be verified by themselves (they are considered by means of exitential concepts, or of the corresponding precompletion rule).

Using this technique, in case of unsatisfiability we can narrow the set of involved constraints. Therefore it can be extremely useful in conjunction with backjumping, because it enables a more precise identification of the backtrack point responsible for the clash.

Example 8.2
Let us consider the following variation of Example 8.1

$$\left\{\begin{array}{l} a{:}(C_1 \sqcup D_1), \ldots, a{:}(C_n \sqcup D_n), \\ a{:}\forall R.\neg C, a{:}\exists R.(C \sqcap D) \end{array}\right\}.$$

In this case backjumping alone would not provide any improvement because the generated precompletion

$$\left\{\begin{array}{l} a{:}(C_1 \sqcup D_1), \ldots, a{:}(C_n \sqcup D_n), \\ a{:}C_1, \ldots, a{:}C_n \\ a{:}\forall R.\neg C, a{:}\exists R.(C \sqcap D) \end{array}\right\},$$

is associated with the single individual a. Therefore, the terminological reasoner is invoked with the concept

$$(C_1 \sqcup D_1) \sqcap \ldots \sqcap (C_n \sqcup D_n) \sqcap C_1 \sqcap \ldots \sqcap C_n \sqcap \forall R.\neg C \sqcap \exists R.(C \sqcap D),$$

and the algorithm is unable to detect which constraints are the cause of the unsatisfiability.

By checking the modal part only, the terminological reasoner is invoked with the concept $\forall R.\neg C \sqcap \exists R.(C \sqcap D)$. When the unsatisfiability is reported, the algorithm knows that the clash is generated by one of the two constraints $a{:}\forall R.\neg C$, $a{:}\exists R.(C \sqcap D)$ and the backjumping is more effective.

Instance checking by subsumption The terminological reasoner can sometimes be used to avoid precompletion in case of instance checking, by using an approximation of the Most Specific Concept technique (see Era and Donini [1992]). The idea is to build a concept expression which is guaranteed to contain the individual, and is as specific as possible. The trivial way of doing it is by conjoining all the concepts in the label of the individual, but the role constraints can be considered as well to narrow the expression by using existential quantification.

For example, from the set of constraints $\{a{:}C, \langle a, b\rangle{:}R, b{:}D\}$ we know that the individual a must be contained in the expression $(C \sqcap \exists R.D)$. This process can be carried on for different levels of role assertions (e.g. b can be related to a third individual c or even to a itself); we decided to build the MSC by stopping when a cycle is detected.

Now, let us assume that we calculated the MSC associated to the element a and we call it MSC_a. If we are interested in verifying whether the individual a is in the extension of the concept C in every interpretation, we can first check if MSC_a is subsumed by C or by $\neg C$ before performing any precompletion. In fact, if MSC_a is subsumed by C then a is an instance of C, while if it is subsumed by $\neg C$ we know for sure that a cannot be an instance of C in any interpretation satisfying the KB.

8.2.7 Other techniques

Query caching If an instance checking like $\langle \mathcal{T}, \mathcal{A}\rangle \models a{:}C$ is verified, the assertion $a{:}C$ is a logical consequence of the KB; therefore the new KB $\langle \mathcal{T}, \mathcal{A} \cup \{a{:}C\}\rangle$ is equivalent to the original one. In addition, the proof for the original instance checking problem can be rather involved.

The idea behind the instance checking is to add the results of positive instance checking problems as new assertions in the KB. This can be useful in cases in which the same query is issued again, or when a contradiction can be detected earlier because of the new assertions.

Semantic branching Semantic branching is a technique for exploring disjoint branches of the search space, when a nondeterministic rule must be applied (see Horrocks and Patel-Schneider [1999]). When a constraint containing a disjunction constructor (i.e. $a{:}C_1 \sqcup \ldots \sqcup C_n$) must be expanded, the $\sqcup$–rule selects the first disjunct and adds it to the constraint system (i.e. $a{:}C_1$). If the new constraint causes a contradiction, the algorithm backtracks and the second disjunct C_2 is choosen. The process continues until either a satisfiable precompletion is found, or all the possibilities have been exhausted.

This method is called *syntactic branching*, because it follows the syntactic order of the disjuncts in the concept expression. However, the options are not necessarily disjoint, so there is nothing to prevent the recurrence of unsatisfiable constraints in different branches.

We have not implemented semanting branching in the way it is described in Horrocks and Patel-Schneider [1999]. This is because it requires a completely different backtracking mechanism, and we wanted a technique which could be easily activated or deactivated. What we do instead, is a modified version of the syntactic branching where every time a new disjunct is tried, the conjunction of the negation of previous failed choices are added as well. For example, if the first option $a{:}C_1$ fails, the new constraints $a{:}C_2$ and $a{:}\neg C_1$ are tried. If the second choice fails as well, the third option is $a{:}C_3$ with the constraint $a{:}(\neg C_1 \sqcap \neg C_2)$, and so on util the last option. The idea is that the negated concept inserted would cause contradictions to be detected at an early stage, pruning the useless part of the search space.

Unfortunately, this implementation is far from optimal, because it may suffer the overhead of evaluating the newly introduced constraints; in fact, with the tests we used the optimisation did not improve the performance of the system (although, it has not degraded either).

8.3 Experiments

The main purpose of the experiments we performed with the system was to verify the feasability of the approach in terms of performance. We compared our implementation with Race (see Haarslev and Möller [2000a]) because it is the fastest available Abox rasoner, and it provides a superset of $\mathcal{SH}f$ (it has unqualified number restrictions as well). The comparison of Race with other DL systems can be found in Franconi et al. [1998b].

8.3.1 DL benchmark suite

The main problem in performing experiments with an Abox reasoner is the lack of real data (i.e. KBs). For our experiments we used the *synthetic Abox tests* of the DL benchmark suite.[5] The DL benchmark suite is a collection of tests for DL systems which has been introduced for the DL'98 workshop (see Horrocks and Patel-Schneider

[5]It is available at the URL `http://kogs-www.informatik.uni-hamburg.de/~moeller/dl-benchmark-suite.html`.

[1998b]). The test suite was designed according to a structure which has been first used for the Comparison of Theorem Provers for Modal Logics at Tableaux '98 (see de Swart [1998]).

The test suite is mainly oriented towards terminological reasoning, because the tests were originated in the modal logic community. There are 9 classes of tests (**k_branch_n**, **k_d4_n**, **k_dum_n**, **k_grz_n**, **k_lin_n**, **k_path_n**, **k_ph_n**, **k_poly_n**, **k_t4p_n**) each one containing different instances of problems of increasing difficulty. Each instance is automatically generated according to a schema related to the class, and it consists of a Tbox, an Abox and a set of Abox queries.

Each instance is evaluated by loading the knowledge base (i.e. Tbox and Abox), then the KB satisfiability is checked and all the Abox queries are evaluated. If the system is unable to evaluate completely all the queries within a CPU time limit of 500 seconds, the test for the current instance is aborted.

For each class the benchmark starts with the easiest instance (the first) and continues until all the instances for the class are evaluated, or a timeout interrupts the evaluation. The number of the latest instance the system has been able to evaluate is recorded together with the CPU time spent to evaluate it, and a new class of problems is considered.

Abox test instances derive from the concepts generated for the *Concept Satisfiability Tests* (see Horrocks and Patel-Schneider [1998b]). The Tbox is generated by naming all the sub-concepts appearing in the test concept, while the Abox is generated from the model generated by a satisfiability test over the concept itself.

This approach has the advantage of being well defined, so the results of each Abox query is known in advance; this is crucial for verifying the correctness of the DL system. On the other hand, it has several disadvantages that make the tests unsatisfactory as examples of realistic problems. Firstly, the originating concepts contain a single role name, and this is very unlikely in any real KB. The second problem is the terminology. Since all the sub–concepts are recursively named the resulting Tbox is unfoldable (i.e. acyclic and definitional, see Section 2.1.2), and this means that it can be considered as an empty terminology. The last problem of the synthetic Abox tests is the fact that Aboxes are built starting from a model generated by a tableaux based DL system for a concept. The kind of models these systems generate are not unstructured, but they are always tree shaped and mostly fully connected. Obviously they cannot be taken as a representative sample for generic KBs.

As we are going to show in the next section, the inadequacy of the data tests is highlighted by some of the results of our experiments. Unfortunately, these synthetic Abox tests are the only coherent suite for testing DL systems, so we must make the most of them. In performing the experiments we were not interested in the absolute performance, but on the actual behaviour of the system with different optimisation features activated. However, the results must be considered in the light of their limitations. We do not think that the available Abox tests reflect realistic KBs, so they cannot give an estimate of the size/type of KB our system can handle.

8.3.2 Measurements

Our objective is to etablish a correlation between the performance of the system on different tests and the behaviour of the optimization techniques. Some of the different optimisations mentioned in Section 8.2 can be turned on and off by flags, and several counters have been placed in crucial parts of the code to obtain a picture of the behaviour of the system during the evaluation of a tests. For example, we counted the number of times the knowledge base is precompleted, in contrast with the times the MSC test is sufficient to answer an instance check. Another counter registers the number of backjumps which have been performed.

We assumed that each class of tests presents a different pattern, therefore we took separate measurements for each class. The results are summarised in Table 8.2, Table 8.3, and Table 8.4.

Results of the experiments are grouped by the class of tests, and each row shows the results for the system with a particular set of optimisations. As a reference, the first row of each class shows the results of the Race system. The following rows contain the results obtained by our system with different configuration of optimisation techniques. The results obtained by our system appear in decreasing order of performance, so the first row contains always the best result, and the last the worse. We chosen this order because it highlights the impact of the different optimisations on the performance of the system.

All the experiments have been performed on a machine equipped with a Celeron 450 MHz processor, with 256 Mb of main memory and running Linux with the kernel version 2.2.17. Both the systems are written in Common Lisp and run using Allegro Common Lisp version 5.0.1.

We will first explain the meaning of each column in the tables, then we will try to draw some conclusions from the results of the experiments.

The System field shows the active optimisations with our system (aFaCT) or the version of the Race system that has been used. The six different optimisations which can be activated are[6]

LABEL-MODAL-CHECK enable the modal verification discussed in Section 8.2.6;

CHECK-BEFORE-PRECOMP enable the verification of the satisfiability of the label of an individual *before* starting to precomplete it (the verification at the end is always performed);

MSC-CHECKING the Most Specific Concept of an individual is build and used for the instance checking before the actual precompletion of the Abox;

BACKJUMPING enable the backjumping;

QUERY-CACHING enable query caching (see Section 8.2.7);

SEMANTIC-BRANCHING activate the semantic branching (see Section 8.2.7).

As we said above, the configuration opt6 has all these optimisation switches activated, then at each different configuration one of the switches is turned off. So in the configuration opt5 the MSC-CHECKING is turned off, in opt4 the QUERY-CACHING, in opt3 the SEMANTIC-BRANCHING, in opt2 the CHECK-BEFORE-PRECOMP, in opt1 the LABEL-MODAL-CHECK, and finally in opt0 the BACK-JUMPING is turned off as well.

We could have tried every combination of the six flags, but it would have taken too much time to run, and a volume of result data rather unmanageable. Therefore we decided to start with all the optimisation turned on, and disable a feature at each step. This gives us seven different configuration of optimisations, named from opt6 to opt0. With opt6, all the optimisations are active, while with opt0 they are all switched off.

In the configuration opt5 the MSC-CHECKING is turned off, in opt4 the QUERY-CACHING, in opt3 the SEMANTIC-BRANCHING, in opt2 the CHECK-BEFORE-PRECOMP, in opt1 the LABEL-MODAL-CHECK, and finally in opt0 the BACK-JUMPING is turned off as well. The order is based on the our experience with the system, we first eliminated the options that we thought did not impact the performance. The first flag MSC-CHECKING is an exception, because the use of subsumption to perform instance checking is not an optimisation which impacts on the behaviour of the algorithm.

[6]The remaining techniques we mentioned in the previous section are always active.

The next two columns Last Instance and Duration show the last instance of the class that has been solved, and how many seconds of CPU time have been used for solving that instance. The following fields are the value of the different counters.

All-Instance-Checks shows the total number of instance checks that have been performed, while Prec-Instance-Checks indicates the number of them which have been performed by precompletion. Note that unless the MSC-CHECKING is active, these two counters have the same value.

Backjumping counts the number of times the following options of a backtracking point have been ignored because of the backjumping optimisation. The counter is incremented at every backatrack point, so for a single failure more than one backtrack points can be "skipped" (see the example in Section 8.2.4). Therefore its quantitative value is not very significant; however, the order of magnitude gives an idea of how much the backjumping is "active". Its value compared with the number of clashes detected gives an idea of how "deep" the backjumps are (how many backtracking points are ignored in a single backtrack).

Call-Reasoner-Full-Label and Call-Reasoner-Modal-Part count the number of times the terminological reasoner has been invoked to verify the satisfiability of a label. Sat-Label on the other hand counts the total number of times the label has been verified. This number is typically higher than the sum of the two previous ones, because the satisfiability value of the label could have been cached (counters Cached-Clash-Label and Cached-Sat-Label), or a contradiction is immediately detected among the concepts of the label (Simple-Clash or Bottom-Detect, this latter one detects the presence of an explicit $\perp$ contraint in the label, usually added by a concept normalisation.) Finally, the counter Label-Clash shows haw many times the verification of the satisfiability of a label returned a failure.

8.3.3 Notes on results

The first thing worth noticing is the fact that most of the instance checking tests can be solved by terminological reasoning only, without precompleting the KB. This can be observed by considering the value of the counter Prec-Instance-Checks when the system has the MSC-CHECKING enabled (configuration opt6). In all classes except **k_d4_n** and **k_grz_n** all the instance checks are solved without precompleting the Abox, and even in those two classes over 98% of the instance checks are solved

Dataset	System	Last Instance	Duration	All-Instance-Checks	Prec-Instance-Checks	Backjumping	Call-Reasoner-Full-Label	Call-Reasoner-Modal-Part	Sat-Label	Label-Clash	Cached-Clash-Label	Cached-Sat-Label	Simple-Clash	Bottom-Detect
k_branch_n	race 1.1r9	3	159.21											
	aFaCT (opt5)	2	4.82	191	191	47150	166	15876	16074	15949	0	32	76515	0
	aFaCT (opt4)	2	8.39	191	191	46854	184	15715	15935	15766	0	36	75064	0
	aFaCT (opt3)	2	8.67	191	191	53515	184	18920	19140	18972	0	36	91257	0
	aFaCT (opt6)	2	9.83	191	0	52720	46	18575	18651	18547	0	30	90105	0
	aFaCT (opt2)	2	15	191	191	54223	0	19330	19342	19251	0	12	93011	0
	aFaCT (opt1)	2	247.81	191	191	485	29886	0	29898	29811	0	12	101606	0
	aFaCT (opt0)	1	0.78	51	51	0	16479	0	16491	15932	0	12	69393	0
k_d4_n	race 1.2	2	19.57											
	aFaCT (opt6)	2	7.05	107	1	24100	2418	69645	72958	66582	0	895	140880	0
	aFaCT (opt5)	2	63.76	107	107	21164	3010	68094	72262	64472	0	1158	140220	0
	aFaCT (opt4)	2	81.21	107	107	28622	4289	94608	100434	89689	0	1534	191498	2
	aFaCT (opt3)	2	81.35	107	107	26903	4144	84725	90360	79986	0	1489	168306	2
	aFaCT (opt2)	2	101.5	107	107	8155	0	32585	32621	30954	0	34	65249	2
	aFaCT (opt1)	2	106.51	107	107	2655	32814	0	32850	31441	0	34	65380	2
	aFaCT (opt0)	1	0.25	28	28	0	84829	0	84863	68165	0	34	109850	0
k_dum_n	race 1.2	13	383.31											
	aFaCT (opt6)	16	177.79	2105	0	1585	748	66047	67528	65397	0	733	71754	0
	aFaCT (opt3)	10	120.8	709	709	128487	11867	132468	154724	113318	21	10368	117626	0
	aFaCT (opt4)	10	123.41	706	706	121973	11339	118418	139571	100346	21	9793	104243	0
	aFaCT (opt1)	10	320.07	698	698	38913	179444	0	179634	164302	0	190	91002	0
	aFaCT (opt5)	8	190.19	418	418	69999	5692	85812	96105	77357	5	4596	108043	0
	aFaCT (opt2)	2	0.83	51	51	1358	0	502	530	270	0	28	315	0
	aFaCT (opt0)	1	0.98	22	22	0	9554	0	9566	7527	0	12	3919	0

Table 8.2: Experiments summary

Dataset	System	Last Instance	Duration	All-Instance-Checks	Prec-Instance-Checks	Backjumping	Call-Reasoner-Full-Label	Call-Reasoner-Modal-Part	Sat-Label	Label-Clash	Cached-Clash-Label	Cached-Sat-Label	Simple-Clash	Bottom-Detect
k_grz_n	race 1.2	10	31.49											
	aFaCT (opt6)	8	94.38	890	15	297128	13214	53772	102736	25080	0	16133	12543	19617
	aFaCT (opt3)	7	213.86	648	648	532831	31807	147167	237731	87655	0	29810	153477	28947
	aFaCT (opt4)	7	217.59	656	656	759159	38125	186553	301712	112041	0	38531	197193	38503
	aFaCT (opt5)	7	253.73	688	688	417621	17823	80709	139834	44564	0	19982	61967	21320
	aFaCT (opt2)	7	275.08	649	649	685282	0	177157	208583	112345	0	196	239458	31230
	aFaCT (opt1)	6	214.32	460	460	59124	61327	0	64882	47533	0	157	66190	3398
	aFaCT (opt0)	2	147.08	61	61	0	148493	0	156337	94691	0	30	229499	7814
k_lin_n	race 1.2	4	395.28											
	aFaCT (opt3)	4	50.48	93	93	0	76	12232	12369	12234	0	61	60370	0
	aFaCT (opt4)	4	51.67	93	93	0	76	12782	12919	12785	0	61	63101	0
	aFaCT (opt2)	4	52.32	93	93	84	0	14283	14309	14239	0	26	70352	0
	aFaCT (opt6)	4	53.51	93	0	0	44	14881	14986	14872	0	61	73405	0
	aFaCT (opt5)	4	54.32	93	93	0	76	13525	13662	13530	0	61	66774	0
	aFaCT (opt1)	3	25.81	77	77	0	17857	0	17883	17821	0	26	14117	0
	aFaCT (opt0)	3	28.03	77	77	0	14345	0	14371	14309	0	26	13922	0
k_path_n	race 1.2	3	199.12											
	aFaCT (opt3)	4	122.83	3147	3147	10805	2762	2528	6139	4507	0	849	5788	0
	aFaCT (opt4)	4	127.4	3147	3147	10805	2762	2528	6139	4507	0	849	5788	0
	aFaCT (opt5)	4	135.16	3147	3147	10805	2762	2528	6139	4507	0	849	5788	0
	aFaCT (opt2)	3	114	1921	1921	166280	0	28724	29241	28227	0	402	53397	115
	aFaCT (opt6)	3	135.82	1455	0	10805	611	2528	3988	2364	0	849	4784	0
	aFaCT (opt0)	1	9.21	140	140	0	12175	0	15297	15172	0	50	6031	3072
	aFaCT (opt1)	1	9.46	141	141	30	14075	0	14128	14003	0	50	6029	3

Table 8.3: Experiments summary

Dataset	System	Last Instance	Duration	All-Instance-Checks	Prec-Instance-Checks	Backjumping	Call-Reasoner-Full-Label	Call-Reasoner-Modal-Part	Sat-Label	Label-Clash	Cached-Clash-Label	Cached-Sat-Label	Simple-Clash	Bottom-Detect
k_ph_n	race 1.2	5	1.7											
	aFaCT (opt0)	6	2.09	391	391	0	55	0	95	0	0	40	539	0
	aFaCT (opt1)	6	2.15	391	391	0	55	0	95	0	0	40	539	0
	aFaCT (opt2)	6	2.67	391	391	0	0	55	95	0	0	40	539	0
	aFaCT (opt5)	6	6	391	391	0	131	12	226	76	0	83	361	0
	aFaCT (opt4)	6	6.12	391	391	0	131	12	226	76	0	83	361	0
	aFaCT (opt3)	6	6.48	391	391	0	131	12	226	76	0	83	361	0
	aFaCT (opt6)	6	45.78	391	0	0	55	12	150	0	0	83	46	0
k_poly_n	race 1.2	4	120.33											
	aFaCT (opt2)	4	3.32	1088	1088	592	0	1864	2000	528	0	136	1274	0
	aFaCT (opt0)	4	3.47	1088	1088	0	1864	0	2000	528	0	136	1866	0
	aFaCT (opt1)	4	3.6	1088	1088	592	1864	0	2000	528	0	136	1274	0
	aFaCT (opt4)	4	4.69	1088	1088	592	1760	1332	3476	648	0	384	1096	0
	aFaCT (opt3)	4	4.76	1088	1088	592	1760	1332	3476	648	0	384	1096	0
	aFaCT (opt5)	4	8.14	1088	1088	296	1576	1000	2812	664	0	236	784	0
	aFaCT (opt6)	4	44.72	1088	0	0	192	256	648	0	0	200	64	0
k_t4p_n	race 1.2	2	118.46											
	aFaCT (opt6)	5	296.21	5690	0	260	2514	2718	7359	349	0	2127	416	0
	aFaCT (opt2)	2	129.61	529	529	56501	0	102185	102582	23115	0	397	56886	0
	aFaCT (opt1)	2	180.51	525	525	46703	89914	0	90386	20948	1	471	51297	0
	aFaCT (opt5)	2	273.26	525	525	33525	50124	92022	153410	38327	1	11263	75139	0
	aFaCT (opt4)	1	2.29	262	262	23192	58025	88223	158457	26352	1	12208	89069	0
	aFaCT (opt3)	1	2.59	207	207	11925	33499	56562	98614	20235	1	8552	76156	0
	aFaCT (opt0)	0	0	52	52	0	284484	0	828210	163266	93378	450348	760	0

Table 8.4: Experiments summary

by using the MSC.[7] This is a clear indication that the tests are not adequate for fully evaluating an Abox reasoner.

Note that using the MSC is not always the best way (in terms of efficiency) of performing the instance checking. Although, in some cases it provides remarkable good results (e.g. **k_dum_n** and **k_t4p_n**), in other it actually manifests a significant loss of performance (e.g. **k_poly_n** and **k_ph_n**).

Unsurprisingly, no particular configuration of optimizations is best in all experiments. Different classes of problems seem to require different optimisations for better performance, and this may suggests that a good approach could be an adaptive system which activates or deactivates some optimizations according to the behaviour of the system with a given KB.

Having said that, the nondeterminism is definitely the main source of slow down of the system, and optimisations that are directed at reducing the nondeterminism (such as backjumping) seem to provide uniformily better results. This is clearly indicated by the fact that the "activeness" of the backjumping (counter Backjumping) is one of the few indicators that can be clearly related to the performance (see for example **k_branch_n**, but the same pattern can be spotted in other classes). Moreover, as predicted in Section 8.2.4, the backjumping alone (configuration opt1) does not seems to provide good results without the LABEL-MODAL-CHECK activated as well. This is clearly indicated by the ratio of the two counters Call-Reasoner-Full-Label and Call-Reasoner-Modal-Part w.r.t. the counter Backjumping.

The rest of the counters do not seem to show a direct correlation with performance. This probably indicates that the optimisations they are monitoring do not significantly influence the behaviour of the system (at least with the available Abox tests).

The last thing we would like to mention is the fact that we really did not expect our system to be faster than the Race system; even in a few classes. We had this impression because we know that Race is no slower than FaCT as a terminological reasoner, and it uses the very same engine for both Abox and terminological reasoning. If you look closely at the precompletion technique, it uses transformation rules which are very similar to the ones implemented in Race; the difference lies the fact that our system made a sharp distiction between rules working with the Abox (the precompletion rules) and rules working at the terminological level (delegated to the terminological reasoner FaCT).

[7]The reader may be puzzled by the fact that the rest of the counters are not equal to zero; this is explained by the fact that the satisfiability of the KB is verified before performing any instance checking tests.

A system without this separation (like Race) has the possibility to maximise the effect of optimisations like backjumping because it has better information about constraints causing contradictions (something we tried to simulate by verifying the modal part of the label only). We think that with classes like **k_branch_n** it is this fact that makes the real difference in performance. In addition, Race is a more mature system and it has more optimisation techniques implemented (for example model merging, see Haarslev and Möller [2000a]).

For these reasons, we did not expect to outperform Race for any test having a small and almost fully connected Abox, where partitioning and memory occupation do not make any difference. Obviously, it could be the case that some of the results exhibited by our system depend on the peculiar structure of the test problems. However, if this behaviour is exhibited even with general KBs, it means that there is some unnecessary overhead in the hybrid reasoning in Race which can be reduced by separating the treatment of Abox and "terminological" constraints. This would suggest that the performance of tableaux–based algorithms, such as the one implemented in Race, can be improved by using an evaluation strategy more similar to the precompletion strategy (e.g. work on individual first).

Chapter 9

Conclusions

This chapter summarise the results presented in this thesis, and suggests some directions for future work.

9.1 Thesis contributions

The aim of our work is to study basic reasoning techniques for DL knowledge bases with Abox, and to determine whether they can lead to practical tractability. This thesis investigates two reasoning problems, namely KB satisfiability and query answering.

9.1.1 KB satisfiability

Different techniques have been considered, and we chose to investigate the possibility of extending the precompletion technique in order to provide an algorithm for the DL $\mathcal{SH}f$.

By exploiting the tree–like model property of $\mathcal{SH}f$, it has been proved that the technique leads to a correct and complete algorithm for KB satisfiability. An experimental DL reasoner based on the studied algorithm has been developed, and some experiments have been performed to evaluate the behaviour of the system.

The experiments provided evidence of the fact that, even with an optimised terminological reasoner, different optimisation techniques must be used in the precompletion phase in order to obtain acceptable performance. Although we have not provided formal proofs of correctness and completeness of the optimisations used, we have provided arguments supporting the fact that their use does not affect the correctness and completeness of the underlying algorithm.

The system developed, although experimental, exhibits good performance compared to a state of the art DL system. As pointed out in Chapter 8, we think that the ideas behind the precompletion technique may be used to improve the performance of tableaux based systems like Race.

9.1.2 Query answering

The language for querying KBs has always been one of the weakest points of DL systems; recently, there have been some attempts to find a solution to this inadequacy. In this thesis we develop a technique for answering conjunctive queries over $\mathcal{SH}f$ KBs.

Again, the technique relies on a careful use of the model–theoretic properties of $\mathcal{SH}f$. A class of interpretations has been identified which is complete w.r.t. the problem of conjunctive query answering in $\mathcal{SH}f$. The properties of this class have been used to devise an algorithm for query answering, and for proving its correctness and completeness.

Although the results which have been presented here are specific to the DL $\mathcal{SH}f$, the technique is general enough to provide query answering algorithms for a wide range of DLs. In addition, since the problem is reduced to KB satisfiability, the algorithm described can easily be implemented on top of most of the available DL systems.

9.2 Future work

There are several research directions which stem from the work presented in this thesis.

KB satisfiability On the KB satisfiability front it is natural to ask ourselves whether the precompletion technique can be extended to more expressive DLs. We think that this technique can easily be adapted to most of the DLs having the tree–model property (or a tree–like property like the one detailed in Chapter 3). For example, the addition of (qualified) number restrictions should not be too difficult. On the other hand, there are DL constructors that appear to be much more difficult (or even impossible) to handle in the precompletion framework as, for example, the inverse role constructor.

The experiments with the developed DL system show that there is still the necessity to produce an adequate test suite for Abox reasoners. The fact that DL systems providing Abox reasoning are becoming fast enough to be used in applications may encourage the production of such tests.

Since the experiments show that different techniques must be incorporated in the basic algorithm to obtain acceptable performance, there is still some theoretical work to do. We provided arguments supporting the correctness and completeness of the modified algorithm, but ideally this should be shown by means of formal proofs.

Query answering Concerning the query answering problem, there is still a long way to go before having usable systems. Moreover, there are still some restrictions on the query language for $\mathcal{SH}f$ which should be lifted (if possible). Some conjectures about the solution to these problems have been made, and they will be investigated in the near future.

Providing a completeness result for the complexity of query answering is an important task that would provide a useful theoretical tool for evaluating query algorithms.

Before the actual implementation of an experimental system, we think that the possibility of optimising the nondeterminism in the query transformation must be carefully considered. In fact, although the algorithm as it is can probably handle small Aboxes, it is not suitable for large ones. In addition, the experience with the results obtained by optimising the terminological and hybrid reasoning encourages research in this promising field.

Bibliography

S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

C. Areces, H. de Nivelle, and M. de Rijke. Prefixed resolution: A resolution method for modal and description logics. In *Proceedings of CADE 99*, pages 187–201, January 1999.

F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 446–451. Morgan Kaufmann, 1991.

F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *Proceedings of the Workshop on Processing Declarative Knowledge*, pages 67–86, 1991a.

F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991b.

F. Baader and U. Sattler. Number restrictions on complex roles in description logics: A preliminary report. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proc. of the fifth International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Massachusetts, USA, 1996. Morgan Kaufmann.

A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

A. Borgida. On the relationship between description logic and predicate logic. In *Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), Gaithersburg, Maryland, November 29 - December 2, 1994*, pages 219–225. ACM, 1994.

A. Borgida and P. F. Patel-Schneider. A semantic and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1, 1994.

M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1: 109–138, 1993.

D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS-98)*, 1998a.

D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views over description logics knowledge bases. In *Proceedings of the 16th National Conference on Artificial Intelligence (AAAI 2000)*, pages 386–391, 2000.

D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publisher, 1998b.

A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Conference Record of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

G. De Giacomo. Eliminating "converse" from converse pdl. *Journal of Logic, Language and Information*, 5:193–208, 1996.

G. De Giacomo and M. Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In AAAI-Press/MIT-Press, editor, *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI'94)*, pages 205–212, 1994.

G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327. Morgan Kaufmann Publishers, 1996.

G. De Giacomo and M. Lenzerini. A uniform framework for concept definitions in description logics. *Journal of Artificial Intelligence Research*, 6:87–110, 1997.

G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, 1998.

H. de Swart, editor. *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in LNAI, 1998. Springer.

F. Donini, G. De Giacomo, and F. Massacci. EXPTIME tableaux for $\mathcal{ALC}$. In L. Padgham, E. Franconi, M. Gehrke, D. L. McGuinness, and P. F. Patel-Schneider, editors, *Collected Papers from the International Description Logics Workshop (DL'96)*, number WS-96-05 in AAAI Technical Report, pages 107–110. AAAI Press, Menlo Park, California, 1996a.

F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996b.

F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4(4): 423–452, 1994.

Francesco M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. *Information and Computation*, 134(1):1–58, 1997.

Q. Elhaik and M. C. Rousset. Making an abox persistent. In Franconi et al. [1998b].

A. Era and F. Donini. Most specific concepts for knowledge bases with incomplete information. In *International Conference on Information and Knowledge Management, Baltimore*, November 1992.

E. Franconi. Description logics for natural language processing. In *Working Notes of the 1994 AAAI Fall Symposium on Knowledge Representation for Natural Language Processing in Implemented Systems*, 1994.

E. Franconi, G. De Giacomo, I. R. Horrocks, D. L. McGuinness, W. Nutt, P. F. Patel-Schneider, and C. A. Welty. Report on the 1998 intl. workshop on description logics (dl'98), 1998a.

Enrico Franconi, G. De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors. *Proceeding of the 1998 International Workshop on Description Logics (DL'98)*, 1998b. CEUR Pubblication.

E. Grädel. Why are modal logics so robustly decidable? *Bulletin of the European Association for Theoretical Computer Science*, 68:90–103, 1999.

V. Haarslev and R. Möller. An empirical evaluation of optimization strategies for abox reasoning in expressive description logics. In Lambrix et al. [1999], pages 115–119.

V. Haarslev and R. Möller. Consistency testing: The race experience. In *Proceedings of TABLEAUX 2000*, volume 1847 of *Lecture Notes in Computer Science*, pages 57–61. Springer, 2000a.

V. Haarslev and R. Möller. Expressive abox reasoning with number restrictions, role hierarchies, and transitively closed roles. In A.G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 273–284. Morgan Kaufmann, 2000b.

V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases. In Franz Baader and Ulrike Sattler, editors, *Proceedings of the 2000 International Workshop on Description Logics (DL2000)*, pages 143–152, 2000c.

B. Hollunder. Consistency checking reduced to satisfiability of concepts in terminological systems. *Annals of Mathematics and Artificial Intelligence*, 18:95–131, 1996.

I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, 1998.

I. Horrocks and P. F. Patel-Schneider. Comparing subsumption optimizations. In Franconi et al. [1998b].

I. Horrocks and P. F. Patel-Schneider. Dl system comparison. In Franconi et al. [1998b].

I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. In Franconi et al. [1998b].

I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. Query containment using a DLR ABox. LTCS-Report 99-15, LuFG Theoretical Computer Science, RWTH Aachen, Germany, 1999a.

I. Horrocks, U. Sattler, S. Tessaris, and S. Tobies. How to decide query containment under constraints using a description logic. In *Logic for Programming and Automated Reasoning (LPAR 2000)*, volume 1955 of *Lecture Notes in Computer Science*, pages 326–343. Springer, 2000a.

I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, 1999b.

I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David MacAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes In Artificial Intelligence, pages 482–496. Springer-Verlag, 2000b.

I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *National Conference on Artificial Intelligence (AAAI 2000)*, pages 399–404. American Association for Artificial Intelligence, 2000.

I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proceedings of the 7th International Conference on the Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 285–296, 2000.

U. Hustadt and R. A. Schmidt. Issues of decidability for description logics in the framework of resolution. In R. Caferra and G. Salzer, editors, *Automated Deduction in Classical and Non-Classical Logics*, volume 1761 of *Lecture Notes in Artificial Intelligence*, pages 191–205. Springer, 2000.

KRSS. Description-logic knowledge representation system specification, November 1993.

Patrick Lambrix, Alex Borgida, M. Lenzerini, Ralf Möller, and P. Patel-Schneider, editors. *Proceeding of the 1999 International Workshop on Description Logics (DL'99)*, 1999. CEUR Pubblication.

A. Y. Levy and M. C. Rousset. Carin: A representation language combining horn rules and description logics. In *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI-96)*. John Wiley and Sons, 1996a.

A. Y. Levy and M. C. Rousset. The limits on combining recursive horn rules and description logics. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence 1996*, 1996b.

C. Lutz and U. Sattler. The complexity of reasoning with boolean modal logic. In *Advances in Modal Logic 2000 (AiML 2000)*, Leipzig, Germany, 2000.

R. M. MacGregor and D. Brill. Recognition algorithms for the LOOM classifier. In *Proceedings of AAAI-92*, pages 774–779. AAAI Press, 1992.

B. Nebel. Terminological cycles: Semantics and computational properties. In J. Sowa, editor, *Principles of Semantic Networks*. Morgan Kaufmann, 1991.

P. F. Patel-Schneider. DLP system description. In Franconi et al. [1998b], pages 87–89.

R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76, 1977.

R. Reiter. Towards a logical reconstruction of relational database theory. In Michael L. Brodie, John Mylopoulos, and Joachim W. Schmidt, editors, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Topics in Information Systems, pages 191–233. Springer, 1984.

M. C. Rousset. Backward reasoning in aboxes for query answering. In Enrico Franconi and Michael Kifer, editors, *Knowledge Representation meets Databases (KRDB'99)*, volume CEUR-WS/Vol-21. CEUR, 1999.

U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in Lecture Notes in Artificial Intelligence. Springer Verlag, 1996.

A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 466–471, 1991.

M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

R. S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1/2):121–141, July/August 1982.

T. Tammet. Using resolution for extending kl-one-type languages. In *CIKM '95, Proceedings of the 1995 International Conference on Information and Knowledge Management, November 28 - December 2, 1995, Baltimore, Maryland, USA*, pages 326–332. ACM, 1995.

S. Tessaris and G. Gough. Abox reasoning with transitive roles and axioms. In Lambrix et al. [1999].

M. Y. Vardi. Why is modal logic so robustly decidable? Technical Report TR97-274, Rice University, 1997.

W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computer and Mathematics with Applications, special issue: Semantic Networks in Artificial Intelligence*, 23 (2-5):133–177, March-May 1992.