Efficient Instance Retrieval over Semi-Expressive Ontologies

Vom Promotionsausschuss der
Technischen Universität Hamburg-Harburg
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

genehmigte Dissertation

von

Sebastian Wandelt

aus Berlin

2011

1. Reviewer: Ralf Möller, Hamburg University of Technology

2. Reviewer: Ian Horrocks, University of Oxford

3. Reviewer: Norbert Ritter, University of Hamburg

Day of the defense: 06.10.2011

Abstract

In the last years, the vision of the Semantic Web fostered the interest in reasoning over growing sets of assertional statements in ontologies. Traditional tableau-based reasoning systems have problems to answer queries over large ontological data sets because these reasoning systems are based on efficient use of main memory data structures. Increasing expressivity and worst-case complexity further tighten the memory burden. The purpose of this thesis was to investigate how to release the main memory burden from tableau-based reasoning systems and perform efficient instance checking and instance retrieval over semi-expressive ontologies.

The key idea was to reduce instance checking for an individual in an ontology to smaller subsets of relevant axioms. Modularization techniques were introduced and further refined in order to reduce the module size. Instance retrieval performance was addressed by defining similarity criteria over individuals and their modules. Finally, this thesis investigated techniques to preserve modularizations under syntactic ontology updates.

For evaluation purposes, experiments on benchmark and real world ontologies were carried out. Modularization techniques gave rise to a distributed implementation for solving instance checking and retrieval problems. The principal conclusion is that the main memory dependency for instance checking and instance retrieval can be released from tableau-based reasoning systems for semi-expressive ontologies in practice.

This page is intentionally left blank.

Contents

List of Figures

List of Symbols

In the following list of symbols, each line contains the symbol notation, the name and a reference for the page, where the symbol is defined.

- Chapter 2, Section 2.1:

$\mathbf{S}$	Set (page 8)
$\wp(\mathbf{S})$	Powerset of $\mathbf{S}$ (page 8)
$\mathbf{R}$	Relation (page 8)
f	Function (page 9)
$FD(f)$	Used domain of f (page 9)
$FVAL(f)$	Range of f (page 9)
$n.d.$	Function value is not defined (page 9)
f^-	Inverse of f (page 9)
$f\!\restriction^{\mathbf{S}}$	Domain restriction on f (page 9)
$\mathbf{MS}$	Multiset (page 10)
Ξ	Empty multiset (page 10)
$\uparrow_S^M(\mathbf{S})$	Transformation of sets to multisets (page 10)
$\downarrow_S^M(\mathbf{MS})$	Transformation of multisets to sets (page 10)
L	List (page 10)
$\square$	Empty list (page 10)
$\mathbb{T}$	Tree (page 11)
n	Node (page 11)
$root$	Root node of a tree (page 11)
$\mathbb{G}$	Graph (page 11)
ϕ	Node labeling function (page 11)
σ	Edge labeling function (page 11)

- Chapter 2, Section 2.2:

$\mathbf{CN}$	Set of concept names (page 13)
$\mathbf{RN}$	Set of role names (page 13)
$\mathbf{NIN}$	Set of named individuals (page 13)
$\mathbf{AIN}$	Set of anonymous individuals (page 13)
$\mathbf{IN}$	Set of individuals (page 13)
$\mathcal{I}$	Interpretation (page 13)
R	Role description (page 14)
$\mathbf{Rol}$	All role descriptions (page 14)
C	Concept description (page 14)
$\mathbf{Con}$	All concept descriptions (page 14)
$\mathbf{AtCon}$	Set of atomic concept descriptions (page 15)

$clos(C)$	Concept closure of C (page 16)
$nnf(C)$	Negation normal form of C (page 16)
$\mathcal{T}$	TBox (page 19)
ST	Set of TBoxes (page 19)
$\mathcal{R}$	RBox (page 19)
SR	Set of RBoxes (page 19)
$\mathcal{A}$	ABox (page 19)
SA	Set of ABoxes (page 19)
$Ind(\mathcal{A})$	Individuals occurring in $\mathcal{A}$ (page 19)
$NInd(\mathcal{A})$	Named individuals occurring in $\mathcal{A}$ (page 19)
$AInd(\mathcal{A})$	Anonymous individuals occurring in $\mathcal{A}$ (page 19)
$\mathcal{O}$	Ontology (page 19)
SO	Set of ontologies (page 19)
$clos(\mathcal{T})$	Concept closure of $\mathcal{T}$ (page 20)
$\underline{rc}^{\mathcal{O}}$	Role classification for $\mathcal{O}$ (page 24)
$\underline{rtc}^{\mathcal{O}}$	Role transitivity classification for $\mathcal{O}$ (page 24)
$\underline{cc}^{\mathcal{O}}$	Concept classification for $\mathcal{O}$ (page 24)
$\underline{ir}^{\mathcal{O}}_{C}$	Instance retrieval result for $\mathcal{O}$ and C (page 24)
$\underline{rr}^{\mathcal{O}}_{R}$	Relation retrieval result for $\mathcal{O}$ and R (page 24)
$tabrapp_{X,\mathcal{T},\mathcal{R}}$	Tableau rule application (page 28)
π	Variable Assignment (page 28)
$\mathbf{tabrapps}_{,\mathcal{T},\mathcal{R}}$	Set of all tableau rule applications (page 28)
$T^{\mathcal{O}}$	Tableau for $\mathcal{O}$ (page 33)
$TP^{\mathcal{O}}$	Tableau proof for $\mathcal{O}$ (page 34)
RUN	Tableau run (page 36)

- Chapter 3, Section 3.1:

$Ind(RUN)$	Tableau run individuals (page 44)
$RUN_1 \circ RUN_2$	Tableau run composition (page 42)
$\mathcal{A}[a_1 \to a_2]$	Renaming of ABox individuals (page 45)
$\pi[a_1 \to a_2]$	Variable assignment individual renaming (page 45)
$tabrapp^{\pi,Y}_{X,\mathcal{T},\mathcal{R}}[a_1 \to a_2]$	Tableau rule application individual renaming (page 45)
$RUN[a_1 \to a_2]$	Tableau run individual renaming (page 45)
$S[a_1, ..., a_n \to b_1, ..., b_n]$	Consecutive individual renaming (page 45)
$RUN^{+\mathcal{A}_{ext}}$	Tableau run assertion extension (page 49)
M	ABox modularization (page 40)

- Chapter 3, Section 3.2:

$\mathbb{G}^{\mathcal{A}}$	ABox graph of $\mathcal{A}$ (page 50)
$MC^{\mathcal{A}}$	Component-based ABox modularization (page 50)

- Chapter 3, Section 3.3:

$info^{\forall}_{\mathcal{T}}$	$\forall$-info structure for $\mathcal{T}$ (page 54)
$\downarrow^{R(a,b)}_{c,d}$	ABox split (page 55)
$extinfo^{\forall}_{\mathcal{T},\mathcal{R}}$	Extended $\forall$-info structure for $\mathcal{T}$ and $\mathcal{R}$ (page 62)

- Chapter 4, Section 4.1:

ISL_a	Individual island for a (page 73)
$islandmap^{\mathcal{O}}$	Individual island map for $\mathcal{O}$ (page 76)

- Chapter 4, Section 4.2:

$\mathbb{IIG}_a$	Individual island graph (page 78)
$a_1 \xrightarrow{\rho}_{\mathbb{G}^{\mathcal{A}}} a_2$	ABox graph successor (page 79)
$a_1 \xrightarrow{\rho}_{\mathbb{IIG}_a} a_2$	Individual island graph neighbor (page 79)
θ	Individual island graph homomorphism (page 79)
$\mathcal{I}_\theta$	Homomorphism interpretation (page 80)

- Chapter 4, Section 4.3:

$pns^{a,\mathcal{A}}$	Pseudo node successor of a in $\mathcal{A}$ (page 83)
$osn^{a,\mathcal{A}}$	One-step node of a in $\mathcal{A}$ (page 84)
OSN	Set of all one-step nodes (page 84)
$ABox^{a_2}(pns^{a,\mathcal{A}})$	Pseudo node successor ABox realization (page 85)
$ABox(osn^{a,\mathcal{A}})$	One-step node ABox realization (page 85)
θ	One-step node homomorphism (page 86)

- Chapter 5, Section 5.1:

upd	Syntactic ontology update (page 96)
$\mathcal{OS}$	Ontology state (page 97)
$history$	List of syntactic ontology updates (page 97)
$\mathcal{OS} \uparrow upd$	Ontology state update (page 97)

- Chapter 5, Section 5.2:

$\alpha^{stcs}_{\mathcal{OS}}$	Sound TBox classification structure (page 99)
$\alpha^{stds}_{\mathcal{OS}}$	Sound TBox disjointness structure (page 99)
$\alpha^{cfis}_{\mathcal{OS}}$	Complete $\forall$-info structure (page 99)
$\alpha^{crcs}_{\mathcal{OS}}$	Complete RBox classification structure (page 99)
$\alpha^{crts}_{\mathcal{OS}}$	Complete RBox transitivity structure (page 99)
$asds_{\mathcal{OS}}$	Abstract split decision system (page 100)

- Chapter 5, Section 5.3:

$\mathbf{occ}\,[C_1 \sqsubseteq C_2]$	Obvious classification consequences (page 103)
$stclss\,[\mathcal{OS}]$	Updatable sound TBox classification snapshot (page 104)
$\beta_{\mathcal{OS}}^{stcs}$	Updatable sound TBox classification structure (page 105)
$\mathbf{odc}\,[C_1 \sqsubseteq C_2]$	Obvious disjointness consequences (page 108)
$\beta_{\mathcal{OS}}^{stds}$	Updatable sound TBox disjointness structure (page 109)
$\mathbf{fac}\,[C_1 \sqsubseteq C_2]$	$\forall$-concept description closure (page 110)
$\beta_{\mathcal{OS}}^{cfis}$	Updatable $\forall$-info structure (page 111)
$\beta_{\mathcal{OS}}^{crcs}$	Updatable complete RBox classification structure (page 112)
$\beta_{\mathcal{OS}}^{crts}$	Updatable complete RBox transitivity structure (page 113)

- Chapter 5, Section 5.4:

$updsds(\mathcal{OS})$	Updatable split decision system (page 113)
$spls(updsds(\mathcal{OS}))$	Split set (page 114)
$sdf(updsds(\mathcal{OS}), upd)$	Update split difference bound (page 114)
$\beta_{\mathcal{OS}}^{aboxspl}$	Updatable complete split structure (page 117)

- Chapter 5, Section 5.5:

$\beta_{\mathcal{OS}}^{osnmap}$	Updatable one-step node map structure (page 118)
$\beta_{\mathcal{OS}}^{islmap}$	Updatable island map structure (page 120)

Chapter 1: Introduction

1.1 Reasoning in the Semantic Web

The Semantic Web is intended to bring structure to the meaningful content of web pages and to create an accessible environment for software agents. Ontologies are one way of representing the knowledge of these agents. For a discussion of the term *ontology* please refer to [Gua98] and a more recent discussion in [Gru09]. The idea to represent datasets on the Internet with ontologies was first widely made public in [BLHL01]. Since then the Semantic Web became a widely used buzzword.

There is increased interest in the development of Semantic Web applications, e.g. digital libraries [KKS09, GFW08], community management [BM07, MP06], and health-care systems [DS05, CdK08]. As the Semantic Web evolves, the amount of data available in these applications is growing with an incredible speed. Since the size of the Semantic Web is expected to further grow in the coming years, scalability and performance of Semantic Web systems become increasingly important. Usually, such systems deal with information described in description-logic based ontology languages such as OWL [HKP$^+$09], and provide services for storing, querying, and updating large numbers of facts.

Decidability results for many expressive description logics and for query answering over these description logics have been shown, e.g., for $\mathcal{SHIQ}$ in [GHLS07], $\mathcal{SHOQ}$ in [GHS08], and $\mathcal{ALCHIOQ}b$ in [GR10]. However, early tableau-based description logic reasoning systems, e.g. Racer [HMW04] and Pellet [SPG$^+$07], do not perform well with large ontologies since the implementation of tableau algorithms is built based on efficient main memory data structures. As long as a tableau representation for an ontology fits into main memory these systems are quite successfully used in practice. However, if the tableau representation does not fit into main memory, these systems are doomed to fail because of out of memory errors or extensive paging activities of the operating system. Until now, to the best of our knowledge, there is no successful implementation of tableau algorithms directly over external memory as, e.g. relational database systems. To sum up, many traditional reasoning systems raise serious scalability concerns, because these systems are not tailored to the peculiarities of secondary storage and do not provide appropriate indexing techniques. There are several solutions proposed in the scientific community. These solutions can be categorized as follows.

There exists a lot of research to identify tractable description logics. For example the descriptions logic $\mathcal{EL}$ and extensions up to $\mathcal{EL}^{++}$, introduced in [BBL08], admit reasoning in polynomial time for classification and instance checking. Another lightweight description logic (family) is *DL-LITE* [CDGL$^+$05]. For an extensive overview see [ACKZ09]. *DL-LITE* allows the use of relational database management systems for query answering.

Another tractable fragment is the rule-based language OWL-R, introduced in [HKP+09]. All tractable fragments have in common that the set of constructors in the ontology language is restricted in order to obtain efficient reasoning algorithms for query answering. However, in practical applications, users often need more expressive languages.

The increasing growth of Semantic Web applications also led to the development of a new class of external memory-based retrieval systems, so called triple stores. Originally motivated to store RDF schema information, see [Bec04], a general architecture to store triples was proposed in [BKvH03]. In the recent years, the amount of these stores substantially increased, see for instance Franz AllegroGraph [Fra11] or OWLIM [Kir06]. An extensive overview over triple stores over large datasets can be found in [RDE+07]. Although the creators of triple stores continuously come up with more impressive performance evaluation results, there are two basic problems with these statistics.

First, in general, it is not clear what kind of reasoning takes place inside the triple store during retrieval - it can be anything from pure lookup to complex description logic reasoning. Second, the hardware test configurations used by triple stores creators seem to be a little over the line. For instance, if one uses four computers with 48 GB of main memory each, then it is not a big surprise that the system is able to handle datasets in the order of several GB. This scenario seems to be at odds with the original intention of triple stores - managing data in external memory.

Tests to build a query answering engine on top of a triple store, for instance in [Spa07], failed. Even though a worst-case efficient algorithm for the description logic $\mathcal{ALC}$ [SSS91] was implemented, the approach turned out to be not useful for query answering.

Another approach to overcome the problem of reasoning over large ontologies is to approximate the ontology by a more compact representation or in a weaker description logic. In [PTZ09], the authors propose to reuse the idea of knowledge compilation to approximate ontologies in a weaker ontology language. For the ontology language of their choice, i.e. *DL-LITE*, efficient query answering algorithms with polynomial data complexity exist. Reasoning on the approximated ontology allows to include/reject potential answers with respect to the original ontology. A similar direction was taken in [RPZ10], where the terminology part of an ontology is approximated to the description logic $\mathcal{EL}^{++}$. The results from the approximated ontology are used for more efficient classification over the original ontology. The classification results can then be used for more efficient retrieval as well.

Another approach focusing on reasoning over instances in large ontologies is presented in [TRKH08]. The algorithms in [TRKH08] are based on KAON2 [Mot08] algorithms, which transform the terminological part of an ontology into Datalog [MW88]. Depending on the transformation strategy, the obtained Datalog program can be used for sound or complete reasoning over instances in the source ontology. The preceding approximation approaches rely on expressivity reduction of the ontology language.

A different approach is proposed in [FKM+06], [DFK+07], and [DFK+09], based on summarization and refinement. First, a summarization of the assertional part is created by aggregating individuals. This is part of a setup step that can be performed offline, i.e.

before query answering takes place. Queries are then executed over the summarization. During the summarization process, one has to take care of inconsistencies. If the summarization leads to inconsistencies, previously merged individuals have to be broken up again.

While approximation techniques usually rely on the summarization of the input or the reduction of the expressivity, there exist modularization techniques which try to extract independent modules with respect to a given reasoning problem. Most of the modularization techniques focus on TBox modularization. In [GPSK06], the notion of a module for the terminological part of an ontology is introduced, and an algorithm for computing modules is presented. This work is further extended in [GHKS09].

Usually, modularization of terminologies has not only the intention to extract modules, but to also combine modules from different source ontologies into one importing ontology. This is in detail discussed in [BS03], where so-called distributed description logics are proposed. The idea is to create rules between parts of terminologies, so-called bridge rules, to propagate information between source ontologies.

The review of state-of-the-art ontology systems shows that many existing reasoning systems are implemented using main memory techniques or at least have to fall back to use main memory techniques for the whole dataset. Therefore, these systems are often unable to handle data which is too big to fit into main memory. The proposed solutions for external memory reasoning systems are usually created for less expressive description logics. Furthermore, to the best of our knowledge, there exist no external memory techniques to explicitly support updates of ontologies. In general, all necessary data structures are recomputed after each ontology update from scratch again.

1.2 Research Objectives and Scientific Contributions

The main goal of the research presented in this thesis is to investigate optimizations and heuristics for query answering with tableau-based reasoning systems. In detail, the thesis has the following objectives:

- Focus on a class of description logics which we call semi-expressive. These semi-expressive description logics are between tractable description logics, such as $\mathcal{EL}^{++}$ or *DL-LITE*, and inherently intractable logics, such as $\mathcal{SHOIQ}$ and $\mathcal{SROIQ}$. Our focus is on the description logic $\mathcal{SHI}$ (no nominals and no choose rule)

- Release the main memory burden from description logic reasoning systems for semi-expressive ontologies. It should be possible to perform instance checks on large ontologies efficiently in the average case.

- Optimize instance retrieval queries beyond naive iteration over all individuals.

- Propose index data structures for easy and direct implementation of instance checks and instance retrieval over semi-expressive ontologies.
- Provide updatable index data structures for reasoning.

When designing and developing a description logic reasoning system, a lot of decisions have to made at design time. During dissemination these decisions have to be reconsidered and sometimes reverted/changed. This leads to the situation that many existing reasoning systems have a lot of tricks and heuristics implemented, which often are not published anywhere. We intend to provide a set of optimization techniques, which are clearly described and can be implemented right away. At some points, we might explain even „simple“ techniques and definitions quite formally, but we think this is necessary in order to put our techniques across and make them reusable in different scenarios.

The major contributions of this thesis are as follows:

- Inspired by graph partitioning approaches, we introduce a set of modularization techniques over the assertional part of $\mathcal{SHI}$-ontologies. It is possible to perform instance checks on usually small „independent“ subsets of assertions. The „independence“ can be exploited to only load a small part of the ontology into main memory at a time.
- In the style of binary instance retrieval techniques [HM04], we achieve optimized instance retrieval by defining a similarity relation over individuals. We define criteria for sufficient degrees of similarity to treat sets of individuals as equivalent for instance retrieval checks. In addition, we define index data structures to manage modules and similarity information. With this idea we can avoid that similar modules are repeatedly loaded into main memory.
- Modularization techniques and similarity criteria give rise to efficient use of recent advances in distributed and parallel computing, such as multicore-systems and cloud computing [Vou08].
- We define a class of syntactic updates over ontologies and describe how the index data structures have to be adapted under each syntactic update operation.
- We evaluate our modularization techniques for benchmark and real world ontology data. It is shown that our techniques help query answering systems to reason over ontologies which do not fit into main memory.

This thesis is not only interesting for developers of description logic reasoners. Understanding possible optimization techniques can also help ontology designers and users to improve (the experience with) their ontologies.

1.3 Dissemination Activities

Several parts of this thesis have been published to disseminate research results at different stages. In the following, dissemination activities are listed in inverse chronological order.

- Year 2011:
  - Sebastian Wandelt, Ralf Möller:
    Islands and Query Answering for ALCHI-Ontologies in *CCIS 128: Third International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management. 2010*, Heidelberg, Germany, 2011, Springer, pages 224 - 236.
- Year 2010:
  - Sebastian Wandelt, Ralf Möller, Michael Wessel:
    Towards Scalable Instance Retrieval over Ontologies in *Journal of Software and Informatics*, 2010,4(3):201 218.
  - Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Sebastian Wandelt:
    Dealing Efficiently with Ontology-Enhanced Linked Data for Multimedia in Proceedings of *International Conference on Semantic and Multimedia Technologies*, SAMT 2010, Saarbrücken, 2010, Springer.
  - Sebastian Wandelt, Ralf Möller:
    Distributed Island-Based Query Answering for Expressive Ontologies in Proceedings of *International Conference on Advances in Grid and Pervasive Computing*, GPC 2010, Hualien, Taiwan, pages 461 - 470.
  - Alissa Kaplunova, Ralf Möller, Sebastian Wandelt, Michael Wessel:
    Towards Scalable Instance Retrieval over Ontologies in Proceedings of *Knowledge Science, Engineering and Management, 4th International Conference*, KSEM 2010, Belfast, Northern Ireland, 2010, pages 436 - 448.
  - Sebastian Wandelt, Ralf Möller:
    Distributed Island-Based Query Answering for Expressive Ontologies in Proceedings of *International Workshop on Description Logics*, DL 2010, Waterloo, Canada.
  - Sebastian Wandelt, Ralf Möller:
    Sound Summarizations for ALCHI-Ontologies - How to Speed up Instance Checking and Instance Retrieval in Proceedings of *International Conference on Agents and Artificial Intelligence*, ICAART 2010, Valencia, 2010, pages 656 - 661.

- Year 2009:
  - Sebastian Wandelt, Ralf Möller:
    Updatable Island Reasoning for ALCHI-Ontologies in Proceedings of *International Conference on Knowledge Engineering and Ontology Development*, KEOD 2009, Funchal, Portugal, 2010, pages 48 - 55.
- Year 2008:
  - Sebastian Wandelt:
    Partitioning OWL Knowledge Bases - Revisited and Revised in Proceedings of *International Workshop on Description Logics*, DL 2008, Dresden, Germany, 2008.
  - Sebastian Wandelt, Ralf Möller:
    Island Reasoning for ALCHI-Ontologies in Proceedings of *Formal Ontology in Information Systems*, FOIS 2008, Saarbrücken, Germany, 2008, pages 164 - 177.
- Year 2007:
  - Sebastian Wandelt, Ralf Möller:
    Scalability of OWL Reasoning: Role condensates in Proceedings of *On the Move to Meaningful Internet Systems: OTM 2007 Workshops*, OTM 2007, Vilamoura, Portugal, 2007, pages 1145 - 1154.
  - D. Calvanese, G. De Giacomo, B. C. Grau, A. Kaplunova, D. Lembo, M. Lenzerini, R. Möller, R. Rosati, U. Sattler, B. Sertkaya, B. Suntisrivaraporn, S. Tessaris, A.-Y. Turhan, and S. Wandelt:
    D14: Ontology-Based Services: Usage Scenarios and Test Ontologies. Project deliverable, *TONES*, 2007. http://www.tonesproject.org.
  - G. De Giacomo, E. Franconi, B. Cuenca Grau, V. Haarslev, A. Kaplunova, A. Kaya, D. Lembo, C. Lutz, M. Milicic, R. Möller, U. Sattler, B. Sertkaya, B. Suntisrivaraporn, A.-Y. Turhan, S. Wandelt, and M. Wessel:
    D23: Analysis of Test-Results on Individual Test Ontologies. Project deliverable, *TONES*, 2007. http://www.tonesproject.org.

1.4 Outline

Before optimization techniques and heuristics for the solution of reasoning problems are introduced, the formal foundations are defined in Chapter 2. Besides defining basic mathematical notions, e.g. set theory, handling of lists and graphs, the focus of Chapter 2 is on the introduction of description logics. The conceptual language is introduced, decision problems for ontologies are defined, and reasoning procedures explained. Moreover, different members of the description logic family are introduced and their formal properties are recapitulated.

Chapter 3 explains the fundamental idea of breaking down a large ontology into smaller parts, called ABox modularization. The idea is to rewrite the assertional part of an ontology into smaller chunks (modules), such that decision problems can be solved by considering these small chunks only. The modularization technique is extended to further break up existing assertional information by so called intensional-based partitioning. The modularization techniques are first shown and proved for the description logic $\mathcal{ALC}$ and then further lifted to the description logic $\mathcal{SHI}$.

While Chapter 3 introduces a purely technical transformation on the assertional part of an ontology, Chapter 4 shows how to use these techniques for more efficient reasoning. First, so-called individual islands are proposed. An individual island for an individual contains a usually small set of the assertional axioms relevant for instance checking. Furthermore, instance retrieval techniques over these individual islands are proposed and discussed. The main concept for optimization of instance retrieval is to use similarity measures over individual islands in order to reduce the number of atomic instance checks. Moreover, a data structure called one-step node is introduced, which can be used for similarity detection, as well as for direct optimization of instance retrieval over ontologies.

In Chapter 5, it is shown how the techniques from Chapter 4 can be applied to manage updates to ontologies. A set of syntactic update functions over ontologies is defined and for each of these update functions we show, how to change the data structures introduced in Chapter 4.

We present a prototypical implementation of our algorithms in Chapter 6. In addition, we evaluate the prototype over test ontologies to show up to what extent updatable modularization techniques are applicable and scale in practice.

Chapter 7 concludes this work by summarizing the main achievements. Furthermore, we indicate interesting directions for future work.

Chapter 2: Preliminaries

In this chapter, we introduce mathematical notions. First, our notation for functions, sets, and graphs is introduced in Section 2.1. In Section 2.2, we introduce the family of description logics, a logical formalism for knowledge representation. We introduce the conceptual language and axioms for descriptions of ontologies. Furthermore, we formally define decision problems over description logic ontologies and recapitulate existing decision algorithms. We define an example ontology in Section 2.3 for further use.

2.1 Basic Preliminaries

First, we define general notions from basic mathematics. The set of natural numbers is denoted with $\mathbb{N}$. The powerset of a set $\mathbf{S}$ is denoted with $\wp(\mathbf{S})$. The number of elements in a set $\mathbf{S}$ is denoted with $|\mathbf{S}|$. If we define elements in a set, the expression $\{x \mid x \in \mathbf{X} \wedge ...\}$ is often abbreviated as $\{x \in \mathbf{X} \mid ...\}$, e.g. the set definition $\{x \mid x \in \mathbb{N} \wedge primenumber(x)\}$ is rewritten as $\{x \in \mathbb{N} \mid primenumber(x)\}$. With $\mathbf{S}_1 \ominus \mathbf{S}_2$ we denote the symmetric difference between two sets, i.e. $\mathbf{S}_1 \ominus \mathbf{S}_2 = (\mathbf{S}_1 \setminus \mathbf{S}_2) \cup (\mathbf{S}_2 \setminus \mathbf{S}_1)$.

Definition 2.1 (N-ary Relations)**:**
Given a collection of sets $\mathbf{X}_1, ..., \mathbf{X}_n$, the *n-ary relation* $\mathbf{R}$ *over* $\mathbf{X}_1, ..., \mathbf{X}_n$ is a subset of $\mathbf{X}_1 \times ... \times \mathbf{X}_n$. To denote the type of an n-ary relation, we write $\mathbf{R} : \mathbf{X}_1 \times ... \times \mathbf{X}_n$. $(x_1, .., x_n) \in \mathbf{R}$ is also denoted with $\langle x_1, ..., x_n \rangle \in \mathbf{R}$ or $\mathbf{R}(x_1, ..., x_n)$. An element $\langle x_1, ..., x_n \rangle$ of an n-ary relation $\mathbf{R}$, is called a *tuple*. $\langle x_1, x_2 \rangle$ is called a *pair* and $\langle x_1, x_2, x_3 \rangle$ is called a *triple*. Given a set $\mathbf{X}$ and a set $\mathbf{Y}$, a *binary relation* $\mathbf{R}$ is a 2-ary relation over $\mathbf{X}$ and $\mathbf{Y}$. If $(x, y) \in \mathbf{R}$, this is also denoted with $x\mathbf{R}y$. A binary relation $\mathbf{R}$ is

- *left-total* if $\forall x \in \mathbf{X}.\exists y \in \mathbf{Y}.\mathbf{R}(x, y)$,
- *surjective* if $\forall y \in \mathbf{Y}.\exists x \in \mathbf{X}.\mathbf{R}(x, y)$,
- *functional* if $\forall x \in \mathbf{X}.\forall y_1 \in \mathbf{Y}.\forall y_2 \in \mathbf{Y}.\mathbf{R}(x, y_1) \wedge \mathbf{R}(x, y_2) \implies y_1 = y_2$,
- *injective* if $\forall x_1 \in \mathbf{X}.\forall x_2 \in \mathbf{X}.\forall y \in \mathbf{Y}.\mathbf{R}(x_1, y) \wedge \mathbf{R}(x_2, y) \implies x_1 = x_2$, and
- *bijective* if $\mathbf{R}$ is surjective and injective.

Given a binary relation $\mathbf{R} : \mathbf{X} \times \mathbf{X}$, we let

- *reflexive closure* of $\mathbf{R}$: $\mathbf{R}^{REF} = \mathbf{R} \cup \{(x, x) \mid x \in \mathbf{X}\}$,
- *symmetric closure* of $\mathbf{R}$: $\mathbf{R}^{SYM} = \mathbf{R} \cup \{(x_2, x_1) \mid (x_1, x_2) \in \mathbf{R}\}$, and

- *transitive closure* of $\mathbf{R}$, denoted $\mathbf{R}^{TRA}$, is the smallest relation satisfying the following constraints:

$$\begin{aligned} \mathbf{R}(x_1, x_2) &\implies \mathbf{R}^{TRA}(x_1, x_2) \\ \mathbf{R}(x_1, x_2)^{TRA} \wedge \mathbf{R}(x_2, x_3) &\implies \mathbf{R}^{TRA}(x_1, x_2). \end{aligned}$$

Sometimes we combine several closure operations, e.g. $\mathbf{R}^{SYM,TRA}$ denotes the transitive closure of the symmetric closure of $\mathbf{R}$.

Definition 2.2 (Functions and their Properties)**:**
Given a set $\mathbf{X}$ and a set $\mathbf{Y}$, a *function* f is a functional binary relation over $\mathbf{X}$ and $\mathbf{Y}$. The set $\mathbf{X}$ is called the *domain* of f and the set $\mathbf{Y}$ is called the *codomain* of f. $(x, y) \in f$ is also denoted with $f(x) = y$ or $x \rightarrow_f y$. In order to denote the type of a function we use the notation $f : \mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{X}$ is the domain of f and $\mathbf{Y}$ is the codomain of f. If $x \in \mathbf{X}$ and the value $f(x)$ is not defined for a (non-total) function $f : \mathbf{X} \rightarrow \mathbf{Y}$, we denote this with $f(x) = n.d.$. The set of values used from the domain of f, i.e. the set $\{x \mid f(x) \neq n.d.\}$, is denoted as $FD(f)$. The set of values used from the codomain of f, i.e. the set $\{y \mid \exists x \in \mathbf{X} \wedge f(x) = y\}$, is denoted as $FVAL(f)$.

Although the properties of functions are directly derived from relations, we define them explicitly here. A function is

- *total* if $\forall x \in \mathbf{X}.\exists y \in \mathbf{Y}.f(x) = y$,
- *partial* if f is not total,
- *surjective* if $\forall y \in \mathbf{Y}.\exists x \in \mathbf{X}.f(x) = y$,
- *injective* if $\forall x \in \mathbf{X}.\forall y \in \mathbf{Y}.(f(x) = f(y) \implies x = y)$, and
- *bijective* if f is surjective and injective.

Given a set $\mathbf{S} \subseteq \mathbf{X}$, the result of applying $f : \mathbf{X} \rightarrow \mathbf{Y}$ to all elements in $\mathbf{S}$ is defined as the set $f(\mathbf{S}) = \{y \in \mathbf{Y} \mid \exists x \in \mathbf{S}.(f(x) = y)\}$.

Since functions are special binary relations, we often use notions from binary relations on the relational representation of functions. For instance, the expression $f = \emptyset$ denotes that the function f has no mappings, i.e. $f(x) = n.d.$ for all elements x in the domain of f. Furthermore, we apply set manipulation operations, such as union and difference, on two functions directly, as long as the domain allows it. In order to define functions, we use the following notation sometimes: $f = \{a \rightarrow 1, b \rightarrow 2\}$, which means that $f(a) = 1$, $f(b) = 2$, and $f(x) = n.d.$ for all other elements in the domain of f.

Given a function $f : \mathbf{X} \rightarrow \mathbf{Y}$, the *inverse function* $f^- : \mathbf{Y} \rightarrow \mathbf{X}$, is defined as

$$f^-(y) = x \iff f(x) = y.$$

Please note that the inverse of a function is only well defined, if the original function f is injective.

Given a function $f : \mathbf{X} \to \mathbf{Y}$, the *restriction of the domain to the set* $\mathbf{S}$, denoted $f^{\restriction \mathbf{S}}$, is defined as

$$f^{\restriction \mathbf{S}}(x) = \begin{cases} f(x) & \text{if } x \in \mathbf{S}, \\ n.d. & \text{otherwise.} \end{cases}$$

In the following, we introduce the notion of multisets [Knu81, Bli89]. As an extension of sets, elements can occur multiple times in a multiset. A function is used to keep track of the number of occurrences of each element.

Definition 2.3 (Multiset)**:**
Given a *base set* **domain**, a *multiset* **MS** *over* **domain** is a total function **MS** : **domain** $\to \mathbb{N}$. Let $\mathbf{MS}_1$: **domain** $\to \mathbb{N}$ and $\mathbf{MS}_2$: **domain** $\to \mathbb{N}$ be multisets over a set **domain**, and $s \in$ **domain** then

- $(\mathbf{MS}_1 \uplus \mathbf{MS}_2)(s) = \mathbf{MS}_1(s) + \mathbf{MS}_2(s)$ (*multiset union*) and
- $(\mathbf{MS}_1 \setminus \mathbf{MS}_2)(s) = max(0, \mathbf{MS}_1(s) - \mathbf{MS}_2(s))$ (*multiset difference*).

The *empty multiset*, denoted Ξ, is a multiset, such that $\forall s \in$ **domain**.$\Xi(s) = 0$. A multiset **MS** *contains an element* s, denoted $s \in \mathbf{MS}$, if $\mathbf{MS}(s) \geq 1$.

Given a base set **domain** and a set $\mathbf{S} \subseteq$ **domain**, the *multiset of* $\mathbf{S}$, denoted $\uparrow_S^M (\mathbf{S})$, is a multiset defined as follows:

$$\uparrow_S^M (\mathbf{S})(x) = \begin{cases} 1 & \text{if } x \in \mathbf{S}, \\ 0 & \text{otherwise.} \end{cases}$$

Given a base set **domain** and a multiset **MS** over **domain**, the *set of* **MS**, denoted $\downarrow_S^M (\mathbf{MS})$, is a set defined as $\downarrow_S^M (\mathbf{MS}) = \{x \mid x \in \mathbf{MS}\}$.

Example 2.1 (Multiset Operations)**:**
Given the two sets $\mathbf{S}_1 = \{a, b, c\}$ and $\mathbf{S}_2 = \{b, c, d\}$, we have

- $\uparrow_S^M (\mathbf{S}_1) = \{a \to 1, b \to 1, c \to 1\}$,
- $\uparrow_S^M (\mathbf{S}_2) = \{b \to 1, c \to 1, d \to 1\}$,
- $(\uparrow_S^M (\mathbf{S}_1)) \uplus (\uparrow_S^M (\mathbf{S}_2)) = \{a \to 1, b \to 2, c \to 2, d \to 1\}$ and
- $\downarrow_S^M ((\uparrow_S^M (\mathbf{S}_1)) \uplus (\uparrow_S^M (\mathbf{S}_2))) = \{a, b, c, d\}$.

We introduce the formal notation of a list in Definition 2.4.

Definition 2.4 (List)**:**
Given a *base set* **domain**, a *list* L *over* **domain** is inductively defined as follows:

- $\square$ is a list (*empty list*).

- $d \circ L$ is a list if L is a list and $d \in \mathbf{domain}$ (*concatenated list*).

The length of a list L, denoted $|L|$, is defined recursively as usual, i.e. $|\square| = 0$ and $|d \circ L| = |L| + 1$. To simplify the handling of lists, we use common notation from the literature, e.g. the expression $[a, b, c]$ is denoted to represent the list $(c \circ (b \circ (a \circ \square)))$. With *begin* (or *start*) of L we refer to the first element in L, e.g. element a above, and with *end* of L we refer to the last element of L, e.g. element c above.

Definition 2.5 (Directed Graph)**:**
A *directed graph* is a tuple $\mathbb{G} = \langle \mathbf{N}, \mathbf{E} \rangle$, such that $\mathbf{N}$ is a set of *nodes* and $\mathbf{E} \subseteq \mathbf{N} \times \mathbf{N}$ is a set of *edges*. Given a node $n \in \mathbf{N}$, an edge $(n, n_2) \in \mathbf{E}$ is called *outgoing edge* of n, and an edge $(n_2, n) \in \mathbf{E}$ is called *incoming edge* of n. The set of outgoing edges of a node n is denoted with $out_{\mathbb{G}}(n)$. The set of incoming edges of a node n is denoted with $in_{\mathbb{G}}(n)$. The *node successors* of n, denoted $succs_{\mathbb{G}}(n)$, are defined as the set of nodes connected by an outgoing edge from n, i.e., $succs_{\mathbb{G}}(n) = \{n_2 \in \mathbf{N} \mid (n, n_2) \in \mathbf{E}\}$. The *node predecessors* of n, denoted $preds_{\mathbb{G}}(n)$, are defined as the set of nodes connected by an outgoing edge to n, i.e. $preds_{\mathbb{G}}(n) = \{n_2 \in \mathbf{N} \mid (n_2, n) \in \mathbf{E}\}$. The *node neighbors* of n, denoted $neighbors_{\mathbb{G}}(n)$, are defined as the set of nodes connected by an outgoing or incoming edge from n, i.e. $neighbors_{\mathbb{G}}(n) = \{n_2 \in \mathbf{N} \mid (n, n_2) \in \mathbf{E} \vee (n_2, n) \in \mathbf{E}\}$.

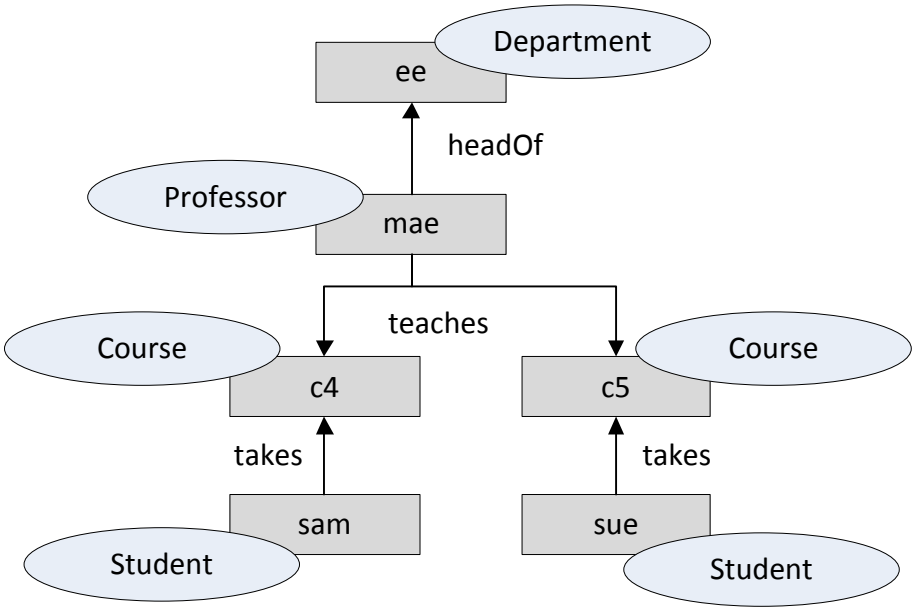
Definition 2.6 (Trees)**:**
A *directed tree* (or short *tree*) is a tuple $\mathbb{T} = \langle \mathbf{N}, root, children \rangle$, where $\mathbf{N}$ is a set of *nodes*, $root \in \mathbf{N}$ is a distinguished *root node*, and $children : \mathbf{N} \to \wp(\mathbf{N})$ is a total function which assigns a set of *child nodes* to each node, such that every node other than the root node is reachable from the root (via *children*) and has exactly one predecessor. Given a directed tree $\mathbb{T} = \langle \mathbf{N}, root, children \rangle$, a node $n \in \mathbf{N}$ is called *leaf node* if $children(n) = \emptyset$, otherwise the node n is called *inner node*. A tree $\mathbb{T} = \langle \mathbf{N}, root, children \rangle$ is called *x-ary* if each node has at most x children, i.e. $\forall n \in \mathbf{N}.|children(n)| \leq x$. We refer to 2-ary trees as *binary trees*.

In the following, we extend our tree and graph definitions by introducing labels. We formally define two different functions which can be used to label nodes (of trees and graphs) and edges (of graphs).

Definition 2.7 (Node and Edge Labeling Function)**:**
Given a set of nodes $\mathbf{N}$, a set of *node labels* $\mathbf{SGNL}$, and a set of *edge labels* $\mathbf{SGEL}$, a *node labeling function* $\phi : \mathbf{N} \to \mathbf{SGNL}$ *for* $\mathbf{N}$ assigns to each node $n \in \mathbf{N}$ a label from $\mathbf{SGNL}$ and an *edge labeling function* $\sigma : \mathbf{N} \times \mathbf{N} \to \mathbf{SGEL}$ *for* $\mathbf{N}$ assigns to each pair of nodes a label from $\mathbf{SGEL}$. A *directed labeled graph* is a tuple $\mathbb{G} = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$, such that $\langle \mathbf{N}, \mathbf{E} \rangle$ is a directed graph, ϕ is a node labeling function for $\mathbf{N}$, and σ is an edge labeling function for $\mathbf{N}$. A *directed labeled tree* is a tuple $\mathbb{T} = \langle \mathbf{N}, root, children, \phi, \sigma \rangle$, such that $\langle \mathbf{N}, root, children \rangle$ is a directed tree, ϕ is a node labeling function for $\mathbf{N}$ and σ is an edge labeling function for $\mathbf{N}$.

Sometimes we also add more than one node labeling functions to on tree. In this case we will mention this fact explicitly. In the following example, we define a graph $\mathbb{G}_{Ex2.2}$ for modeling an excerpt of a university domain.

Figure 2.1 Graph example $\mathbb{G}_{Ex2.2}$



Example 2.2 (Graphs)**:**
Given a set of node labels $\mathbf{SGNL} = \wp(\{Department, Professor, Course, Student\})$ and edge labels $\mathbf{SGEL} = \wp(\{headOf, teaches, takes\})$, an example directed labeled graph $\mathbb{G}_{Ex2.2} = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$ is given as follows:

$$\mathbf{N} = \{ee, mae, c4, c5, sam, sue\}$$
$$\mathbf{E} = \{(mae, ee), (mae, c4), (mae, c5), (sam, c4), (sue, c5)\}$$
$$\phi(n) = \begin{cases} \{Department\} & \text{if } n = ee, \\ \{Professor\} & \text{if } n = mae, \\ \{Course\} & \text{if } (n = c4 \vee n = c5), \\ \{Student\} & \text{if } (n = sam \vee n = sue), \\ \emptyset & \text{otherwise.} \end{cases}$$

$$\sigma(\mathbf{e}) = \begin{cases} \{headOf\} & \text{if } \mathbf{e} = (mae, ee), \\ \{teaches\} & \text{if } (\mathbf{e} = (mae, c4) \vee \mathbf{e} = (mae, c5)), \\ \{takes\} & \text{if } (\mathbf{e} = (sam, c4) \vee \mathbf{e} = (sue, c5)), \\ \emptyset & \text{otherwise.} \end{cases}$$

The graph is depicted in Figure 2.1.

Examples for further graph notations over $\mathbb{G}_{Ex2.2}$ are given as follows:

- Outgoing edges of mae: $out_{\mathbb{G}_{Ex2.2}}(mae) = \{(mae, ee), (mae, c4), (mae, c5)\}$,

- Incoming edges of $c4$: $in_{\mathbb{G}_{Ex2.2}}(c4)\{(mae, c4), (sam, c4)\}$,
- Node neighbors of $c5$: $neighbors_{\mathbb{G}_{Ex2.2}}(c5) = \{mae, sue\}$.

We finalize our introduction of basic mathematical notions here and proceed to description logics.

2.2 Description Logics

Description logics are a family of languages for knowledge representation. Historically, description logics are descendants of semantic nets [Qui68] and frame systems [Min74]. In Artificial Intelligence, description logics are used for formal reasoning about application domains. The most prominent application of description logics might be the use as a formalism for the Semantic Web [BHS05]. For further information on the historical background of description logics, we refer to [BCM+07]. A general review on logic-based knowledge representation with description logics and other logics as well, such as modal logics, is given in [Baa99].

2.2.1 Conceptual Language

In the following, we introduce the conceptual language underlying description logics by defining syntax and semantics for different constructors and descriptions.

Definition 2.8 (Base sets)**:**
We assume a number of disjoint *base sets* as follows:

- **CN** is a non-empty set of *concept names*,
- **RN** is a non-empty set of *role names*,
- **NIN** is a non-empty set of *named individuals*, and
- **AIN** is a non-empty set of *anonymous individuals*.

The set of *individuals* is $\mathbf{IN} = \mathbf{NIN} \cup \mathbf{AIN}$.

The elements from base sets form the basis for descriptions in description logics. The semantics of base descriptions is defined by an interpretation.

Definition 2.9 (Interpretations)**:**
An *interpretation* $\mathcal{I}$ is a pair $\langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, such that

- $\Delta_{\mathcal{I}}$ is a non-empty set, the *domain of* $\mathcal{I}$, and

- $\cdot^{\mathcal{I}}$ is an *interpretation function* which assigns to every $A \in \mathbf{CN}$ a set $A^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}}$, to every $S \in \mathbf{RN}$ a set $S^{\mathcal{I}} \subseteq \Delta_{\mathcal{I}} \times \Delta_{\mathcal{I}}$, and to every individual $a \in \mathbf{IN}$ an element $a^{\mathcal{I}} \in \Delta_{\mathcal{I}}$.

The definition of interpretation functions is extended step-by-step for each of the description constructors introduced below. First, we introduce role descriptions, which are used to relate individuals in an application domain.

Definition 2.10 (Role Descriptions)**:**
The expression R is a *role description* if and only if

- $R = S$ and $S \in \mathbf{RN}$ (R is called *role name*) or
- $R = R_2^-$ and R_2 is a role description (R is called *inverse role* of R_2). If R_2 is a role name, then R is called *inverse role name*.

Interpretation functions are extended such that we have $(R^-)^{\mathcal{I}} = \{(a_1, a_2) \mid (a_2, a_1) \in R^{\mathcal{I}}\}$. The set of all role descriptions is denoted with **Rol**. A role description R is called *atomic role* if R is a role name or R is a inverse role name.

In Example 2.3, we define some role descriptions which can be constructed from the constructors introduced above. For instance, the role description $teaches^-$, is the inverse role of $teaches$.

Example 2.3 (Role Descriptions)**:**
Given a set $\mathbf{RN} = \{headOf, takes, teaches\}$, examples of role descriptions are:

- Inverse role name: $teaches^-$.
- Atomic roles: $teaches$, $takes$, $headOf^-$.
- Non-atomic role: $teaches^{--}$.

It is easy to see that each non-atomic role is equivalent (with respect to interpretations) to an atomic role. Hence, in the following, we assume that each role description is atomic. Next, we introduce concept descriptions, which are used to categorize and classify individuals in an application domain.

Definition 2.11 (Concept Descriptions)**:**
The expression C is a *concept description* if and only if

- $C = \top$ (*top-symbol*),
- $C = \bot$ (*bottom-symbol*),
- $C = A$, such that $A \in \mathbf{CN}$ (C is called *concept name*),
- $C = C_1 \sqcap C_2$, such that C_1 and C_2 are concept descriptions (C is called *concept intersection*),

- $C = C_1 \sqcup C_2$, such that C_1 and C_2 are concept descriptions (C is called *concept union*),
- $C = \neg C_2$, such that C_2 is a concept description (C is called *concept negation*),
- $C = \exists R.C_2$, such that C_2 is a concept description and R is a role description (C is called *exists constraint* or *existential restriction*),
- $C = \forall R.C_2$, such that C_2 is a concept description and R is a role description (C is called *forall constraint* or *value restriction*),
- $C = \geq_n R.C_2$, such that C_2 is a concept description, R is a role description and $n \in \mathbb{N}$ (C is called *minimum cardinality restriction* or *cardinality restriction*),
- $C = \leq_n R.C_2$, such that C_2 is a concept description, R is a role description and $n \in \mathbb{N}$ (C is called *maximum cardinality restriction* or *cardinality restriction*), or
- $C = \{a\}$, such that $a \in \mathbf{NIN}$ (C is called *nominal*).

The set of all concept descriptions is denoted with **Con**. Interpretation functions are extended such that we have

- $(\top)^{\mathcal{I}} = \Delta_{\mathcal{I}}$,
- $(\bot)^{\mathcal{I}} = \emptyset$,
- $(C_1 \sqcap C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cap {C_2}^{\mathcal{I}}$,
- $(C_1 \sqcup C_2)^{\mathcal{I}} = {C_1}^{\mathcal{I}} \cup {C_2}^{\mathcal{I}}$,
- $(\neg C)^{\mathcal{I}} = \Delta_{\mathcal{I}} \setminus C^{\mathcal{I}}$,
- $(\exists R.C)^{\mathcal{I}} = \{\delta_1 \in \Delta_{\mathcal{I}} \mid \exists \delta_2 \in \Delta_{\mathcal{I}}.((\delta_1, \delta_2) \in R^{\mathcal{I}} \wedge \delta_2 \in C^{\mathcal{I}}\})$,
- $(\forall R.C)^{\mathcal{I}} = \{\delta_1 \in \Delta_{\mathcal{I}} \mid \forall \delta_2 \in \Delta_{\mathcal{I}}.((\delta_1, \delta_2) \in R^{\mathcal{I}} \implies \delta_2 \in C^{\mathcal{I}}\})$,
- $(\geq_n R.C)^{\mathcal{I}} = \{\delta_1 \in \Delta_{\mathcal{I}} \mid (|\{\delta_2 \mid (\delta_1, \delta_2) \in R^{\mathcal{I}} \wedge \delta_2 \in C^{\mathcal{I}}\}|) \geq n\}$,
- $(\leq_n R.C)^{\mathcal{I}} = \{\delta_1 \in \Delta_{\mathcal{I}} \mid (|\{\delta_2 \mid (\delta_1, \delta_2) \in R^{\mathcal{I}} \wedge \delta_2 \in C^{\mathcal{I}}\}|) \leq n\}$, and
- $(\{a\})^{\mathcal{I}} = a^{\mathcal{I}}$.

A concept description C is called *atomic* if C is a concept name or C is a negated concept name.

Example 2.4 (Example for Concept Descriptions)**:**
Given a set $\mathbf{CN} = \{Course, Professor, Student\}$ and a set $\mathbf{RN} = \{headOf, takes, teaches\}$, examples for concept descriptions are:

- Negated concept names: $\neg Professor, \neg Student$.

- Atomic concepts: $Professor, \neg Student.$
- Non-atomic concepts: $Professor \sqcup Student, \forall headOf^{-}.Professor, \exists takes.Course.$

In Example 2.5 we show one example for an interpretation.

Example 2.5 (Interpretations)**:**
Given a set $\mathbf{CN} = \{Course, Professor, Student\}$ and a set $\mathbf{RN} = \{takes, teaches\}$, an example interpretation $\mathcal{I}_{Ex2.5}$ is defined as follows:

$$\begin{aligned}
\Delta_{\mathcal{I}_{Ex2.5}} &= \{\delta_a, \delta_b, \delta_c, \delta_d\} \\
Course^{\mathcal{I}_{Ex2.5}} &= \{\delta_b\} \\
Professor^{\mathcal{I}_{Ex2.5}} &= \{\delta_a\} \\
Student^{\mathcal{I}_{Ex2.5}} &= \{\delta_c, \delta_d\} \\
teaches^{\mathcal{I}_{Ex2.5}} &= \{(\delta_a, \delta_b)\} \\
takes^{\mathcal{I}_{Ex2.5}} &= \{(\delta_c, \delta_b), (\delta_d, \delta_b)\}.
\end{aligned}$$

The extension of $\mathcal{I}_{Ex2.5}$ yields for example $(\forall takes.Course)^{\mathcal{I}_{Ex2.5}} = \{\delta_c, \delta_d\}$.

Interpretations are often depicted as directed, labeled graphs, such that the domain of the interpretation is mapped to nodes and the interpretation function is mapped to either node or edge labels.

Definition 2.12 (Closure of Concepts)**:**
Given a concept description C, the *concept closure of* C, denoted $clos(C)$, is defined as follows:

$$clos(C) = \begin{cases}
\{\top\} & \text{if } C = \top, \\
\{\bot\} & \text{if } C = \bot, \\
\{A\} & \text{if } C = A, \\
\{\{a\}\} & \text{if } C = \{a\}, \\
\{C\} \cup clos(C_1) \cup clos(C_2) & \text{if } C = C_1 \sqcup C_2, \\
\{C\} \cup clos(C_1) \cup clos(C_2) & \text{if } C = C_1 \sqcap C_2, \\
\{C\} \cup clos(C_1) & \text{if } C = \neg C_1, \\
\{C\} \cup clos(C_1) & \text{if } C = \forall R.C_1, \\
\{C\} \cup clos(C_1) & \text{if } C = \exists R.C_1, \\
\{C\} \cup clos(C_1) & \text{if } C = \geq_n R.C_1, \\
\{C\} \cup clos(C_1) & \text{if } C = \leq_n R.C_1.
\end{cases}$$

The closure of a concept description is usually used for syntactical analysis. We introduce the notion of a concept description in negation normal form in order to further ease syntactical analysis.

Definition 2.13 (Negation Normal Form)**:**
A concept description C is in *negation normal form* if all negations occur in front of

concept names only, i.e. for all $\neg C_1 \in clos(C)$, C_1 is a concept name. The *negation normal form of a* concept description C is denoted $nnf(C)$.

Every concept description can be transformed into a concept description in negation normal form, see [HST00b] and [HS07] for details about the transformation.

In the remaining part, we always assume that each concept description is in negation normal form, unless stated otherwise.

Apart from the introduced syntactical constructors, there exist many description logic extensions. For the sake of completeness we mention some of these extensions here. The most prominent extension might be concrete domains - a means to represent concrete qualities of instances such as age, duration, ids or even spatial information, see [Lut03] for an overview. Other research is focused on probabilistic [Luk08], fuzzy [Str05], and temporal [AFWZ02] extensions. In addition functional and uniqueness constraints [BW97] as well as epistemic operators [DLN+98] have been investigated.

2.2.2 Ontologies

A description logic ontology is a formal representation of knowledge as a set of axioms. One axiom set describes the intensional knowledge and the axiom set describes the extensional knowledge. Both kinds of axioms and their semantics are introduced next.

Definition 2.14 (Role Axioms and their Models)**:**
A *general role inclusion axiom* has the form $R_1 \sqsubseteq R_2$, where R_1 and R_2 are role descriptions. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a general role inclusion axiom $R_1 \sqsubseteq R_2$, denoted $\mathcal{I} \vDash R_1 \sqsubseteq R_2$, if and only if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$. A *general role equality axiom* has the form $R_1 \equiv R_2$, where R_1 and R_2 are role descriptions. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a general role equivalence axiom $R_1 \equiv R_2$, denoted $\mathcal{I} \vDash R_1 \equiv R_2$, if and only if $R_1^{\mathcal{I}} = R_2^{\mathcal{I}}$. A *role transitivity axiom* has the form $Trans(R)$, where R is a role description. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a role transitivity axiom $Trans(R)$, denoted $\mathcal{I} \vDash Trans(R)$, if and only if $R^{\mathcal{I}} = (R^{\mathcal{I}})^{TRA}$.

It is easy to see that an interpretation $\mathcal{I}$ is a model of a general role equality axiom $R_1 \equiv R_2$ if and only if $\mathcal{I}$ is a model for $R_1 \sqsubseteq R_2$ and for $R_2 \sqsubseteq R_1$. Therefore, it is common to take into account general role inclusion axioms only, since each general role equality axiom can be trivially dealt with as two general role inclusion axioms.

Definition 2.15 (Concept Axioms and their Models)**:**
A *general concept inclusion axiom* (GCI) has the form $C_1 \sqsubseteq C_2$, where C_1 and C_2 are concept descriptions. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a general concept inclusion axiom $C_1 \sqsubseteq C_2$, denoted $\mathcal{I} \vDash C_1 \sqsubseteq C_2$, if and only if $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$. General concept inclusion axioms are often just called *concept inclusions*. The set of all general concept inclusion axioms is denoted with **GCIs**. A *general concept equality axiom* has the form $C_1 \equiv C_2$, where C_1 and C_2 are concept descriptions. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a general concept equivalence axiom $C_1 \equiv C_2$, denoted $\mathcal{I} \vDash C_1 \equiv C_2$, if and

only if $C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$. A *general concept disjointness axiom* has the form $C_1 \oslash C_2$, where C_1 and C_2 are concept descriptions. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a general concept disjointness axiom $C_1 \oslash C_2$, denoted $\mathcal{I} \vDash C_1 \oslash C_2$, if and only if $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$.

An interpretation $\mathcal{I}$ is a model of a general concept equality axiom $C_1 \equiv C_2$ if and only if $\mathcal{I}$ is a model for $C_1 \sqsubseteq C_2$ and for $C_2 \sqsubseteq C_1$. Therefore it is a common assumption in description logic research to take into account general concept inclusion axioms only, since each general concept equality axiom can be trivially dealt with as two general concept inclusion axioms.

An interpretation $\mathcal{I}$ is a model of a general concept disjointness axiom $C_1 \oslash C_2$ if and only if $\mathcal{I}$ is a model for $C_1 \sqcap C_2 \sqsubseteq \bot$. Therefore it is again common to take into account general concept inclusion axioms only, since each general concept disjointness axiom can be trivially dealt with using one general concept inclusion axiom.

Definition 2.16 (Individual Axioms and their Models)**:**
A *concept assertion axiom* has the form $C(a)$, where $a \in \mathbf{IN}$ and C is a concept description. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a concept assertion axiom $C(a)$, denoted $\mathcal{I} \vDash C(a)$, if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$. Concept assertion axioms are often just called *concept assertions*. A *role assertion axiom* has the form $R(a_1, a_2)$, where $\{a_1, a_2\} \subseteq \mathbf{IN}$ and R is a role description. An interpretation $\mathcal{I}$ *satisfies* (*is a model of*) a role assertion axiom $R(a_1, a_2)$, denoted $\mathcal{I} \vDash R(a_1, a_2)$, if and only if $(a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in R^{\mathcal{I}}$. Role assertion axioms are often just called *role assertions*.

Please note that, in general, we only allow atomic concepts in concept assertion axioms. This way, it is easier to define efficient structures for reasoning. The restriction is without loss of generality, since every non-atomic concept description, used to describe extensional knowledge, can be given a name in the terminological part of the ontology.

Throughout our work we make use of the unique name assumption for all named individuals. This means that different individual names refer to different domain objects, i.e. formally we have for all interpretations $\mathcal{I}$: $a_1 \neq a_2 \implies a_1^{\mathcal{I}} \neq a_2^{\mathcal{I}}$. This assumption is commonly made in description logics.

In Example 2.6 we provide example declarations for the axioms defined above.

Example 2.6 (Example for Description Logic Axioms)**:**
Given the sets

$$\begin{aligned}\mathbf{CN} &= \{Chair, Course, Person, Department, GraduateCourse, GraduateStudent,\\ &\qquad Professor, Student, UndergraduateCourse\}\\ \mathbf{RN} &= \{headOf, isTaughtBy, memberOf, takes, teaches\}\\ \mathbf{IN} &= \{ani, c1\}\end{aligned}$$

axioms can be built as follows:

- Role equality axiom: $teaches \equiv isTaughtBy^{-}$.
- Concept inclusion axioms: $Professor \sqsubseteq Person, \exists takes.\top \sqsubseteq Student$.

- Individual axioms: $Person(ani), Course(c1), takes(ani, c1)$.

Definition 2.17 (TBoxes, RBoxes, ABoxes, Ontologies, and their Models)**:**
A *TBox* $\mathcal{T}$ is a finite set of general concept inclusion axioms $C_1 \sqsubseteq C_2$. With **ST** we denote the *set of all TBoxes*. An interpretation $\mathcal{I}$ is a *model* for TBox $\mathcal{T}$, denoted with $\mathcal{I} \vDash \mathcal{T}$, if and only if $\mathcal{I}$ satisfies all concept inclusions axioms in $\mathcal{T}$. A *RBox* $\mathcal{R}$ is a finite set of general role inclusion axioms and role transitivity axioms. With **SR** we denote the *set of all RBoxes*. An interpretation $\mathcal{I}$ is a *model* for an RBox $\mathcal{R}$, denoted with $\mathcal{I} \vDash \mathcal{R}$, if and only if $\mathcal{I}$ satisfies all axioms in $\mathcal{R}$. An *ABox* $\mathcal{A}$ is a set of concept assertion axioms and role assertion axioms. With **SA** we denote the *set of all ABoxes*. An interpretation $\mathcal{I}$ is a *model* for ABox $\mathcal{A}$, denoted with $\mathcal{I} \vDash \mathcal{A}$, if and only if $\mathcal{I}$ satisfies all axioms in $\mathcal{A}$. An ontology $\mathcal{O}$ is a tuple $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, such that $\mathcal{T} \in \mathbf{ST}$, $\mathcal{R} \in \mathbf{SR}$, and $\mathcal{A} \in \mathbf{SA}$. With **SO** we denote the *set of all ontologies*. An interpretation $\mathcal{I}$ is a *model* for ontology $\mathcal{O}$, denoted with $\mathcal{I} \vDash \mathcal{O}$, if and only if $\mathcal{I} \vDash \mathcal{T}$, $\mathcal{I} \vDash \mathcal{R}$, and $\mathcal{I} \vDash \mathcal{A}$.

Definition 2.18 (ABox Individuals)**:**
Given an *ABox* $\mathcal{A}$, the set of *ABox individuals* in $\mathcal{A}$, denoted $Ind(\mathcal{A})$, is a subset of **IN** defined as follows:

$$Ind(\mathcal{A}) = \{a \mid C(a) \in \mathcal{A} \vee \exists a_2 \in \mathbf{IN}.\exists R \in \mathbf{Rol}.(R(a, a_2) \in \mathcal{A} \vee R(a_2, a) \in \mathcal{A})\}.$$

We denote the set of *named ABox individuals* in $\mathcal{A}$ with $NInd(\mathcal{A})$. The set of *anonymous ABox individuals* in $\mathcal{A}$ is denoted with $AInd(\mathcal{A})$.

Sometimes, we are only interested in the terminological part of an ontology because some decision problems can be solved by ignoring the ABox $\mathcal{A}$. Terminologies are formally defined in Definition 2.19.

Definition 2.19 (Terminology)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, the *terminology* of $\mathcal{O}$ is the pair $\langle \mathcal{T}, \mathcal{R} \rangle$.

Example 2.7 (Example for an Ontology)**:**
Given the sets

$$\begin{aligned}
\mathbf{CN} &= \{Course, Department, GraduateCourse, GraduateStudent, \\
&\qquad Professor, Student\} \\
\mathbf{RN} &= \{headOf, isTaughtBy, memberOf, takes, teaches\} \\
\mathbf{IN} &= \{c5, ee, mae, sue, zoe\},
\end{aligned}$$

examples for TBoxes, RBoxes, ABoxes, and ontologies are given as follows:

$$\begin{aligned}
\mathcal{T}_{Ex2.7} = \{ & \\
& Student \equiv \exists takes.Course, UndergraduateCourse \sqsubseteq Course \\
\} & \\
\mathcal{R}_{Ex2.7} = \{ & headOf \sqsubseteq memberOf, teaches \equiv isTaughtBy^{-}\} \\
\mathcal{A}_{Ex2.7} = \{ & \\
& Department(ee), Professor(mae), UndergraduateCourse(c5), \\
& Student(sue), Student(zoe), \\
& headOf(mae, ee), teaches(mae, c5), takes(sue, c5), takes(zoe, c5) \\
& \} \\
\mathcal{O}_{Ex2.7} = & \langle \mathcal{T}_{Ex2.7}, \mathcal{R}_{Ex2.7}, \mathcal{A}_{Ex2.7} \rangle
\end{aligned}$$

For instance, in TBox $\mathcal{T}_{Ex2.7}$, we define that each student has to take at least one course. Furthermore, only students take courses, and undergraduate courses are special courses. In RBox $\mathcal{R}_{Ex2.7}$, we define that the role $headOf$ is is a sub role of $memberOf$ and that the roles $teaches$ and $isTaughtBy^{-}$ are equivalent. In ABox $\mathcal{A}_{Ex2.7}$, we define knowledge about the domain objects in an application domain, for instance that individual zoe is a student.

Definition 2.20 (TBox Concept Closure)**:**
Given a TBox $\mathcal{T}$, the *concept closure of* $\mathcal{T}$, denoted $clos(\mathcal{T})$, is defined as

$$clos(\mathcal{T}) = \bigcup_{C_1 \sqsubseteq C_2 \in \mathcal{T}} (clos(nnf(\neg C_1)) \cup clos(C_2)).$$

Example 2.8 (Example for TBox Concept Closure)**:**
Given the TBox

$$\mathcal{T}_{Ex2.8} = \{Student \equiv \exists takes.Course, UndergraduateCourse \sqsubseteq Course\},$$

the concept closure of $\mathcal{T}_{Ex2.8}$ is

$$\begin{aligned}
clos(\mathcal{T}_{Ex2.8}) = \{ & \\
& Student, \neg Student, Course, \neg Course, UndergraduateCourse, \\
& \neg UndergraduateCourse, \exists takes.Course, \forall takes.\neg Course \\
& \}.
\end{aligned}$$

Please note that in Example 2.8, we implicitly split up a concept equivalence axiom into two concept inclusion axioms for deriving the TBox concept closure of $\mathcal{T}_{Ex2.8}$.

Example 2.9 (Example Interpretation)**:**
One example interpretation $\mathcal{I}_{Ex2.9}$, which models $\mathcal{O}_{Ex2.7}$, i.e. we have $\mathcal{I}_{Ex2.9} \vDash \mathcal{O}_{Ex2.7}$, is as follows:

$$\begin{aligned}
\Delta_{\mathcal{I}_{Ex2.9}} &= \{\delta_1, \delta_2, \delta_3, \delta_4, \delta_5\} \\
Course^{\mathcal{I}_{Ex2.9}} &= \{\delta_3\} \\
Department^{\mathcal{I}_{Ex2.9}} &= \{\delta_1\} \\
Professor^{\mathcal{I}_{Ex2.9}} &= \{\delta_2, \delta_4\} \\
Student^{\mathcal{I}_{Ex2.9}} &= \{\delta_4, \delta_5\} \\
UndergraduateCourse^{\mathcal{I}_{Ex2.9}} &= \{\delta_3\} \\
headOf^{\mathcal{I}_{Ex2.9}} &= \{(\delta_2, \delta_1)\} \\
memberOf^{\mathcal{I}_{Ex2.9}} &= \{(\delta_2, \delta_1)\} \\
takes^{\mathcal{I}_{Ex2.9}} &= \{(\delta_4, \delta_3), (\delta_5, \delta_3)\} \\
teaches^{\mathcal{I}_{Ex2.9}} &= \{(\delta_2, \delta_3)\} \\
isTaughtBy^{\mathcal{I}_{Ex2.9}} &= \{(\delta_3, \delta_2)\} \\
c5^{\mathcal{I}_{Ex2.9}} &= \delta_3 \\
ee^{\mathcal{I}_{Ex2.9}} &= \delta_1 \\
mae^{\mathcal{I}_{Ex2.9}} &= \delta_2 \\
sue^{\mathcal{I}_{Ex2.9}} &= \delta_4 \\
zoe^{\mathcal{I}_{Ex2.9}} &= \delta_5.
\end{aligned}$$

Please note that interpretation $\mathcal{I}_{Ex2.9}$ assigns more members to the concept description $Professor$ than strictly enforced by the ontology $\mathcal{O}_{Ex2.7}$. For example, individual *sue* is not obligatorily an instance of concept description $Professor$. In fact, ontology $\mathcal{O}_{Ex2.7}$ allows models with *sue* being mapped to $\neg Professor$ as well. This is an important difference to databases, where partial knowledge is usually not assumed. In databases, if a fact is not present in the system, then it is usually assumed to be false. See [Rei77] for more information about closed world assumption in database systems, and [PSH07] for more information about closed and open world assumption with respect to ontologies.

We now have defined the syntax and semantics of description-logic based ontologies. In the following, we focus on decision problems for ontologies.

2.2.3 Decision Problems for Ontologies

2.2.3.1 Boolean Decision Problems

First, we briefly repeat decision problems for the terminological part of an ontology, i.e. we assume that the assertional part is empty, or at least not considered.

Definition 2.21 (Boolean Decision Problems for Role Descriptions)**:**
A role description R_1 is *subsumed* by a role description R_2 *with respect to an ontology* $\mathcal{O}$, denoted $\mathcal{O} \models R_1 \sqsubseteq R_2$, if and only if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{O} \implies R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$. A role description R_1 is *equivalent* to a role description R_2 *with respect to an ontology* $\mathcal{O}$, denoted $\mathcal{O} \models R_1 \equiv R_2$, if and only if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{O} \implies R_1^{\mathcal{I}} = R_2^{\mathcal{I}}$.

Since the assertional part of an ontology is not relevant for deciding boolean decision problems for role descriptions, we sometimes write $\langle \mathcal{T}, \mathcal{R} \rangle$ instead of $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, e.g. $\langle \mathcal{T}, \mathcal{R} \rangle \models R_1 \sqsubseteq R_2$ instead of $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R_1 \sqsubseteq R_2$.

Definition 2.22 (Boolean Decision Problems for Concept Descriptions)**:**
A concept description C is *satisfiable* with respect to an ontology $\mathcal{O}$ if and only if there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{O}$ and $C^{\mathcal{I}} \neq \emptyset$. A concept description C_1 is *subsumed* by a concept description C_2 with respect to an ontology $\mathcal{O}$, denoted $\mathcal{O} \models C_1 \sqsubseteq C_2$, if and only if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \models \mathcal{O} \implies {C_1}^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$. Given a set $\mathbf{S}$ of concept inclusion axioms, we write $\mathcal{O} \models \mathbf{S}$ if $\mathcal{O} \models C_1 \sqsubseteq C_2$ for each $C_1 \sqsubseteq C_2 \in \mathbf{S}$. A concept description C_1 is *equivalent* to a concept description C_2 with respect to an ontology $\mathcal{O}$, denoted $\mathcal{O} \models C_1 \equiv C_2$, if and only if for all interpretations $\mathcal{I}$ we have $\mathcal{I} \models \mathcal{O} \implies C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$. A concept description C_1 is *disjoint* from a concept description C_2 with respect to an ontology $\mathcal{O}$, denoted $\mathcal{O} \models C_1 \oslash C_2$, if and only if for all interpretations $\mathcal{I}$ we have $\mathcal{I} \models \mathcal{O} \implies C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \emptyset$.

Concept satisfiability checking is a key decision problem for description logics. Many other decision problems can be reduced to concept satisfiability problems, as shown in [BCM+07] and [Smo88]. For instance, the problem of concept description subsumption can be reduced to concept description satisfiability as follows: We have $\mathcal{O} \models C_1 \sqsubseteq C_2$ if and only if $\neg C_2 \sqcap C_1$ is unsatisfiable with respect to $\mathcal{O}$. Please note that this transformation only works in the presence of full negation. However, this does not necessarily lead to an optimal algorithm.

Given the axioms in a TBox $\mathcal{T}$ and an RBox $\mathcal{R}$, one can define satisfiability of the terminology $\langle \mathcal{T}, \mathcal{R} \rangle$, i.e. whether there exists a model for $\langle \mathcal{T}, \mathcal{R} \rangle$ at all.

Definition 2.23 (TBox and RBox Satisfiability)**:**
The terminology of $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ is *satisfiable* if and only if there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{R}$. Satisfiable terminologies are also called *consistent*, all other terminologies are called *unsatisfiable* or *inconsistent*.

Example 2.10 (Example Solutions of Decision Problems for Terminologies)**:**
Given an ontology $\mathcal{O}_{Ex2.10} = \langle \mathcal{T}_{Ex2.10}, \mathcal{R}_{Ex2.10}, \mathcal{A}_{Ex2.10} \rangle$, where

- $\mathcal{T}_{Ex2.10} = \{GraduateStudent \sqsubseteq Student, Student \sqsubseteq \forall takes.Course\}$,
- $\mathcal{R}_{Ex2.10} = \{\}$, and
- $\mathcal{A}_{Ex2.10} = \{\}$,

some decision problems for $\mathcal{O}_{Ex2.10}$ and their solutions are:

- The concept description $Student \sqcap \exists takes.\neg Course$ is unsatisfiable with respect to $\mathcal{O}_{Ex2.10}$.
- The concept description $GraduateStudent$ is subsumed by $Student$ with respect to $\mathcal{O}_{Ex2.10}$.
- The terminology of $\mathcal{O}_{Ex2.10}$ is consistent.

In the following, we do not only consider terminological information, but we take into account assertional information as well. The basic decision problem is consistency again.

Definition 2.24 (Ontology Consistency)**:**
An ontology $\mathcal{O}$ is *consistent* if and only if there exists an interpretation $\mathcal{I}$, such that we have $\mathcal{I} \vDash \mathcal{O}$. An ontology which is not consistent is called *inconsistent*.

In general, we assume that an ontology is initially consistent. This is further discussed in Chapter 3.

Definition 2.25 (Instance Checking and Relation Checking)**:**
A named individual $a \in NInd(\mathcal{A})$ is an *instance* of concept description C with respect to an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, denoted $\mathcal{O} \vDash C(a)$, if and only if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash \mathcal{O} \implies a^{\mathcal{I}} \in C^{\mathcal{I}}$. A named individual $a_1 \in NInd(\mathcal{A})$ is *related* to a named individual $a_2 \in NInd(\mathcal{A})$ via role description R with respect to an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if and only if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash \mathcal{O} \implies (a_1^{\mathcal{I}}, a_2^{\mathcal{I}}) \in R^{\mathcal{I}}$.

The problem of instance checking can be easily reduced to consistency checking.

Proposition 2.1 (Reduction of Instance Checking to Inconsistency)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description C, and a named individual $a \in NInd(\mathcal{A})$, $\mathcal{O} \vDash C(a)$ if and only if $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a)\} \rangle$ is inconsistent.

Proof of Proposition 2.1. [BCM+07]. □

In Example 2.11, we show some solutions for decision problems with respect to an example ontology.

Example 2.11 (Example Solutions of Decision Problems for Ontologies)**:**
Given an ontology $\mathcal{O}_{Ex2.11} = \langle \mathcal{T}_{Ex2.11}, \mathcal{R}_{Ex2.11}, \mathcal{A}_{Ex2.11} \rangle$, where

- $\mathcal{T}_{Ex2.11} = \{GraduateStudent \sqsubseteq Student, Student \sqsubseteq \forall takes.Course\}$,
- $\mathcal{R}_{Ex2.11} = \{\}$, and
- $\mathcal{A}_{Ex2.11} = \{Student(zoe), takes(zoe, c5)\}$,

some decision problems for $\mathcal{O}_{Ex2.11}$ and their solutions are:

- The ontology $\mathcal{O}_{Ex2.11}$ is consistent.

- The individual $c5$ is an instance of the concept description $Course$.
- The individuals zoe and $c5$ are related via role description $takes$.

2.2.3.2 Computational Problems

Boolean decision problems build the foundation for computational problems.

Definition 2.26 (Role Classifications over Ontologies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *role classification*, denoted $\underline{rc}^{\mathcal{O}}$, is a subset of $\mathbf{Rol} \times \mathbf{Rol}$, such that $\underline{rc}^{\mathcal{O}}(R_1, R_2) \iff \mathcal{O} \vDash R_1 \sqsubseteq R_2$. Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *transitivity classification* for $\mathcal{O}$, denoted $\underline{rtc}^{\mathcal{O}}$, is defined as $\underline{rtc}^{\mathcal{O}} = \{R_1 \in \mathbf{Rol} \mid \mathcal{O} \vDash Trans(R_1)\}$.

Since the ABox is not necessary for role classification and transitivity classification, we often use the the terminology of an ontology as an index, e.g. $\underline{rtc}^{\langle \mathcal{T}, \mathcal{R} \rangle}$ instead of $\underline{rtc}^{\mathcal{O}}$ and $\underline{rc}^{\langle \mathcal{T}, \mathcal{R} \rangle}$ instead of $\underline{rc}^{\mathcal{O}}$.

Definition 2.27 (Concept Classification over Ontologies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *concept classification*, denoted $\underline{cc}^{\mathcal{O}}$, is a subset of $\mathbf{AtCon} \times \mathbf{AtCon}$, such that $\underline{cc}^{\mathcal{O}}(C_1, C_2) \iff \mathcal{O} \vDash C_1 \sqsubseteq C_2$.

The computational problem of instance retrieval is to find all named individuals which are instance of a given concept description C. It can easily be seen that the problem of instance retrieval can in principle be reduced to instance checking.

Definition 2.28 (Instance Retrieval over Ontologies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an atomic concept description C, an *instance retrieval result*, denoted $\underline{ir}_C^{\mathcal{O}}$, is the set of named individuals, such that

$$\underline{ir}_C^{\mathcal{O}} = \{a \in NInd(\mathcal{A}) \mid \mathcal{O} \vDash C(a)\}.$$

Please note that by Definition 2.28 and Definition 2.25, the elements in $\underline{ir}_C^{\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle}$ are restricted to the individuals in $NInd(\mathcal{A})$.

The computational problem relation retrieval is to find all pairs of named individuals which are related by a given role description R. The formal definition of relation retrieval is given in Definition 2.29. Please note that the definition of relation retrieval is not so common in description logic research.

Definition 2.29 (Relation Retrieval)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a role description R, a *relation retrieval result*, denoted $\underline{rr}_R^{\mathcal{O}}$, is a set of pairs of named individuals, such that

$$\underline{rr}_R^{\mathcal{O}}(a_1, a_2) \iff \mathcal{O} \vDash R(a_1, a_2).$$

Please note again that by Definition 2.29 and Definition 2.25, the elements in $\underline{rr}_R^{\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle}$ are restricted to the individual pairs in $NInd(\mathcal{A}) \times NInd(\mathcal{A})$.

2.2.4 Naming Schemes

Different classes of description logics were proposed by the research community in the past. The need for gradually different expressivity is motivated by different users and their requirements. Second, when the logical constructors are limited, it might be possible to prove more interesting computational properties about the description logic of concern. It is well known that there is a trade off between the expressive power of a description logic and its computational properties [BCM+07, HST00a]. In the following, we give a short overview of different members of the description logics family.

The description logic $\mathcal{FL}_0$ was first discussed in [Baa90]. The language has two constructors for specifying concept descriptions: concept conjunction and value restrictions. The decision problems considered for $\mathcal{FL}_0$ are focused on terminological reasoning. The concept subsumption problem for $\mathcal{FL}_0$ was shown to be PSPACE for least fixpoint semantics in [Baa90], for greatest fixpoint semantics in [Baa96] and for descriptive semantics in [KN03].

Extending the description logic $\mathcal{FL}_0$ with existential restrictions leads to the description logic $\mathcal{FL}^-$, introduced in [LB87]. In the same paper, the authors also introduce a more general general description logic $\mathcal{FL}$, which allows a form of „existential restriction“ in addition. It is shown in [LB87] that the concept subsumption problem for $\mathcal{FL}^-$ is in P and that concept subsumption for $\mathcal{FL}$ is in coNP. In recent years, research on $\mathcal{FL}$ and its sublogics has been more and more reduced. The reasons are manifold. First, users and researchers are always looking for more expressivity. Second, improved technology allows for implementation of practical reasoning procedures for more expressive description logics. And third, other lightweight description logics got into the focus of researchers: $\mathcal{EL}$, its extensions, and the family of description logics summarized as *DL-LITE*.

The description logic $\mathcal{EL}$ was first defined in [BKM99]. The concept subsumption problem for $\mathcal{EL}$ was shown to be PSPACE for least fixpoint semantics, for greatest fixpoint semantics and for descriptive semantics in [Baa02]. It is shown in [BBL05] that the description logic $\mathcal{EL}$ can be extended by further constructors without losing tractability. Examples for these description logics are $\mathcal{EL}+$ and $\mathcal{EL}^{++}$. There exist further extensions of $\mathcal{EL}^{++}$, e.g. adding reflexive roles and range restrictions, which are proposed and discussed in [BBL08]. Reasoning over these extensions is still tractable under certain syntactic restrictions.

For an extensive overview over *DL-LITE* see [ACKZ09].

Next, we consider the family of $\mathcal{AL}$ languages. More expressive languages are built from the $\mathcal{AL}$ language by adding constructors. The syntax for $\mathcal{ALC}$ is shown in Figure 2.2. $\mathcal{ALC}$ is a syntactic variant of the propositional modal logic K_m [Sch91]. The concept subsumption problem for $\mathcal{ALC}$ and general TBoxes is shown in [DM00] to be in EXPTIME, hardness was shown in [Sch91].

The extension $\mathcal{ALCHI}$ is obtained from $\mathcal{ALC}$ by allowing inverse roles and role hierarchies. Further extending $\mathcal{ALCHI}$ with transitive roles yields the description logic $\mathcal{SHI}$

Figure 2.2 Restrictions on the description logic $\mathcal{ALC}$

Role description constructors:	$R := S$
Concept description constructors:	$C_1, C_2 := \top, \bot, A, \neg C_1$ $C_1 \sqcap C_2, C_1 \sqcup C_2, \forall R.C_1, \exists R.C_1$
TBox axioms:	$C_1 \sqsubseteq C_2$
RBox axioms:	none

[HS99], and extending $\mathcal{SHI}$ with cardinality restrictions yields $\mathcal{SHIQ}$ [HST00b]. These extensions do not change the upper bound of the complexity for deciding concept subsumption, concept satisfiability, and ABox consistency, as shown in [Tob01]. The last description logic we would like to mention is $\mathcal{SHOIQ}$ [HS07], which, in addition, allows for the use of nominals, i.e. usage of individual names in the TBox. It is shown in [Tob01] that concept satisfiability and ABox consistency for $\mathcal{SHOIQ}$ turn out to be in NEXPTIME.

In the remaining part of our work, whenever we mention the word ontology, we mean an ontology which only uses constructors from the $\mathcal{SHI}$ description logic, unless mentioned otherwise.

2.2.5 Reasoning Procedures

In the following, we give an overview of reasoning procedures to solve decision problems over ontologies. In our work, we focus on tableau-based reasoning procedures. Especially for description logics with full negation, tableau algorithms have turned out to be very useful [BCM+07]. There exist various other approaches, for instance automata-based approaches[BHLW03] or approaches motivated by logic programming, e.g. using resolution [HMS04], or theorem provers for more expressive logics [RV02, KL04].

According to [BS00], one can distinguish four different phases for description logics (reasoning systems) research:

1. First implementations of efficient, but incomplete structural algorithms, e.g. KL-ONE [BS85].
2. First theoretical complexity and undecidability results obtained. It was shown for example that subsumption is undecidable in KL-ONE [SS89]. A modification of KL-ONE, called CLASSIC, was proposed in [BBM+92], using a less expressive description logic.
3. First tableau algorithms for expressive description logics. In addition a thorough complexity analysis on description logics started. The reasoning system KRIS [BH91] was, for instance, the first reasoning system which supported a complete tableau algorithm for an expressive description logic.

4. Optimized algorithms and implementations for expressive description logics, e.g. the system DLP [PS98], FaCT [Hor98], FaCT++ [FS06], Pellet [SPG+07], and Racer [HMW04].

The first tableau algorithm was presented by [SSS91] for satisfiability of $\mathcal{ALC}$-concepts. Since then similar approaches have been used to obtain sound and complete satisfiability algorithms for many extensions of $\mathcal{ALC}$, e.g $\mathcal{SHIQ}$ [HST00b], $\mathcal{SHOQ}$ [HS01], $\mathcal{SHOIQ}$ [HS07], and $\mathcal{SROIQ}$ [HKS06].

The purpose of a tableau algorithm is to check consistency of a given ontology $\mathcal{O}$. As pointed out before, (in-)consistency is one of the basic decision problems. Other decision problems can be reduced to inconsistency checking. Given an input ontology $\mathcal{O}$, a tableau algorithm tries to generate a finite representation for a model of $\mathcal{O}$. If the algorithm succeeds, the algorithm computed a compact model representation and therefore shows that the ontology is consistent. If the algorithm fails, then the output is false, i.e. there does not exists a model for $\mathcal{O}$.

There are different representations used as a basis for tableau algorithms, e.g. graph-based and ABox-based. Here, we select the ABox-based view on tableau algorithms as for instance chosen in [BS01]. In each step, a tableau algorithm applies one tableau rule to an intermediate ABox, in order to try to generate a model representation.

Depending on the expressivity of the description logic used for ontology $\mathcal{O}$, there are different rules necessary. In general, there is one tableau rule for each constructor, with some exceptions (see below).

We define additional notions on ABoxes, which are necessary for the definition of a tableau algorithm.

Definition 2.30 (Role Successor and Role Predecessor in ABoxes)**:**
Let $\sqsubseteq^*$ be the transitive-reflexive closure of the role hierarchy (as defined in [HS99]). Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and two individuals $a_1 \in Ind(\mathcal{A})$ and $a_2 \in Ind(\mathcal{A})$, individual a_2 is called R_1-*successor* of a_1 (*with respect to* $\mathcal{O}$) if there exists a $R_2 \in \mathbf{Rol}$, such that $R_2(a_1, a_2) \in \mathcal{A}$ and $R_2 \sqsubseteq^* R_1$. The individual a_2 is called R_1-*predecessor* of a_1 (*with respect to* $\mathcal{O}$) if there exists a $R_2 \in \mathbf{Rol}$, such that $R_2(a_2, a_1) \in \mathcal{A}$ and $R_2 \sqsubseteq^* R_1$.

Definition 2.31 (Role Neighbors and Neighbors in ABoxes)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and two individuals a_1 and a_2, individual a_2 is a R-*neighbor of* individual a_1 (*with respect to* $\mathcal{O}$) if and only if a_2 is a R-successor of a_1, or a_1 is a R^--successor of a_2. The individuals a_1 and a_2 are called *neighbors* (*with respect to* $\mathcal{O}$) if there exists a $R \in \mathbf{Rol}$, such that a is a R-neighbor of a_1 or a_1 is a R_2-neighbor of a_2.

Based on these neighbor relationships we define the set of tableau rules below. A tableau rule application is a (in-)consistency-preserving application of a tableau rule to an ABox $\mathcal{A}$, such that the output ABoxes are inconsistent if and only if the input ABox is inconsistent.

Definition 2.32 (Tableau Rule Application)**:**
Let X be the name of a tableau rule. Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *tableau rule application* is a function $tabrapp_{X,\mathcal{T},\mathcal{R}} : \mathbf{SA} \rightarrow \wp(\mathbf{SA})$, such that $\mathcal{O}$ is inconsistent if and only if for all $\mathcal{A}_i \in tabrapp_{X,\mathcal{T},\mathcal{R}}(\mathcal{A})$ we have that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle$ is inconsistent.

In the following, we define a number of tableau rules, similar to the way they have been introduced in the literature [BCM$^+$07, BS01]. In particular, a tableau algorithm for the description logic $\mathcal{SHI}$ has been defined in [HS99]. Our definition of the algorithm is a slightly changed version, since we attach to each rule a variable assignment, which indicates, how the tableau rule is applied. We need this additional information for proofs about tableau properties in Chapter 3.

Definition 2.33 (Variable Assignment)**:**
Given a set of variables **VAR**, a *variable assignment* is a function $\pi : \mathbf{VAR} \rightarrow \mathbf{IN}$.

Before we introduce the tableau rules, we need to discuss the notion of blocking. There exist several levels of blocking, depending on the usage of concept description constructors in an ontology:

- *Subset-blocking* [BBH96]: In the presence of general concept inclusions, subset-blocking is employed in order to ensure termination of the tableau algorithm.
- *Dynamic-blocking* [HS99]: In the additional presence of inverse roles, blocking is dynamic, i.e., blocked nodes (and their sub-branches) can be un-blocked and blocked again later.
- *Pairwise-blocking* [HST00a]: In the additional presence of number restrictions, pairs of nodes are blocked rather than single nodes only.

For the description logic $\mathcal{SHI}$, we assume that ABox individuals are (directly/indirectly) blocked with the dynamic-blocking strategy. For the details please refer to [HS99].

Definition 2.34 ($\sqcap$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description $C = C_1 \sqcap C_2$, and a variable assignment π, the $\sqcap$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C(\pi(x)) \in \mathcal{A}$,
2. $\pi(x)$ is not indirectly blocked, and
3. $\{C_1(\pi(x)), C_2(\pi(x))\} \nsubseteq \mathcal{A}$.

Given that the $\sqcap$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept description $C = C_1 \sqcap C_2$, the *tableau rule application* of the $\sqcap$-tableau rule with π and C is defined as

$$tabrapp^{\pi, C_1 \sqcap C_2}_{\sqcap,\mathcal{T},\mathcal{R}}(\mathcal{A}) = \{\mathcal{A} \cup \{C_1(\pi(x)), C_2(\pi(x))\}\}.$$

Depending on the input ABox, π, and C there are several rule applications possible. The set of all these *possible* tableau rule applications of the $\sqcap$-tableau rule is denoted with $\mathbf{tabrapps}_{\sqcap,\mathcal{T},\mathcal{R}}$.

In order to illustrate our representation of tableau rules, we present one example tableau rule application in Example 2.12.

Example 2.12 (Example $\sqcap$-Tableau Rule Application)**:**
Given an ontology $\mathcal{O}_{Ex2.12} = \langle \mathcal{T}_{Ex2.12}, \mathcal{R}_{Ex2.12}, \mathcal{A}_{Ex2.12} \rangle$, where

- $\mathcal{T}_{Ex2.12} = \emptyset$,
- $\mathcal{R}_{Ex2.12} = \emptyset$, and
- $\mathcal{A}_{Ex2.12} = \{Student \sqcap Person(zoe), Course \sqcap UndergraduateCourse(c5)\}$,

we have that:

- the $\sqcap$-tableau rule is applicable to $\mathcal{O}_{Ex2.12}$, for a variable assignment $\{x \rightarrow zoe\}$ and concept description $Student \sqcap Person$,
- the $\sqcap$-tableau rule is applicable to $\mathcal{O}_{Ex2.12}$, for a variable assignment $\{x \rightarrow c5\}$ and concept description $Course \sqcap UndergraduateCourse$, and
- $tabrapp_{\sqcap,\mathcal{T},\mathcal{R}}^{\{x \rightarrow zoe\}, Student \sqcap Person}(\mathcal{A}_{Ex2.12}) = \mathcal{A}_{Ex2.12} \cup \{Student(zoe), Person(zoe)\}$.

Definition 2.35 ($\sqcup$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description $C = C_1 \sqcup C_2$, and a variable assignment π, the $\sqcup$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C(\pi(x)) \in \mathcal{A}$,
2. $\pi(x)$ is not indirectly blocked, and
3. $\{C_1(\pi(x)), C_2(\pi(x))\} \cap \mathcal{A} = \emptyset$.

Given that the $\sqcup$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept description $C = C_1 \sqcup C_2$, the *tableau rule application* of the $\sqcup$-tableau rule with π and C is defined as

$$tabrapp_{\sqcup,\mathcal{T},\mathcal{R}}^{\pi, C_1 \sqcup C_2}(\mathcal{A}) = \{\mathcal{A} \cup \{C_1(\pi(x))\}, \mathcal{A} \cup \{C_2(\pi(x))\}\}.$$

The set of all possible tableau rule applications of the $\sqcup$-tableau rule is denoted with $\mathbf{tabrapps}_{\sqcup,\mathcal{T},\mathcal{R}}$.

Definition 2.36 ($\exists$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description $C = \exists R.C_1$, and a variable assignment π, the $\exists$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C(\pi(x)) \in \mathcal{A}$,
2. $\pi(x)$ is not blocked,
3. $\pi(x)$ has no anonymous R-neighbor a_1 with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, such that $C_1(a_1) \in \mathcal{A}$, and
4. $\pi(x_1)$ is a fresh anonymous individual name, i.e. $\pi(x_1) \in (\mathbf{AIN} \setminus Ind(\mathcal{A}))$.

Given that the $\exists$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept description $C = \exists R.C_1$, the *tableau rule application* of the $\exists$-tableau rule with π and C is defined as

$$tabrapp^{\pi, \exists R.C_1}_{\exists, \mathcal{T}, \mathcal{R}}(\mathcal{A}) = \{\mathcal{A} \cup \{R(\pi(x), \pi(x_1)), C_1(\pi(x_1))\}\}.$$

The set of all syntactically possible tableau rule applications of the $\exists$-tableau rule is denoted with $\mathbf{tabrapps}_{\exists, \mathcal{T}, \mathcal{R}}$.

Please note that the fourth criterion in Definition 2.36 could be seen as a post-condition. However, we fix the chosen fresh individual name already in the pre-condition. This makes our definitions and proofs easier. In Example 2.13, we show another example for a tableau rule application. This example indicates why we attach variable assignments to all tableau rule applications.

Example 2.13 (Example $\exists$-Tableau Rule Application)**:**
Given an ontology $\mathcal{O}_{Ex2.13} = \langle \mathcal{T}_{Ex2.13}, \mathcal{R}_{Ex2.13}, \mathcal{A}_{Ex2.13} \rangle$, where

- $\mathcal{T}_{Ex2.13} = \emptyset$,
- $\mathcal{R}_{Ex2.13} = \emptyset$, and
- $\mathcal{A}_{Ex2.13} = \{\exists takes.UndergraduateCourse(zoe), takes(zoe, c5), Course(c5)\}$,

we have that:

- The $\exists$-tableau rule is applicable to $\mathcal{O}_{Ex2.13}$, for a variable assignment $\{x \to zoe, x_1 \to new\}$ and concept description $\exists takes.UndergraduateCourse$.
- $tabrapp^{\{x \to zoe, x_1 \to new\}, \exists takes.UndergraduateCourse}_{\exists, \mathcal{T}, \mathcal{R}}(\mathcal{A}_{Ex2.13}) =$
  $\mathcal{A}_{Ex2.13} \cup \{takes(zoe, new), UndergraduateCourse(new)\}$.

Definition 2.37 ($\forall$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description $C = \forall R.C_1$, and a variable assignment π, the $\forall$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C(\pi(x)) \in \mathcal{A}$,
2. $\pi(x)$ is not indirectly blocked, and
3. $\pi(x)$ has an R-neighbor $\pi(x_1)$ with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, such that $C_1(\pi(x_1)) \notin \mathcal{A}$.

Given that the $\forall$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept description $C = \forall R.C_1$, the *tableau rule application* of the $\forall$-tableau rule with π and C is defined as

$$tabrapp^{\pi,\forall R.C_1}_{\forall,\mathcal{T},\mathcal{R}}(\mathcal{A}) = \{\mathcal{A} \cup \{C_1(\pi(x_1))\}\}.$$

The set of all possible tableau rule applications of the $\forall$-tableau rule is denoted with $\mathbf{tabrapps}_{\forall,\mathcal{T},\mathcal{R}}$.

In the following the verbalization *propagation of a concept description* C is used to describe the process of applying the $\forall$-tableau rule and adding C to $\pi(x_1)$.

Definition 2.38 ($\forall_+$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept description $C = \forall R.C_1$, and a variable assignment π, the $\forall_+$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C(\pi(x)) \in \mathcal{A}$,
2. $\pi(x)$ is not indirectly blocked,
3. there exists some R_2 with $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models Trans(R_2)$ and $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R_2 \sqsubseteq R$, and
4. $\pi(x)$ has a R_2-neighbor $\pi(x_1)$ with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, with $\forall R_2.C_1(\pi(x_1)) \notin \mathcal{A}$.

Given that the $\forall_+$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept description $C = \forall R.C_1$, the *tableau rule application* of the $\forall_+$-tableau rule with π and C is defined as

$$tabrapp^{\pi,\forall R.C_1}_{\forall_+,\mathcal{T},\mathcal{R}}(\mathcal{A}) = \{\mathcal{A} \cup \{\forall R_2.C_1(\pi(x_1))\}\}.$$

The set of all possible tableau rule applications of the $\forall_+$-tableau rule is denoted with $\mathbf{tabrapps}_{\forall_+,\mathcal{T},\mathcal{R}}$.

Definition 2.39 ($TBox$-Tableau Rule Applicability and Application)**:**
Given an ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a concept inclusion axiom $C_1 \sqsubseteq C_2$, and a variable assignment π, the $TBox$-tableau rule is *applicable* to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if

1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$,
2. $\pi(x)$ is not indirectly blocked, and
3. $(\neg C_1 \sqcup C_2)(\pi(x)) \notin \mathcal{A}$.

Given that the $TBox$-tableau rule is applicable to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for a variable assignment π and concept inclusion axiom $C_1 \sqsubseteq C_2$, the *tableau rule application* of the $TBox$-tableau rule with π and C is defined as

$$tabrapp^{\pi,C_1 \sqsubseteq C_2}_{TBox,\mathcal{T},\mathcal{R}}(\mathcal{A}) = \{\mathcal{A} \cup \{(\neg C_1 \sqcup C_2)(\pi(x))\}\}.$$

The set of all possible tableau rule applications of the $TBox$-tableau rule is denoted with $\mathbf{tabrapps}_{TBox,\mathcal{T},\mathcal{R}}$.

In Definition 2.40, we define the set of all possible tableau rule applications.

Definition 2.40 (Tableau Rule Application Set)**:**
The *set of all possible tableau rule applications*, denoted $\mathbf{tabrapps}_{,\mathcal{T},\mathcal{R}}$, is defined as

$$\begin{aligned}\mathbf{tabrapps}_{\mathcal{T},\mathcal{R}} =& \mathbf{tabrapps}_{\sqcap,\mathcal{T},\mathcal{R}} \cup \mathbf{tabrapps}_{\sqcup,\mathcal{T},\mathcal{R}} \cup \mathbf{tabrapps}_{\exists,\mathcal{T},\mathcal{R}} \\ & \cup \mathbf{tabrapps}_{\forall,\mathcal{T},\mathcal{R}} \cup \mathbf{tabrapps}_{\forall_+,\mathcal{T},\mathcal{R}} \cup \mathbf{tabrapps}_{TBox,\mathcal{T},\mathcal{R}}.\end{aligned}$$

Please note that $\mathbf{tabrapps}_{\mathcal{T},\mathcal{R}}$ is a *set of functions* over ABoxes, and the functions are parametrized with variable assignments and concept descriptions/concept inclusions axioms (plus a TBox and RBox). Next, we need to define further properties on ABoxes to decide whether the tableau algorithm is finished or not.

Definition 2.41 (ABox Properties)**:**
An ABox $\mathcal{A}$ contains a *clash* if $\{C(a), \neg C(a)\} \subseteq \mathcal{A}$ or $\bot(a) \in \mathcal{A}$ for an individual $a \in Ind(\mathcal{A})$ and concept description C. An ABox $\mathcal{A}$ is called *closed* if $\mathcal{A}$ contains a clash, and *open* otherwise. Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an ABox $\mathcal{A}$ is called *complete* if no tableau rule (Definitions 2.34 to 2.39) is applicable to $\mathcal{O}$, ABox $\mathcal{A}$ is called *incomplete* otherwise.

In Example 2.14, we provide examples for tableau definitions with respect to an example ontology in order to make the definitions above clear.

Example 2.14 (Example for ABox Properties)**:**
Given an ontology $\mathcal{O}_{Ex2.14} = \langle \mathcal{T}_{Ex2.14}, \mathcal{R}_{Ex2.14}, \mathcal{A}_{Ex2.14} \rangle$, where

- $\mathcal{T}_{Ex2.14} = \{Student \sqsubseteq \forall takes.Course\}$,
- $\mathcal{R}_{Ex2.14} = \{\}$, and
- $\mathcal{A}_{Ex2.14} = \{Student(zoe), takes(zoe, c5)\}$,

some examples for ABox properties are:

- $c5$ is a *takes*-neighbor of *zoe* in $\mathcal{A}_{Ex2.14}$,
- *zoe* and $c5$ are neighbors in $\mathcal{A}_{Ex2.14}$,
- The ABox $\mathcal{A}_{Ex2.14} \cup \{\neg Student(zoe)\}$ contains a clash (on individual *zoe*), and
- The ABox

$$\begin{aligned}\mathcal{A}_{Ex2.14} \cup \{ & \\ & (\neg Student \sqcup \forall takes.Course)(zoe), \\ & \forall takes.Course(zoe) \\ & \}\end{aligned}$$

  is incomplete, because the $\forall$-tableau rule is applicable to individual *zoe* and concept description $\forall takes.Course$. Please note that there exist further applicable tableau

rules, e.g. the $TBox$-tableau rule is applicable for $\pi = \{x \rightarrow zoe\}$ and $Student \sqsubseteq \forall takes.Course$,

- The ABox

$$\mathcal{A}_{Ex2.14} \cup \{ \\ (\neg Student \sqcup \forall takes.Course)(zoe), \forall takes.Course(zoe), \\ Course(c5), (\neg Student \sqcup \forall takes.Course)(c5), \neg Student(c5) \\ \}$$

is complete, because no tableau rule is applicable. Furthermore the ABox is open since the ABox does not contain a clash.

Based on the tableau rules, we define the notion of a tableau next. A tableau is an abstraction of a model . From a technical point of view, we look at a tableau as a tree of tableau rule applications, such that the root of the tree is labeled with the input ABox which has to be checked for inconsistency. Tableau rules are applied to the leaf node ABoxes of the tree as long as possible.

Definition 2.42 (Tableau)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *tableau for* $\mathcal{O}$ is a tuple

$$T^{\mathcal{O}} = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle,$$

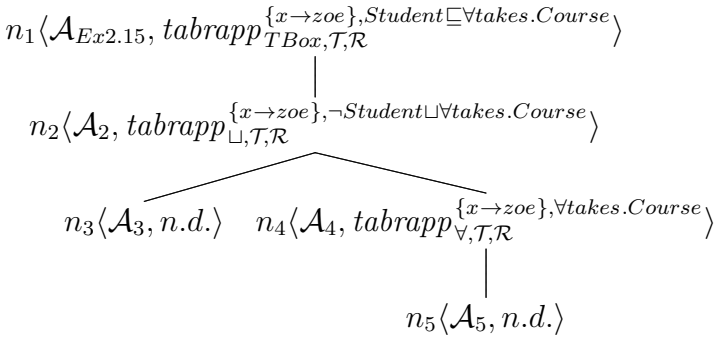
such that

- $\mathbb{T}_{T^{\mathcal{O}}} = \langle \mathbf{N}, root, children \rangle$ is a directed tree,
- $\phi^{abox} : \mathbf{N} \rightarrow \mathbf{SA}$ is a total node labeling function, with $\phi^{abox}(root) = \mathcal{A}$,
- $\phi^{ruleapp} : \mathbf{N} \rightarrow \mathbf{tabrapps}_{\mathcal{T},\mathcal{R}}$ is a node labeling function on all inner nodes of $\mathbb{T}_{T^{\mathcal{O}}}$, such that $\phi^{ruleapp}$ assigns a tableau rule application $tabrapp_{X,\mathcal{T},\mathcal{R}}$ to node n and $tabrapp_{X,\mathcal{T},\mathcal{R}}$ is applicable to $\langle \mathcal{T}, \mathcal{R}, \phi^{abox}(n) \rangle$, and
- for every inner node n of $\mathbb{T}_{T^{\mathcal{O}}}$, we have $\phi^{abox}(children(n)) = (\phi^{ruleapp}(n))(\phi^{abox}(n))$.

Please note the application of ϕ^{abox} to sets of nodes, instead of a single node. The result of that function application to $children(n)$ is, as defined in Definition 2.2, the set of all the ABoxes from the child nodes of n.

In the following, we often refer to tableaux as trees. Formally, we mean $\mathbb{T}_{T^{\mathcal{O}}}$ when we talk about the tableau $T^{\mathcal{O}}$ as a tree. To describe a particular node in a tableau, we sometimes use the following notation: $n\langle \mathcal{A}, tabrapp_{X,\mathcal{T},\mathcal{R}} \rangle$ means that we have $\phi^{abox}(n) = \mathcal{A}$ and $\phi^{ruleapp}(n) = tabrapp_{X,\mathcal{T},\mathcal{R}}$ for a node n in the tableau. With $n.d.$ instead of $tabrapp_{X,\mathcal{T},\mathcal{R}}$ we denote that there is no tableau rule application attached to n. An example for a tableau is shown in Example 2.15.

Figure 2.3 Example of a tableau for $\mathcal{O}_{Ex2.15}$



Example 2.15 (Tableau Example)**:**
Given an ontology $\mathcal{O}_{Ex2.15} = \langle\mathcal{T}_{Ex2.15}, \mathcal{R}_{Ex2.15}, \mathcal{A}_{Ex2.15}\rangle$, where

- $\mathcal{T}_{Ex2.15} = \{Student \sqsubseteq \forall takes.Course\}$,
- $\mathcal{R}_{Ex2.15} = \{\}$, and
- $\mathcal{A}_{Ex2.15} = \{Student(zoe), takes(zoe, c5)\}$,

the tableau depicted in Figure 2.3 is a tableau for $\mathcal{O}_{Ex2.15}$, where

$$\begin{aligned}
\mathcal{A}_2 &= \mathcal{A}_{Ex2.15} \cup \{\neg Student \sqcup \forall takes.Course(zoe)\}\\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{\neg Student(zoe)\}\\
\mathcal{A}_4 &= \mathcal{A}_2 \cup \{\forall takes.Course(zoe)\}\\
\mathcal{A}_5 &= \mathcal{A}_4 \cup \{Course(c5)\}.
\end{aligned}$$

It is easy to see that the tableau is not yet finished since we can still apply additional rules, e.g. the $TBox$-rule to individual $c5$ and concept inclusion $Student \sqsubseteq \forall takes.Course$ at node n_5. Please note that all the leave nodes have no tableau rule application attached (these leaf nodes are labeled with $n.d.$).

In Definition 2.43, we define a special class of tableau, namely proofs, for which every leaf ABox is complete, and hence every path is complete.

Definition 2.43 (Tableau Proof, Result)**:**
Given an ontology $\mathcal{O} = \langle\mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$, a *tableau proof for* $\mathcal{O}$, denoted $TP^{\mathcal{O}}$, is a tableau for $\mathcal{O}$, such that all leaf nodes of $TP^{\mathcal{O}}$ are labeled by ϕ^{abox} with a complete ABox. The *result of a tableau proof* $TP^{\mathcal{O}}$ is *yes* (or *consistent* or *satisfying*) if at least one leaf node is labeled with an open (and complete) ABox, and the result is *no* (or *inconsistent* or *unsatisfying*) otherwise.

Figure 2.4 General tableau algorithm

Input: Ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$

Output: Tableau $T^{\mathcal{O}}$

Algorithm:

Let $T^{\mathcal{O}} = \langle \{n_1\}, n_1, children, \phi^{abox}, \phi^{ruleapp} \rangle$, such that $children = \emptyset$, $\phi^{abox} = \{n_1 \rightarrow \mathcal{A}\}$ and $\phi^{ruleapp} = \emptyset$

While there exists a $tabrapp_{X,\mathcal{T},\mathcal{R}} \in \mathbf{tabrapps}_{\mathcal{T},\mathcal{R}}$ for an open ABox $\mathcal{A}_{leaf}$ labeled by ϕ^{abox} to a leaf node $leaf$ in $T^{\mathcal{O}}$,

Set $\phi^{ruleapp}(leaf) = tabrapp_{X,\mathcal{T},\mathcal{R}}$
For each $\mathcal{A}_{new} \in tabrapp_{X,\mathcal{T},\mathcal{R}}(\mathcal{A}_{leaf})$ do
Add a new node $newleaf$ to $\mathbf{N}$
Set $\phi^{abox}(newleaf)$ to $\mathcal{A}_{new}$
Set $children(leaf) = children(leaf) \cup \{newleaf\}$

Please note the usage of terms *satisfying* and *unsatisfying* in Definition 2.43. The intuition is that the tableau proof either creates a model representation *satisfying* the input ontology or shows that no model can exist.

In Figure 2.4, we define a general tableau algorithm in pseudo-code.

The general tableau algorithm from Figure 2.4 together with the tableau rules in Definitions 2.34 to 2.39, is designed for description logics up to $\mathcal{SHI}$. In order to ensure termination of the given tableau algorithm, it is necessary to avoid an infinite number of rule applications.

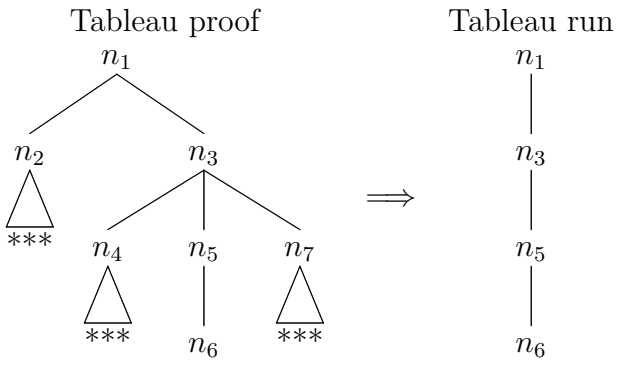
In Proposition 2.2, we show that the introduced tableau algorithm is sufficient to test for (in-)consistencies of ontologies. Please note again that our tableau representation only differs from standard implementations in the attachment of the variable assignment. The variable assignment fixes the individuals involved in each tableau rule application. This is necessary for our proof techniques below.

Proposition 2.2 (Tableau Inconsistency Test)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, $\mathcal{O}$ is inconsistent if and only if there exists an unsatisfying tableau proof $TP^{\mathcal{O}}$ for ontology $\mathcal{O}$.

Proof of Proposition 2.2. A direct consequence of [HS99], since we implement all necessary and sufficient tableau rules and apply dynamic-blocking. □

Proposition 2.3 (Instance Checking with Tableau Inconsistency Test)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an individual a, and a concept description C, $\mathcal{O} \models C(a)$ if and only if there exists a unsatisfying tableau proof $TP^{\mathcal{O}}$ for ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a)\} \rangle$.

Figure 2.5 Intuition of tableau run extraction



Proof of Proposition 2.3. By Proposition 2.1 and Proposition 2.2. □

In order to prove consistency of an ontology, one only needs to find one path of the tableau tree, derived in a tableau proof, such that the leaf of the path is labeled with a complete and open ABox. We define the notion of a tableau run, which represents one path in a tableau proof tree. The intuition of a tableau run, given a tableau proof, is depicted in Figure 2.5.

Definition 2.44 (Tableau Run, Result)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *tableau run for* $\mathcal{O}$ is a tableaux proof

$$RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle,$$

for $\mathcal{O}$, such that for each inner node $n \in \mathbf{N}$:

- the cardinality of $children\ (n)$ is 1 and
- we have $\phi^{abox}(children(n)) \in \phi^{ruleapp}(n)(\phi^{abox}(n))$.

The leaf node of RUN is denoted as $leaf$. The ABox of the leaf node of RUN, denoted $\phi^{abox}\ (leaf)$, is called *leaf ABox of RUN*. The *result of a tableau run RUN* is *yes* (or *consistent* or *satisfying*) if the leaf node is labeled with an open ABox, and the result is *no* (or *inconsistent* or *unsatisfying*) otherwise.

It is easy to see that given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a tableau proof $TP^{\mathcal{O}}$ for $\mathcal{O}$, we have $TP^{\mathcal{O}}$ is satisfying if and only if there exists a satisfying tableau run RUN for $\mathcal{O}$ in $TP^{\mathcal{O}}$.

Below, we assume that subsets (subsets with respect to $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}$) of consistent ontologies are consistent and supersets of inconsistent ontologies are inconsistent without further proof. Both relationships hold because our description logics of concern can directly be mapped to first order logic.

2.3 Running Example

In the following, we introduce an example ontology, which is used throughout the remaining part of the thesis. The example ontology is situated in the university domain and inspired by the Lehigh University Benchmark, introduced in [GPH05]. Sometimes we only use subsets of the example ontology.

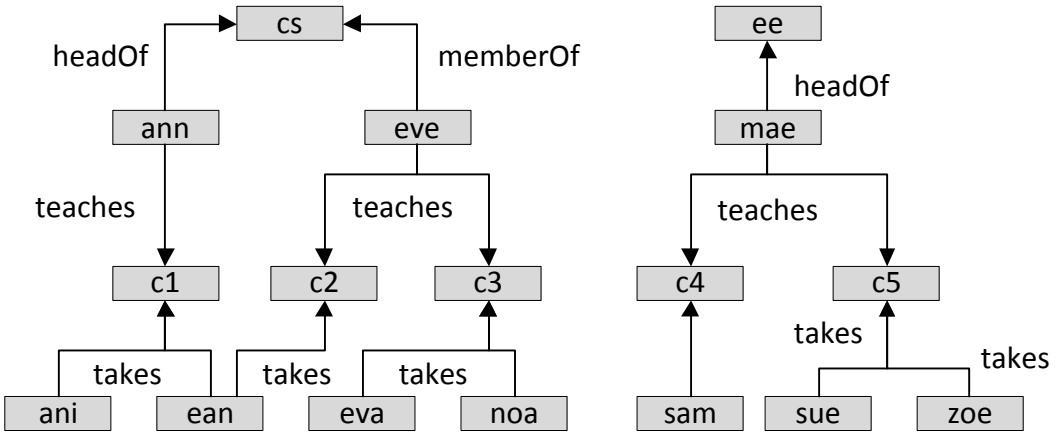
Example 2.16 (Running Example)**:**
The example ontology $\mathcal{O}_{Ex2.16} = \langle \mathcal{T}_{Ex2.16}, \mathcal{R}_{Ex2.16}, \mathcal{A}_{Ex2.16} \rangle$ is defined as follows

$$\begin{aligned}
\mathcal{T}_{Ex2.16} = \{ & \\
& Chair \equiv \exists headOf.Department, \\
& Student \equiv \exists takes.Course, \\
& GraduateStudent \equiv \exists takes.GraduateCourse, \\
& Student \sqsubseteq Person, Professor \sqsubseteq Person, \\
& UndergraduateCourse \sqsubseteq Course, \\
& GraduateCourse \sqsubseteq Course, \\
& GraduateCourse \sqcap UndergraduateCourse \sqsubseteq \bot, \\
& \top \sqsubseteq \forall teaches.Course, \top \sqsubseteq \forall takes.Course, \\
& \top \sqsubseteq \forall memberOf^{-}.Person, \top \sqsubseteq \forall isTaughtBy.Professor, \\
& \exists memberOf.\top \sqsubseteq Person \\
\} &
\end{aligned}$$

$$\mathcal{R}_{Ex2.16} = \{headOf \sqsubseteq memberOf, teaches \equiv isTaughtBy^{-}\}$$

$$\begin{aligned}
\mathcal{A}_{Ex2.16} = \{ & \\
& Department(cs), Department(ee), \\
& Professor(ann), Professor(eve), Professor(mae), \\
& UndergraduateCourse(c1), UndergraduateCourse(c4), \\
& UndergraduateCourse(c5), \\
& GraduateCourse(c2), GraduateCourse(c3), \\
& Student(ani), Student(ean), Student(eva), Student(noa), \\
& Student(sam), Student(sue), Student(zoe), \\
& headOf(ann, cs), memberOf(eve, cs), headOf(mae, ee), \\
& teaches(ann, c1), teaches(eve, c2), teaches(eve, c3), \\
& teaches(mae, c4), teaches(mae, c5), \\
& takes(ani, c1), takes(ean, c1), takes(ean, c2), takes(eva, c3), \\
& takes(noa, c3), takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
\}. &
\end{aligned}$$

Figure 2.6 Individual relationships for Example 2.16



In $\mathcal{T}_{Ex2.16}$, we define, for instance, the concept name $Chair$ as someone who is the head of a $Department$. In a similar style, we define the concept names $Student$ and $GraduateStudent$. We introduce a $GraduateCourse$ and an $UndergraduateCourse$, and enforce that both concept descriptions are disjoint. In addition, we define domain and range restrictions on roles descriptions used in $\mathcal{O}_{Ex2.16}$.

We define two role subsumptions in $\mathcal{R}_{Ex2.16}$. The role description $headOf$ is subsumed by role description $memberOf$. Furthermore, we state that the role description $teaches$ is the inverse role of the role description $isTaughtBy$.

The relationships between individuals in $\mathcal{A}_{Ex2.16}$ are depicted in Figure 2.6. Please note that only role assertions are shown in the graph, since we only intend to visualize the relationship between the individuals to avoid clutter.

It is easy to see that individual ann is an instance of concept description $Chair$ with respect to the ontology $\mathcal{O}_{Ex2.16}$. In order to prove the entailment of the concept assertion $Chair(ann)$, not the whole ABox is necessary. The two ABox assertions $headOf(ann, cs)$ and $Department$ (cs) already suffice to derive the fact that ann is a $Chair$. This small example already suggests that modularization techniques can be valuable in reasoning over ontologies. We define different kinds of modularization techniques in Chapter 3.

Chapter 3: Modularization

Reasoning over description logic ontologies, such as the task of instance retrieval, is difficult. The worst-case time complexity even for solving the basic decision problem instance checking is known to be (double-)exponential for semi-expressive description logics. Furthermore, the sheer amount of data, supported by todays advanced storage technologies, makes efficiency in information retrieval increasingly difficult.

We think that modularization techniques can be used in order to release the main memory dependency from reasoning systems and speed up query answering. Recent advances in distributed and parallel computing, such as multicore-systems and cloud computing, give further support, since it might be possible to distribute modules over multiple cores or computers. We focus on the modularization of ABoxes here, since the size of the assertional part often exceeds the size of the terminological part by orders of magnitude, especially in database-motivated scenarios.

During the remaining part of this thesis we make two assumptions with respect to the input ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$:

- We assume that $\mathcal{O}$ is initially consistent. Despite recent research trends on reasoning over inconsistent (web-)ontologies, e.g. see [HVHT05] and [HH08], we focus on standard decision problems.

- We assume that all concept assertions in $\mathcal{A}$ (and in instance checking/retrieval queries) only contain atomic concept descriptions. This restriction is without loss of generality, since every non-atomic concept description can be given a name in the terminology.

In this chapter, we introduce techniques to break down an ABox into smaller chunks (modules), such that decision (and computational) problems can be solved by considering the smaller parts only. In Section 3.1, we formally define ABox modularizations and additional operations on tableau runs. In Section 3.2, we present an initial ABox partitioning algorithm. While the initial partitioning technique is quite naive, since it is basically inspired by graph components, the technique builds the basis of further modularization techniques. We extend the naive partitioning for the description logic $\mathcal{ALC}$ in Section 3.3, by taking into account terminological information. This kind of technique usually offers a more fine grained partitioning/modularization. We further extend the technique to the semi-expressive description logic $\mathcal{SHI}$. The chapter is concluded with Section 3.4.

3.1 Modularization Preliminaries

3.1.1 ABox Modularization

We define the (very general) notion of an ABox modularization in Definition 3.1. While our criterion for ABox modularization seems quite general, we would like to keep the definition of ABox modularization as open as possible. For instance, we will define modularization techniques, such that the modules are not necessarily subsets of the original ABox. The intuition for these kinds of modules will become clear below. Please note that whenever we use the term *modularization* we usually refer to the result of the modularization process.

Definition 3.1 (ABox Modularization)**:**
An *ABox modularization* M is defined as a set of ABoxes $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$. Each $\mathcal{A}_i$ is called an *ABox module*.

Definition 3.2 (ABox Modularization Entailment)**:**
Given a TBox $\mathcal{T}$, a RBox $\mathcal{R}$, and an *ABox modularization* M, we say that M *entails a concept assertion* $C(a)$, denoted $\langle \mathcal{T}, \mathcal{R}, M\rangle \models C(a)$, if $\exists \mathcal{A}_i \in M.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i\rangle \models C(a)$. We say that M *entails a role assertion* $R(a_1, a_2)$, denoted $\langle \mathcal{T}, \mathcal{R}, M\rangle \models R(a_1, a_2)$, if $\exists \mathcal{A}_i \in M.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i\rangle \models R(a_1, a_2)$.

Given Definition 3.2, we can define soundness and completeness of ABox modularizations. Informally speaking, we have chosen to base soundness and completeness of modularizations on entailment of atomic concept descriptions for all named individuals. This assumption makes the definition and implementation of our techniques easier. However, please note that the restriction to atomic query concepts is without loss of generality, since we can assign fresh concept names to non-atomic query concepts in the TBox and execute a query for the (atomic) concept names.

Informally speaking, again, we would like to obtain modularizations which preserve entailment of atomic concept assertions for all named individuals in the input ontology.

Definition 3.3 (Soundness and Completeness of ABox Modularizations)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ and an ABox modularization $M = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$, we say that M is *sound for instance retrieval in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$ and all individuals $a \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, M\rangle \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models C(a)$. The ABox modularization M is *complete for instance retrieval in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$ and all individuals $a \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, M\rangle \models C(a)$.

We say that M is *sound for relation retrieval in ontology* $\mathcal{O}$ if for all role descriptions $R \in \mathbf{Rol}$ and all pairs of individuals $a_1, a_2 \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, M\rangle \models R(a_1, a_2) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a_1, a_1)$. The ABox modularization M is *complete for relation retrieval in ontology* $\mathcal{O}$ if for all atomic role descriptions $R \in \mathbf{Rol}$ and all pairs of individuals $a_1, a_2 \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \models R(a_1, a_2) \implies \langle \mathcal{T}, \mathcal{R}, M\rangle \models R(a_1, a_2)$.

We say that M is *sound for reasoning in ontology* $\mathcal{O}$ if M is sound for instance and relation retrieval in $\mathcal{O}$. We say that M is *complete for reasoning in ontology* $\mathcal{O}$ if M is complete for instance and relation retrieval in $\mathcal{O}$.

We present examples in Example 3.2 and Example 3.3. First, we introduce an example ontology in Example 3.1.

Example 3.1 (Example Ontology for ABox Modularization)**:**
The ontology $\mathcal{O}_{Ex3.1} = \langle \mathcal{T}_{Ex3.1}, \mathcal{R}_{Ex3.1}, \mathcal{A}_{Ex3.1} \rangle$ is defined as follows

$$\begin{aligned}
\mathcal{T}_{Ex3.1} &= \{Chair \equiv \exists headOf.Department\} \\
\mathcal{R}_{Ex3.1} &= \{headOf \sqsubseteq memberOf\} \\
\mathcal{A}_{Ex3.1} &= \{ \\
&\quad Department(ee), Professor(mae), \\
&\quad UndergraduateCourse(c4), UndergraduateCourse(c5), \\
&\quad Student(sam), Student(sue), Student(zoe), \\
&\quad headOf(mae, ee), \\
&\quad teaches(mae, c4), teaches(mae, c5), \\
&\quad takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
&\quad \}.
\end{aligned}$$

Example 3.2 (First Example for an ABox Modularization)**:**
One possible ABox modularization for ontology $\mathcal{O}_{Ex3.1}$ is $M_{Ex3.2} = \{\mathcal{A}_{Ex3.2,1}, \mathcal{A}_{Ex3.2,2}\}$, such that

$$\begin{aligned}
\mathcal{A}_{Ex3.2,1} &= \{ \\
&\quad Department(ee), \\
&\quad headOf(mae, ee), \\
&\quad Professor(mae) \\
&\quad \} \\
\mathcal{A}_{Ex3.2,2} &= \{ \\
&\quad UndergraduateCourse(c4), UndergraduateCourse(c5), \\
&\quad Student(sam), Student(sue), Student(zoe), \\
&\quad teaches(mae, c4), teaches(mae, c5), \\
&\quad takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
&\quad \}.
\end{aligned}$$

It is easy to see that with respect to the original ontology $\mathcal{O}_{Ex3.1}$, we have that *mae* is an instance of the concept description $Chair$, since she has a $headOf$-relationship to a *Department* called *ee*. The ABox modularization in Example 3.2 entails that individual *mae* is an instance of the concept description $Chair$, since all necessary axioms are being kept in one ABox module. Moreover, it can be shown that the modularization in Example 3.2 is sound and complete for reasoning over ontology $\mathcal{O}_{Ex3.1}$.

Another example modularization is given in Example 3.3.

Example 3.3 (Second Example for an ABox Modularization)**:**
Another possible ABox modularization for Ontology $\mathcal{O}_{Ex3.1}$ is $M_{Ex3.3} = \{\mathcal{A}_{Ex3.3,1}, \mathcal{A}_{Ex3.3,2}\}$, such that

$$\begin{aligned}
\mathcal{A}_{Ex3.3,1} = \{ & \\
& Department(ee), \\
& UndergraduateCourse(c4), UndergraduateCourse(c5) \\
& \} \\
\mathcal{A}_{Ex3.3,2} = \{ & \\
& Professor(mae), \\
& Student(sam), Student(sue), Student(zoe), \\
& headOf(mae, ee), \\
& teaches(mae, c4), teaches(mae, c5), \\
& takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
& \}.
\end{aligned}$$

The ABox modularization in Example 3.3 is chosen quite arbitrarily and it can be seen that neither of the two modules entails that *mae* is an instance of the concept description $Chair$. This happens because the necessary information for entailment was split up into different ABoxes. In [BS03], this problem is solved by so called *bridge rules*, which communicate useful temporary reasoning results from one module to another module. However, we would like to keep relevant information together, in order to avoid the communication overhead.

The ABox modularizations from Example 3.2 and 3.3 show that the choice of ABox modularization is critical for completeness during reasoning. Before we discuss in detail how to obtain sound and complete ABox modularizations (one might even have to add new assertions), we need to define additional mathematical notions. Our proofs for modularization techniques are based on the use and modification of tableau runs. In detail, the basic technique used in our proofs is tableau run composition.

3.1.2 Tableau Run Compositions

We introduce a notion of tableau run composition next. The idea is that under a certain condition (individual disjointness), satisfying tableau runs from different ontologies can be combined to a satisfying tableau run over the union of these ontologies. First, we formally define tableau run compositions in Definition 3.4.

Definition 3.4 (Tableau Run Composition of two Tableau Runs)**:**
Given a tableau run $RUN_1 = \langle \mathbf{N}_1, root_1, children_1, \phi_1^{abox}, \phi_1^{ruleapp} \rangle$ and a tableau run

$RUN_2 = \langle \mathbf{N}_2, root_2, children_2, \phi_2^{abox}, \phi_2^{ruleapp} \rangle$, the *composition* of RUN_1 and RUN_2, denoted $RUN_1 \circ RUN_2$, is defined as

$$RUN_1 \circ RUN_2 = \langle \mathbf{N}_c, root_c, children_c, \phi_c^{abox}, \phi_c^{ruleapp} \rangle,$$

where

$$\begin{aligned}
\mathbf{N}_c &= \mathbf{N}_1 \cup \mathbf{N}_2 \setminus \{leaf\} \\
root_c &= root_1 \\
children_c &= children_1 \cup children_2 \setminus \{leafp \to leaf\} \cup \{leafp \to root_2\} \\
\phi_c^{abox}(n) &= \begin{cases} \phi_1^{abox}(n) \cup \phi_2^{abox}(root_2) & \text{if } n \in (\mathbf{N}_1 \setminus \{leaf\}), \\ \phi_1^{abox}(leaf) \cup \phi_2^{abox}(n) & \text{if } n \in \mathbf{N}_2, \\ n.d. & \text{otherwise.} \end{cases} \\
\phi_c^{ruleapp} &= \phi_1^{ruleapp} \cup \phi_2^{ruleapp},
\end{aligned}$$

$leaf$ denotes the leaf node of RUN_1, and $leafp$ denotes the parent node of $leaf$. Given n tableau runs, the *composition* is defined stepwise as

$$(((RUN_1 \circ RUN_2) \circ RUN_3) ... \circ RUN_n).$$

Please note that in Definition 3.4, we assume that $\mathbf{N}_1$ and $\mathbf{N}_2$ are disjoint. If both sets are not disjoint, then they can be easily made disjoint by renaming all shared nodes and changing the runs accordingly.

Example 3.4 (Tableau Run Composition)**:**
Given the following two ontologies (for simplification we have chosen an empty TBox and an empty RBox),

$$\begin{aligned}
\mathcal{O}_{Ex3.4,1} &= \langle \emptyset, \emptyset, \{\exists takes.Course(zoe)\} \rangle \\
\mathcal{O}_{Ex3.4,2} &= \langle \emptyset, \emptyset, \{\exists takes.Course(noa)\} \rangle,
\end{aligned}$$

a tableau run RUN_1 for $\mathcal{O}_{Ex3.4,1}$, a tableau run RUN_2 for $\mathcal{O}_{Ex3.4,2}$, and the composition $RUN_1 \circ RUN_2$ is depicted in Figure 3.1, where

$$\begin{aligned}
\mathcal{A}_{c1} &= \{\exists takes.Course(zoe), \exists takes.Course(noa)\} \\
\mathcal{A}_{c2} &= \mathcal{A}_{c1} \cup \{takes(zoe, ind_1), Course(ind_1)\} \\
\mathcal{A}_{c3} &= \mathcal{A}_{c2} \cup \{takes(noa, ind_5), Course(ind_5)\}.
\end{aligned}$$

We would like to show that, under some conditions, the composition of satisfying tableau runs can be used to construct a satisfying tableau run for the composed ontology. For instance, given two consistent ontologies $\mathcal{O}_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \rangle$ and $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$, we would like to deduce that $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \rangle$ is consistent by only considering tableau runs.

Figure 3.1 Example for tableau run composition

RUN_1:

$$n_1\langle\{\exists takes.Course(zoe)\}, tabrapp^{\{x\to zoe, x_1\to ind_1\},\exists takes.Course}_{\exists,\emptyset,\emptyset}\rangle$$

$$\Big|$$

$$n_2\langle\{\exists takes.Course(zoe), takes(zoe, ind_1), Course(ind_1)\}, n.d.\rangle$$

RUN_2:

$$n_3\langle\{\exists takes.Course(noa)\}, tabrapp^{\{x\to noa, x_1\to ind_5\},\exists takes.Course}_{\exists,\emptyset,\emptyset}\rangle$$

$$\Big|$$

$$n_4\langle\{\exists takes.Course(noa), takes(noa, ind_5), Course(ind_5)\}, n.d.\rangle$$

$RUN_1 \circ RUN_2$:

$$n_1\langle\mathcal{A}_{c1}, tabrapp^{\{x\to zoe, x_1\to ind_1\},\exists takes.Course}_{\exists,\emptyset,\emptyset}\rangle$$

$$\Big|$$

$$n_3\langle\mathcal{A}_{c2}, tabrapp^{\{x\to noa, x_1\to ind_5\},\exists takes.Course}_{\exists,\emptyset,\emptyset}\rangle$$

$$\Big|$$

$$n_4\langle\mathcal{A}_{c3}, n.d.\rangle$$

However, not in every case two satisfying tableau runs can be combined (composed) to a satisfying tableau run for $\langle\mathcal{T},\mathcal{R},\mathcal{A}_1\cup\mathcal{A}_2\rangle$. A problem occurs if both tableau runs (for ontology $\mathcal{O}_1$ and ontology $\mathcal{O}_2$) contain information about the same individual. This can lead to inconsistencies during the tableau run composition.

For instance, assume that in RUN_1 an individual a is labeled with concept description C, and in RUN_2 the same individual a is labeled with concept description $\neg C$. While both tableau runs maybe yield an open and complete ABox, the composition merges the information about individual a and that will lead to a clash (closed ABox).

Thus, in order to define and prove properties of tableau run compositions, we need to formally define properties of individuals and renaming operations on individuals in tableau runs. Please note that the intuition of the following definitions is usually quite clear. However, in order to be precise and use these notions in proofs below, we provide definitions formally step by step.

In Definition 3.5, the set of (existing) individuals in a tableau run is computed from the union of all ABoxes in the tableau run.

Definition 3.5 (Tableau Run Individuals)**:**
Given a tableau run $RUN = \langle\mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp}\rangle$, the set of *tableau run individuals of* RUN, denoted $Ind(RUN)$, is defined as

$$Ind(RUN) = \{a \mid \exists n \in \mathbf{N}.a \in Ind(\phi^{abox}(n))\}.$$

In order to define individual renamings on tableau runs, we introduce individual renamings for ABoxes, variable assignments, and tableau rule applications as well in Definition 3.6.

Definition 3.6 (Individual Renamings)**:**
Given a structure S, such that S is an ABox, a variable assignment, a tableau rule application, or a tableau run, the *individual renaming* on S, denoted $S[a_1 \to a_2]$, is defined as follows:

- If $S = \mathcal{A}$, then $\mathcal{A}[a_1 \to a_2] = \mathcal{A} \setminus \mathcal{A}_{remove} \cup \mathcal{A}_{add}$, where

$$\begin{aligned}\mathcal{A}_{remove} =&\{C(a_1) \mid C(a_1) \in \mathcal{A}\} \cup \{R(a_1, a_3) \mid R(a_1, a_3) \in \mathcal{A}\} \cup \\ &\{R(a_3, a_1) \mid R(a_3, a_1) \in \mathcal{A}\} \\ \mathcal{A}_{add} =&\{C(a_2) \mid C(a_1) \in \mathcal{A}\} \cup \{R(a_2, a_3) \mid R(a_1, a_3) \in \mathcal{A}\} \cup \\ &\{R(a_3, a_2) \mid R(a_3, a_1) \in \mathcal{A}\}.\end{aligned}$$

- If $S = \pi$, then $\pi[a_1 \to a_2](x) =$

$$\begin{cases} a_2 & \text{if } \pi(x) = a_1, \\ \pi(x) & \text{otherwise.} \end{cases}$$

- If $S = tabrapp^{\pi,Y}_{X,\mathcal{T},\mathcal{R}}$, then $tabrapp^{\pi,Y}_{X,\mathcal{T},\mathcal{R}}[a_1 \to a_2] = tabrapp^{\pi[a_1 \to a_2],Y}_{X,\mathcal{T},\mathcal{R}}$.

- If S is a tableau run $RUN = \langle \mathbf{N}_1, root_1, children_1, \phi_1^{abox}, \phi_1^{ruleapp} \rangle$, then $RUN[a_1 \to a_2] = \langle \mathbf{N}_1, root_1, children_1, \phi_2^{abox}, \phi_2^{ruleapp} \rangle$, such that for each $n \in \mathbf{N}_1$:

$$\begin{aligned}\phi_2^{abox}(n) =&(\phi_1^{abox}(n))[a_1 \to a_2] \\ \phi_2^{ruleapp}(n) =&(\phi_1^{ruleapp}(n))[a_1 \to a_2].\end{aligned}$$

An individual renaming $S[x \to a_2]$ on S is called *fresh*

- if $S = \mathcal{A}$ and $a_2 \notin Ind(\mathcal{A})$,
- if $S = \pi$ and there exists no $x \in \mathbf{VAR}$, such that $\pi(x) = a_2$,
- if $S = tabrapp^{\pi,Y}_{X,\mathcal{T},\mathcal{R}}$ and a_2 is a fresh individual renaming for π,
- if $S = RUN$ and a_2 does not occur in $Ind(RUN)$.

The definition of individual renamings is extended to sets of structures by applying the renaming to each structure in the set.

We extend single individual renamings to consecutive individual renamings.

Definition 3.7 (Consecutive Individual Renamings)**:**
Given a structure S, such that S is either an ABox, a variable assignment, a tableau rule application, or a tableau run, the *consecutive individual renaming*, denoted with $S[a_1, ..., a_n \to b_1, ..., b_n]$, is defined as $(...(S[a_1 \to b_1])...)[a_n \to b_n]$.

In Example 3.5 it can be seen that an ABox individual renaming just replaces all occurrences of an individual, by another individual.

Example 3.5 (ABox Individual Renaming)**:**
Given the ABox

$$\mathcal{A} = \{Student(ani), takes(ani, c1), UndergraduateCourse(c1)\},$$

we have

$$\mathcal{A}[c1 \to c6] = \{Student(ani), takes(ani, c6), UndergraduateCourse(c6)\}.$$

In Example 3.6, the intuition of a tableau rule application renaming is shown. Once we rename an individual in an ABox, for instance $c1$ to $c6$, we also need to consider this individual renaming on tableau rule applications to individual $c1$. All tableau rule applications on $c1$ should be applied to $c6$ after the renaming.

Example 3.6 (Tableau Rule Application Individual Renaming)**:**
Given the tableau rule application

$$tabrapp^{\{x \to c1\}, \neg UndergraduateCourse \sqcup Course}_{\sqcup, \mathcal{T}, \mathcal{R}},$$

which applies the $\sqcup$-tableau rule to individual $c1$ and $\neg UndergraduateCourse \sqcup Course$, we have that

$$tabrapp^{\{x \to c1\}, \neg UndergraduateCourse \sqcup Course}_{\sqcup, \mathcal{T}, \mathcal{R}}[c1 \to c6] = tabrapp^{\{x \to c6\}, \neg UndergraduateCourse \sqcup Course}_{\sqcup, \mathcal{T}, \mathcal{R}}$$

applies the $\sqcup$-tableau rule to individual $c6$ and $\neg UndergraduateCourse \sqcup Course$.

Example 3.7 (Tableau Run Individual Renaming)**:**
Given a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$, such that

$$\begin{aligned}
\mathbf{N} =& \{n_1, n_2, n_3\} \\
root_1 =& n_1 \\
children =& \{n_1 \to n_2, n_2 \to n_3\} \\
\phi^{abox} =& \{n_1 \to \{Student(ani)\}, \\
& n_2 \to \{Student(ani), \neg Student \sqcup Person(ani)\}, \\
& n_3 \to \{Student(ani), \neg Student \sqcup Person(ani), \neg Student(ani)\} \\
& \} \\
\phi^{ruleapp} =& \{n_1 \to tabrapp^{\{x \to ani\}, Student \sqsubseteq Person}_{TBox, \mathcal{T}, \mathcal{R}}, \\
& n_2 \to tabrapp^{\{x \to ani\}, \neg Student \sqcup Person}_{\sqcup, \mathcal{T}, \mathcal{R}} \\
& \},
\end{aligned}$$

we have that $RUN[ani \rightarrow luis] = \langle \mathbf{N}, root, children, \phi_2^{abox}, \phi_2^{ruleapp} \rangle$, such that

$$\begin{aligned}
\phi_2^{abox} =& \{n_1 \rightarrow \{Student(luis)\}, \\
& n_2 \rightarrow \{Student(luis), \neg Student \sqcup Person(luis)\}, \\
& n_3 \rightarrow \{Student(luis), \neg Student \sqcup Person(luis), \neg Student(luis)\} \\
& \} \\
\phi_2^{ruleapp} =& \{n_1 \rightarrow tabrapp_{TBox,\mathcal{T},\mathcal{R}}^{\{x \rightarrow luis\}, Student \sqsubseteq Person}, \\
& n_2 \rightarrow tabrapp_{\sqcup,\mathcal{T},\mathcal{R}}^{\{x \rightarrow luis\}, \neg Student \sqcup Person} \\
& \}.
\end{aligned}$$

We formally define the notion of individual-disjoint tableau runs in Definition 3.8.

Definition 3.8 (Individual-disjoint Tableau Runs)**:**
A set of n tableau runs $\{RUN_1, ..., RUN_n\}$ is *individual-disjoint* if

$$Ind(RUN_1) \cap Ind(RUN_2) \cap ... \cap Ind(RUN_n) = \emptyset.$$

In Figure 3.2, RUN_1 is individual-disjoint with RUN_2, but not with RUN_3. Please note the introduction of the anonymous individual ind_1 in RUN_1 and RUN_3.

Figure 3.2 Example tableau runs for individual disjointness

$$\begin{aligned}
RUN_1\text{:} \quad & n_1\langle\{\exists takes.Course(zoe)\}, tabrapp_{\exists,\emptyset,\emptyset}^{\{x \rightarrow zoe, x_1 \rightarrow ind_1\}, \exists takes.Course}\rangle \\
& \quad | \\
& n_2\langle\{\exists takes.Course(zoe), takes(zoe, ind_1), Course(ind_1)\}, n.d.\rangle \\
RUN_2\text{:} \quad & n_1\langle\{\exists takes.Course(ean)\}, tabrapp_{\exists,\emptyset,\emptyset}^{\{x \rightarrow ean, x_1 \rightarrow ind_5\}, \exists takes.Course}\rangle \\
& \quad | \\
& n_2\langle\{\exists takes.Course(ean), takes(ean, ind_5), Course(ind_5)\}, n.d.\rangle \\
RUN_3\text{:} \quad & n_1\langle\{\exists takes.Course(noa)\}, tabrapp_{\exists,\emptyset,\emptyset}^{\{x \rightarrow noa, x_1 \rightarrow ind_1\}, \exists takes.Course}\rangle \\
& \quad | \\
& n_2\langle\{\exists takes.Course(noa), takes(noa, ind_1), Course(ind_1)\}, n.d.\rangle
\end{aligned}$$

In the following, we show that tableau runs over individual-disjoint $\mathcal{SHI}$-ontologies can always be made individual-disjoint. The idea of the proof of Proposition 3.1 is to rename all individuals generated by the tableau run/rules, until the tableau runs are disjoint.

Proposition 3.1 (Tableau Runs of Individual-disjoint $\mathcal{SHI}$-ontologies can be made Individual-disjoint)**:**
Given n ontologies $\mathcal{O}_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \rangle$, $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$, ..., $\mathcal{O}_n = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_n \rangle$, such that $Ind(\mathcal{A}_i) \cap Ind(\mathcal{A}_j) = \emptyset$, for all $1 \leq i < j \leq n$, and n tableau runs (one for each ontology) RUN_1, RUN_2, ..., RUN_n, the tableau runs can be made individual-disjoint.

Proof of Proposition 3.1. Whenever an individual occurs in two tableau runs (note that the individual must have been introduced by tableau rule applications, since the source ontologies are individual-disjoint), we can rename the shared individual in one of the tableau runs to a new individual not used in any of the n tableau runs. Validity of this renaming step is shown as follows:

Given RUN is a tableau run for ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and $RUN[a_1 \rightarrow a_2]$ is a fresh individual renaming on RUN, it is easy to see that:

- $RUN[a_1 \rightarrow a_2]$ is a tableau run for $\mathcal{O}$: First, note that $\mathcal{A}[a_1 \rightarrow a_2] = \mathcal{A}$, i.e. the ABox of the root node of $RUN[a_1 \rightarrow a_2]$ is labeled with $\mathcal{A}$. It can be seen that all tableau rules remain applicable in $RUN[a_1 \rightarrow a_2]$. Furthermore, by induction on the tableau rules, it can be seen that the leaf ABox of $RUN[a_1 \rightarrow a_2]$ is complete: if a tableau rule is applicable in the leaf ABox of $RUN[a_1 \rightarrow a_2]$, then it would have to be already applicable in the leaf ABox of RUN.
- RUN contains a clash if and only if $RUN[a_1 \rightarrow a_2]$ contains a clash: a fresh individual renaming does not change the concept set of any individual in the ABoxes of the tableau run (only the name). Thus, the fresh individual renaming does not change the result (clash or no clash) of the tableau run.

Repeated application of fresh individual renamings yield a set of n individual-disjoint tableau runs. □

In Proposition 3.2 we show that the composition of two individual-disjoint tableau runs yields a tableau run for the union of the two source ontologies.

Proposition 3.2 (Composition of two Individual-disjoint Tableau Runs is a Tableau Run)**:**
Given two ontologies $\mathcal{O}_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \rangle$ and $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$, and two individual-disjoint tableau runs, RUN_1 for $\mathcal{O}_1$ and RUN_2 for $\mathcal{O}_2$, $RUN_1 \circ RUN_2$ is a tableau run for $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \rangle$.

Proof of Proposition 3.2. By Definition 3.4, the ABox of the root node of $RUN_1 \circ RUN_2$ is $\mathcal{A}_1 \cup \mathcal{A}_2$. It can be seen by induction on the tableau rules that tableau rules remain applicable for each node in $RUN_1 \circ RUN_2$. The fact that the leaf node of $RUN_1 \circ RUN_2$ is labeled with a complete ABox can be shown by contradiction: if a tableau rule is applicable to $\phi_1^{abox}(n_1) \cup \phi_2^{abox}(n_2)$ then the tableau rule must be applicable already either in ϕ_1^{abox} (n_1) or ϕ_2^{abox} (n_2), since ϕ_1^{abox} (n_1) and ϕ_2^{abox} (n_2) are individual-disjoint. □

We show in Lemma 3.1 that the composition of n individual-disjoint tableau runs yields a tableau run for the union of the source ontologies.

Lemma 3.1 (Composition of n Individual-disjoint Satisfying Tableau Runs)**:**
Given n ontologies $\mathcal{O}_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \rangle$, $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$, ..., $\mathcal{O}_n = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_n \rangle$, and n individual-disjoint tableau runs (one for each ontology) RUN_1, RUN_2, ..., RUN_n, it is true that: The tableau run $RUN = RUN_1 \circ RUN_2 \circ ... \circ RUN_n$ is a satisfying tableau run for $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \cup \mathcal{A}_n \rangle$ if and only if each tableau run RUN_i (such that $1 \leq i \leq n$) is satisfying.

Proof of Lemma 3.1. By repeated application of Proposition 3.2, RUN is a tableau run for $\mathcal{O}$. It is easy to see that RUN contains a clash if and only if there exists a RUN_i containing a clash: since the individuals of all tableau runs are disjoint, tableau run composition does not change the concept set of any individual in the ABoxes of the tableau run. Thus, the composition does not change the result (clash or no clash) of the tableau run. □

To summarize, the important condition used in Lemma 3.1 is individual disjointness. If the individuals of two tableau runs overlap, then the ABox merging can create direct clashes. Moreover, even if the individuals of the source ABoxes are disjoint, one still has to ensure disjointness of all fresh individuals (generated by tableau rule applications, e.g. the ∃-tableau rule).

In order to prove properties about tableau run compositions below, we define a general notion of an assertion extension to a tableau run. The motivation is that under some conditions, these assertion extensions of tableau runs are tableau runs again.

Definition 3.9 (Tableau Run Assertion Extension)**:**
Given

- an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$,
- a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$ for $\mathcal{O}$, and
- an ABox $\mathcal{A}_{ext}$,

the *tableau run assertion extension of RUN with* $\mathcal{A}_{ext}$, denoted $RUN^{+\mathcal{A}_{ext}}$, is defined as $RUN^{+\mathcal{A}_{ext}} = \langle \mathbf{N}, root, children, \phi_2^{abox}, \phi^{ruleapp} \rangle$, such that for all $n \in \mathbf{N}$ we have $\phi_2^{abox}(n) = \phi^{abox}(n) \cup \mathcal{A}_{ext}$.

In the remaining part of this chapter, we focus on how to find „interesting“ ABox modularizations, i.e. ABox modularizations which guarantee soundness and completeness of modularization-based instance tests for different classes of description logics.

3.2 Component-based Modularization

With *component-based modularization* we refer to modularization techniques which only consider the assertional part of an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ in order to decide how to break up an ABox into an ABox modularization. For this purpose, we look at ABoxes as graphs. The intuition is as follows: each individual in the ABox is mapped to a node in the graph. Node labels are concept assertions from the ABox and the edges of the graph are derived from the role assertions. We introduce a formal notion in order to define algorithms and proofs. Given an ABox $\mathcal{A}$, we define the corresponding ABox-graph in Definition 3.10.

Definition 3.10:
Given an ABox $\mathcal{A}$, an *ABox-graph* $\mathbb{G}^{\mathcal{A}} = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$ for $\mathcal{A}$ is a directed labeled graph such that

- $\mathbf{N} = Ind(\mathcal{A})$,
- the codomain of ϕ is $\wp(\mathbf{Con})$,
- the codomain of σ is $\wp(\mathbf{Rol})$,
- for all $n \in \mathbf{N}$, we have $C \in \phi(n)$ if and only if $C(n) \in \mathcal{A}$, and
- for all pairs of nodes $(n_1, n_2) \in \mathbf{N} \times \mathbf{N}$, we have $R \in \sigma(n_1, n_2)$ if and only if $R(n_1, n_2) \in \mathcal{A}$.

Please note that the construction of an ABox-graph for a given ABox $\mathcal{A}$ is deterministic and invertible. This means that given a ABox-graph $\mathbb{G}^{\mathcal{A}}$, we can reconstruct the corresponding ABox $\mathcal{A}$. Given this relationship, we often change between the usual ABox-view and the ABox-graph-view whenever it is convenient.

Since the ABox $\mathcal{A}$ of an ontology $\mathcal{O}$ can be seen as a graph, it seems natural to apply standard connectedness-based graph partitioning techniques to determine ABox modules: if two individuals a_1 and a_2 are connected in the ABox-graph, then these two individuals end up in the same ABox module.

Definition 3.11 (Graph Component-based ABox Modularization for an ABox)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a *graph component-based ABox modularization for* $\mathcal{A}$, denoted $MC^{\mathcal{A}}$, is an ABox Modularization $\{\mathcal{A}_1, ..., \mathcal{A}_n\}$ for $\mathcal{A}$, such that $\mathcal{A}_i \in MC^{\mathcal{A}} \iff \mathbb{G}^{\mathcal{A}_i}$ is a component in $\mathbb{G}^{\mathcal{A}}$.

An optimal algorithm for obtaining graph components is shown in [HT73]. The components of a graph can be obtained in linear time in the number of edges. In the following, we often refer to the term „graph component-based“ with the term *component-based*.

Please note that the graph component-based ABox modularization for an ABox $\mathcal{A}$ is unique. An example of a component-based ABox modularization is shown in Example 3.8.

Example 3.8 (Example for ABox Modularization by Graph Components)**:**
Given the ontology $\mathcal{O}_{Ex3.8} = \langle \mathcal{T}_{Ex3.8}, \mathcal{R}_{Ex3.8}, \mathcal{A}_{Ex3.8} \rangle$, such that

$$\begin{aligned}
\mathcal{T}_{Ex3.8} &= \{Chair \equiv \exists headOf.Department\} \\
\mathcal{R}_{Ex3.8} &= \{headOf \sqsubseteq memberOf\} \\
\mathcal{A}_{Ex3.8} &= \{ \\
&\quad Department(cs), Professor(ann), \\
&\quad headOf(ann, cs), \\
&\quad Department(ee), Professor(mae), \\
&\quad headOf(mae, ee) \\
&\quad \},
\end{aligned}$$

the graph component-based ABox modularization is $M_{Ex3.8} = \{\mathcal{A}_{Ex3.8,1}, \mathcal{A}_{Ex3.8,2}\}$, such that

$$\begin{aligned}
\mathcal{A}_{Ex3.8,1} &= \{Department(cs), Professor(ann), headOf(ann, cs)\} \\
\mathcal{A}_{Ex3.8,2} &= \{Department(ee), Professor(mae), headOf(mae, ee)\}.
\end{aligned}$$

In the following, we show soundness and completeness of instance retrieval over graph component-based ABox modularizations. We start with the proof of soundness, which is quite straightforward. First, we formally prove that the union of all ABox modules is equivalent to the original ABox.

Proposition 3.3 (Graph Components Recover ABox)**:**
For each component-based ABox modularization $M = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ we have $\mathcal{A} = (\mathcal{A}_1 \cup ... \cup \mathcal{A}_n)$.

Proof of Proposition 3.3. Easy to see by the formal definition of graph components. □

Lemma 3.2 ($\mathcal{SHI}$-Instance Retrieval over Component-based ABox Modularizations is Sound)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, the component-based ABox modularization $MC^{\mathcal{A}} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ for $\mathcal{A}$ is sound for instance retrieval in $\mathcal{O}$.

Proof of Lemma 3.2. We have to show that for all atomic concept descriptions $C \in$ **AtCon** and all individuals $a \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, MC^{\mathcal{A}} \rangle \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. Assume that $\langle \mathcal{T}, \mathcal{R}, MC^{\mathcal{A}} \rangle \models C(a)$. Thus, $\exists \mathcal{A}_i \in MC^{\mathcal{A}}. \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a)$. From Proposition 3.3 it is clear that $\mathcal{A}_i \subseteq \mathcal{A}$, and hence $\mathcal{A}_i \cup \{\neg C(a)\} \subseteq \mathcal{A} \cup \{\neg C(a)\}$. Furthermore, from $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a)$, it can be seen that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \cup \{\neg C(a)\} \rangle$ is inconsistent. Thus, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a)\} \rangle$ must be inconsistent as well, and thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. □

Next, we show a proof for completeness with respect to $\mathcal{SHI}$ and give a negative result for the description logic $\mathcal{SHOQ}$. Before we show the actual proof, we prove a property about the union of individual-disjoint ontologies.

Proposition 3.4 (Consistency Preservation of Unions of Individual-disjoint Ontologies)**:** Given n individual-disjoint ontologies $\mathcal{O}_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \rangle$, $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$, ..., and $\mathcal{O}_n = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_n \rangle$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \cup \mathcal{A}_n \rangle$ is consistent if $\mathcal{O}_1$, $\mathcal{O}_2$, ... and $\mathcal{O}_n$ are consistent respectively.

Proof of Proposition 3.4. If $\mathcal{O}_1$, $\mathcal{O}_2$,..., $\mathcal{O}_n$ are consistent, then there exist n tableau runs RUN_1 for $\mathcal{O}_1$, RUN_2 for $\mathcal{O}_2$,..., RUN_n for $\mathcal{O}_n$, such that all of these tableau runs are satisfying. Since all input ontologies are individual-disjoint to each other, one can compute a set of n individual-disjoint tableau runs RUN_{new1}, ..., RUN_{newn}, by Proposition 3.1. Then by Lemma 3.1, a satisfying tableau run for $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \cup \mathcal{A}_n \rangle$ exists. Thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \cup \mathcal{A}_n \rangle$ is consistent. □

Lemma 3.3 ($\mathcal{SHI}$-Instance Retrieval over Component-based ABox Modularizations is Complete)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a component-based ABox modularization $MC^{\mathcal{A}} = \{\mathcal{A}_1, ..., \mathcal{A}_n\}$ for $\mathcal{A}$, the ABox modularization $MC^{\mathcal{A}}$ is complete for instance retrieval in $\mathcal{O}$.

Proof of Lemma 3.3. We have to show that for all atomic concept descriptions $C \in$ **AtCon** and all individuals $a \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, MC^{\mathcal{A}} \rangle \models C(a)$. By contraposition: We have to show $\langle \mathcal{T}, \mathcal{R}, MC^{\mathcal{A}} \rangle \not\models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a)$. Assume that $\langle \mathcal{T}, \mathcal{R}, MC^{\mathcal{A}} \rangle \not\models C(a)$. Thus, for all $\mathcal{A}_i \in MC^{\mathcal{A}}$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \cup \{\neg C(a)\} \rangle$ is consistent. Let $\mathcal{A}_j$ be the ABox module, such that $a \in Ind(\mathcal{A}_j)$. There exists only one such module, by Definition 3.11. By Proposition 3.4, we can conclude that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \mathcal{A}_{j-1} \cup (\mathcal{A}_j \cup \{\neg C(a)\}) \cup \mathcal{A}_{j+1} \cup ... \cup \mathcal{A}_n \rangle$ is consistent as well. Since, by Proposition 3.3, $\mathcal{A} \cup \{\neg C(a)\} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \mathcal{A}_{j-1} \cup (\mathcal{A}_j \cup \{\neg C(a)\}) \cup \mathcal{A}_{j+1} \cup ... \cup \mathcal{A}_n$, the ontology $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a)\} \rangle$ is consistent, and thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a)$. □

Theorem 3.1 (Instance Retrieval over Component-based ABox Modularizations is Sound and Complete for $\mathcal{SHI}$)**:**
Instance Retrieval over component-based ABox modularizations is sound and complete for $\mathcal{SHI}$-ontologies.

Proof of Theorem 3.1. In Lemma 3.2 we have shown soundness and in Lemma 3.3 we have shown completeness. □

Unfortunately, the completeness result does not hold for ontologies containing nominals, e.g. $\mathcal{SHOQ}$-ontologies. It is not always possible to ensure individual disjointness of tableau runs, because TBox axioms can introduce individuals and these individuals cannot be renamed without changing the semantics and result of the tableau run. While all modules of an ABox modularization are consistent, the complete ontology might be inconsistent. Thus, graph component-based ABox modularization techniques cannot be applied directly to ontologies containing nominals.

The effectiveness of component-based modularization techniques is usually quite low, since in most ontologies each individual is related to many other individuals, either directly or indirectly.

3.3 Intensional-based Modularization

Component-based modularization alone can be too naive for the modularization of real-world ontologies. Usually, most individuals in an ABox are connected by paths of role assertions to many other individuals. Thus, the number of modules obtained by component-based ABox modularizations can be quite small and the average module size is usually quite big. In the following section, we discuss how to compute smaller modules by splitting up role assertions whenever possible. After the split process is finished, we can apply component-based modularization techniques on the result. Please note again that, during the modularization process, we are interested in preserving the entailment of all atomic concept descriptions for each named individual.

The idea is to analyze the terminological part of the ontology (hence called intensional-based modularization) to find out in which ways role assertions are used in the ontology. It is important to note that we only use a purely syntactic analysis of the TBox. Otherwise, for complex ontologies, a more sophisticated analysis could turn out to be too complex. To illustrate the idea of intensional-based modularization in a more detailed way, an example ontology is given in Example 3.9.

Example 3.9 (Example Ontology)**:**
Let $\mathcal{O}_{Ex3.9} = \langle \mathcal{T}_{Ex3.9}, \mathcal{R}_{Ex3.9}, \mathcal{A}_{Ex3.9} \rangle$ as follows:

$$\begin{aligned}\mathcal{T}_{Ex3.9} =& \{\top \sqsubseteq \forall takes.Course\}\\ \mathcal{R}_{Ex3.9} =& \{\}\\ \mathcal{A}_{Ex3.9} =& \{Course(c5), Student(zoe),\\ & \quad takes(zoe, c5), teaches(mae, c5)\}.\end{aligned}$$

Looking closer at the ontology defined in Example 3.9 reveals the following details about the role assertions in $\mathcal{A}_{Ex3.9}$:

- $teaches(mae, c5)$: The role *teaches* is not used (mentioned) anywhere in the TBox or RBox of the ontology $\mathcal{O}_{Ex3.9}$. Thus, no information can be propagated in a tableau algorithm from *mae* to $c5$ and vice versa via *teaches*, and it might be safe to ignore/remove the role assertion to obtain a more fine grained modularization in some cases.
- $takes(zoe, c5)$: Although the role *takes* is mentioned in $\mathcal{T}_{Ex3.9}$, we can see that it is only used to propagate the concept description *Course*. Since individual $c5$ is already known to be an instance of *Course*, because that fact is directly asserted in $\mathcal{A}_{Ex3.9}$, we might further split up this role assertion in some cases.

3.3.1 Technical Preliminaries

In the following, we define necessary criteria for identifying concept descriptions which are propagated over role descriptions in the worst-case during the application of the tableau algorithm. Since we only allow atomic concept assertions in ABoxes, we can focus on the syntactic analysis of the TBox to obtain this set of concept descriptions. First, we define a normal form of general concept inclusions and TBoxes. The normal form makes the syntactical analysis of a terminology easier.

Definition 3.12 (Normal Form of GCIs)**:**
A general concept inclusion axiom is in *normal form* if it has the shape $\top \sqsubseteq C$, such that C is a concept description in negation normal form. A TBox $\mathcal{T}$ is in *normal form* (or *normalized*) if all general concept inclusion axioms in $\mathcal{T}$ are in normal form.

It is easy to see that each GCI can be transformed into an equivalent GCI in negation normal form. Furthermore, it is easy to see that each TBox can be transformed into an equivalent TBox in negation normal form.

In Definition 3.13, we formally define a structure which associates the worst-case set of propagated concept descriptions with each role description. The idea is to extract subconcept descriptions of all ∀-concept descriptions from the closure of the input TBox.

Definition 3.13 (∀-info structure)**:**
A ∀-*info structure for* a TBox $\mathcal{T}$ in normal form is a function $info^{\forall}_{\mathcal{T}} : \mathbf{Rol} \rightarrow \wp(\mathbf{Con})$, such that we have $C \in info^{\forall}_{\mathcal{T}}(R)$ if and only if $\forall R.C \in clos(\mathcal{T})$.

Example 3.10 (Example for a ∀-info structure)**:**
Let

$$\begin{aligned}\mathcal{T}_{Ex3.10} = \{&\top \sqsubseteq \forall takes.Course,\\ &\exists takes.Course \sqsubseteq Student,\\ &\exists memberOf.\top \sqsubseteq Person,\\ &GraduateStudent \sqsubseteq Student,\\ &UndergraduateStudent \sqsubseteq Student\\ \},&\end{aligned}$$

then one TBox in normal form is

$$\begin{aligned}\mathcal{T}_{Ex3.10norm} = \{&\top \sqsubseteq \forall takes.Course,\\ &\top \sqsubseteq \forall takes.\neg Course \sqcup Student,\\ &\top \sqsubseteq \forall memberOf.\bot \sqcup Person,\\ &\top \sqsubseteq \neg GraduateStudent \sqcup Student,\\ &\top \sqsubseteq \neg UndergraduateStudent \sqcup Student\\ \}&\end{aligned}$$

and the $\forall$-info structure for $\mathcal{T}_{Ex3.10norm}$ is:

$$info^{\forall}_{\mathcal{T}_{Ex3.10norm}}(R) = \begin{cases} \{Course, \neg Course\} & \text{if } R = takes, \\ \{\bot\} & \text{if } R = memberOf, \\ \emptyset & \text{otherwise.} \end{cases}$$

The $\forall$-info structure helps us to check which concept descriptions are (in the worst case) propagated over role assertions during the application of tableau rules in tableau proofs. First, we prove a general property about concept descriptions in tableau runs.

Proposition 3.5 (Labeling with Non-Atomic Concept Descriptions in Tableau Runs for $\mathcal{ALCHI}$-Ontologies)**:**
Given

- an $\mathcal{ALCHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$,
- a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$,
- a node $n \in \mathbf{N}$,
- an individual a, and
- a non-atomic concept description C,

$C(a) \in \phi^{abox}(n) \implies C \in clos(\mathcal{T})$.

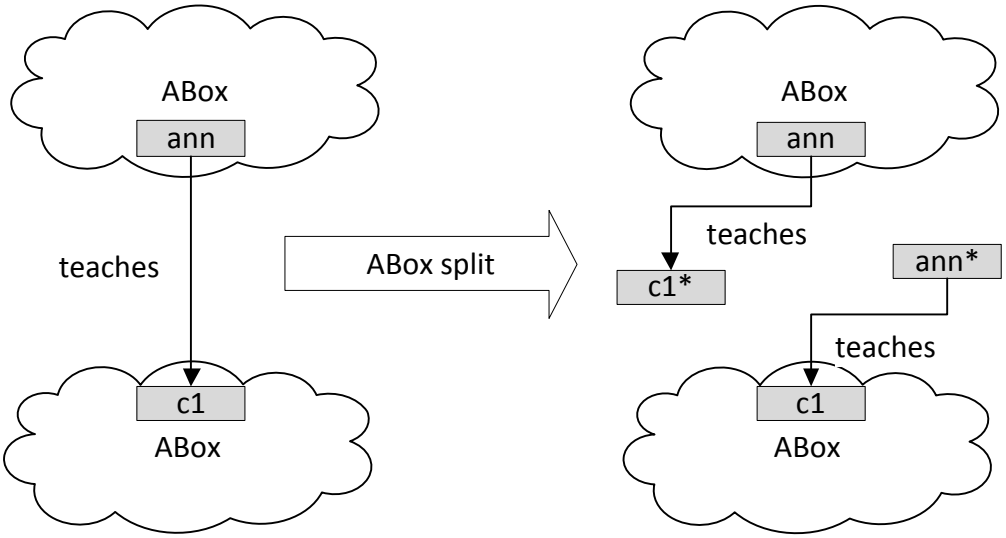
Proof of Proposition 3.5. Initially we only have concept assertions with atomic concept descriptions in $\mathcal{A}$. By induction on the tableau run (the applications of tableau rules), it is easy to see that every newly added concept description is in $clos(\mathcal{T})$. □

Given the above results, we define an operation which splits up role assertions in such a way that we can apply graph component-based modularization techniques over the outcome of the split. Then we show that under some conditions the operation retains soundness and completeness for instance checking and instance retrieval.

Definition 3.14 (ABox Split)**:**
Given

- a role description R,
- two distinct named individuals a and b,
- two distinct anonymous individuals c and d, and,
- an ABox $\mathcal{A}$,

Figure 3.3 Intuition of an ABox split



an *ABox split* is a function $\downarrow_{c,d}^{R(a,b)}$: $\mathbf{SA} \rightarrow \mathbf{SA}$, defined as follows:

- If $R(a,b) \in \mathcal{A}$ and $\{c,d\} \cap Ind(\mathcal{A}) = \emptyset$, then
$$\downarrow_{c,d}^{R(a,b)}(\mathcal{A}) = \mathcal{A} \setminus \{R(a,b)\} \cup \{R(a,d), R(c,b)\} \cup \{C(c) \mid C(a) \in \mathcal{A}\} \cup \{C(d) \mid C(b) \in \mathcal{A}\}$$
- Else
$$\downarrow_{c,d}^{R(a,b)}(\mathcal{A}) = \mathcal{A}.$$

The intuition of Definition 3.14 is depicted in Figure 3.3. The clouds in Figure 3.3 indicate a set of ABox assertions. We split up a role assertion and keep the concept assertions for each fresh individual copy. The reason for keeping the asserted concept descriptions is explained below. If the ABox does not contain the role assertion in question, then the split returns an unchanged ABox.

In Definition 3.15, we define soundness and completeness of ABox splits. While soundness of ABox splits is shown by simply applying Lemma 3.4, the proof of completeness is harder and depends on several criteria.

Definition 3.15 (Sound, Complete and Valid ABox Split)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, we say that

- $\downarrow_{c,d}^{R(a,b)}$ is *sound with respect to* $\mathcal{O}$ if and only if for all individuals $a_1 \in NInd(\mathcal{A})$ and all atomic concept descriptions $C \in \mathbf{AtCon}$:
$$\exists \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a_1) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a_1),$$

- $\downarrow_{c,d}^{R(a,b)}$ is *complete with respect to* $\mathcal{O}$ if and only if for all individuals $a_1 \in NInd(\mathcal{A})$ and all atomic concept descriptions $C \in \mathbf{AtCon}$:

$$\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a_1) \implies \exists \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a_1),$$

- $\downarrow_{c,d}^{R(a,b)}$ is *valid with respect to* $\mathcal{O}$ if $\downarrow_{c,d}^{R(a,b)}$ is sound and complete with respect to $\mathcal{O}$.

Lemma 3.4 (Soundness of ABox Splits)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is sound with respect to $\mathcal{O}$.

Proof of Lemma 3.4. We have to show that for all individuals $a_1 \in NInd(\mathcal{A})$ and all atomic concept descriptions $C \in \mathbf{AtCon}$: $\exists \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a_1) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a_1)$. Assume that $\exists \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a_1)$. Without loss of generality, let $\mathcal{A}_X \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}$ be the ABox module which makes $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_X \cup \{\neg C(a_1)\} \rangle$ inconsistent. Furthermore, let $\mathcal{A}_* = \mathcal{A}_X \cup \{\neg C(a_1)\}$. It is easy to see that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_* \rangle$ is inconsistent and we have to show that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a_1)\} \rangle$ is inconsistent. By contraposition: We show that if $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a_1)\} \rangle$ is consistent, then $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_* \rangle$ is consistent. Assuming that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a_1)\} \rangle$ is consistent, there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a_1)\} \rangle$. It is easy to see that for the interpretation $\mathcal{I}_{new}$, an extension of $\mathcal{I}$ by setting $c^{\mathcal{I}_{new}} = a^{\mathcal{I}}$ and $d^{\mathcal{I}_{new}} = b^{\mathcal{I}}$, $\mathcal{I}_{new} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_* \rangle$ and thus, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_* \rangle$ is consistent. □

The criteria for ensuring completeness of ABox splits are introduced below and proven step-wise for the description logic $\mathcal{ALC}$ and extensions up to $\mathcal{SHI}$. We define the notion of a consistency-preserving ABox split, for which, informally speaking, the split-up role assertion can be added to the outcome of the ABox split without changing consistency. This is formally defined in Definition 3.16.

Definition 3.16 (Consistency-preserving ABox Split)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, we say that $\downarrow_{c,d}^{R(a,b)}$ is a *consistency-preserving ABox split for* $\mathcal{O}$ if for all atomic concept descriptions C and all individuals $e \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)}(\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)}(\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent.

Lemma 3.5 (Completeness for Consistency-preserving ABox Splits)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is complete with respect to $\mathcal{O}$ if $\downarrow_{c,d}^{R(a,b)}$ is a consistency-preserving ABox split for $\mathcal{O}$.

Proof of Lemma 3.5. We have to show that for all named individuals $a_1 \in NInd(\mathcal{A})$ and all atomic concept descriptions $C \in \mathbf{AtCon}$:

$$\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a_1) \implies \exists \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \rangle \models C(a_1).$$

By contraposition: We have to show that $\forall \mathcal{A}_i \in MC^{\downarrow_{c,d}^{R(a,b)}(\mathcal{A})}.\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i\rangle \not\models C(a_1) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \not\models C(a_1)$. Assume that all $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_i \cup \{\neg C(a_1)\}\rangle$ are consistent. Let $\mathcal{A}_j$ be the ABox module, such that $a_1 \in NInd(\mathcal{A}_j)$. There exists only one such module, by Definition 3.11. Let $\mathcal{A}_* = \mathcal{A}_1 \cup \mathcal{A}_2 \cup ... \mathcal{A}_{j-1} \cup (\mathcal{A}_j \cup \{\neg C(a_1)\}) \cup \mathcal{A}_{j+1} \cup ... \cup \mathcal{A}_n$. By Proposition 3.4, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_*\rangle$ is consistent. Since $\downarrow_{c,d}^{R(a,b)}$ is a consistency-preserving ABox split for $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ (precondition), we know that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_* \cup \{R(a,b)\}\rangle$ is consistent, because $\mathcal{A}_* = \downarrow_{c,d}^{R(a,b)}(\mathcal{A}) \cup \{\neg C(a_1)\}$. Since $\mathcal{A} \cup \{\neg C(a_1)\} \subseteq \mathcal{A}_* \cup \{R(a,b)\}$, we can conclude that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a_1)\}\rangle$ is consistent as well, and thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle \not\models C(a_1)$. □

Lemma 3.5 and Definition 3.16 help us to identify complete ABox splits, by finding consistency-preserving ABox splits. We identify classes of these consistency-preserving ABox splits below. Please note that consistency-preserving ABox splits do not affect the blocking of individuals, i.e. adding the role assertions does not change the blocking condition for any individual.

We distinguish the following three scenarios as candidate criteria for consistency-preserving ABox splits $\downarrow_{c,d}^{R(a,b)}$:

1. No concept descriptions are propagated over R.
2. Only the concept description $\bot$ is propagated over R.
3. Only atomic concept descriptions are propagated over R, such that each propagation, informally speaking, either yields redundant information or an obvious clash.

Each scenario is discussed in detail for the description logic $\mathcal{ALC}$ below.

3.3.2 Consistency-preserving ABox Splits for $\mathcal{ALC}$

First, we revisit the definition of neighbors again. The neighborship of individuals is important for the application of the ∀-tableau rule. In Proposition 3.6, we define a criterion for the role neighborship of individuals in $\mathcal{ALC}$-ontologies.

Proposition 3.6 (Role Assertion Neighbor Impact in $\mathcal{ALC}$)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$, a role description $R \in \mathbf{Rol}$, and two individuals $a_1 \in Ind(\mathcal{A})$ and $a_2 \in Ind(\mathcal{A})$, we have a_2 is an R-neighbor of a_1 only if $R(a_1, a_2) \in \mathcal{A}$.

Proof of Proposition 3.6. By Definition 2.31, Definition 2.30, and the absence of inverse roles and role hierarchies in $\mathcal{ALC}$. □

In Proposition 3.7, we show that a ∀-info structure can be used for applicability checks of tableau rules in tableau algorithms for $\mathcal{ALC}$-ontologies.

Proposition 3.7 (Relationship between ∀-Info Structure and Tableau Rule Applications in $\mathcal{ALC}$)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a ∀-info structure $info^{\forall}_{\mathcal{T}}$ for $\mathcal{T}$, and a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$ for $\mathcal{O}$, for each $n \in \mathbf{N}$, $\phi^{ruleapp}(n) = tabrapp^{\pi,\forall R.C_1}_{\forall,\mathcal{T},\mathcal{R}} \implies C_1 \in info^{\forall}_{\mathcal{T}}(R)$.

Proof of Proposition 3.7. If the ∀-tableau rule is applicable then, by Definition 2.37 and Proposition 3.6, we must have $\forall R.C_1(\pi(x)) \in \phi^{abox}(n)$. By Proposition 3.5, we conclude that $\forall R.C_1 \in clos(\mathcal{T})$. By Definition 3.13, we have $C_1 \in info^{\forall}_{\mathcal{T}}(R)$. □

Below, we discuss three cases for consistency-preserving ABox splits. First, in Lemma 3.6, we prove that an ABox split is consistency-preserving, if no concept descriptions can be propagated over the role assertion of the ABox split during the application of a tableau algorithm to the ontology.

Lemma 3.6 (Propagationless ABox Splits)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow^{R(a,b)}_{c,d}$, $\downarrow^{R(a,b)}_{c,d}$ is a consistency-preserving ABox split for $\mathcal{O}$ if $info^{\forall}_{\mathcal{T}}(R) = \emptyset$.

Proof of Lemma 3.6. We have to show that for all atomic concept descriptions C and all individuals $e \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. Assume that $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent. Thus, there exists a satisfying tableau run RUN for $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$. It is easy to see that $RUN^{+\{R(a,b)\}}$ is a tableau run for $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e) \cup \{R(a,b)\}\} \rangle$. For each node in the tableau run, the tableau rules remain applicable. Please note that especially the ∃-tableau rules remain applicable, since the applicability condition requires an anonymous successor individual (a and b are named individuals). In the remaining part, we only discuss the completeness of the leaf ABox of $RUN^{+\{R(a,b)\}}$.

Since the leaf ABox $\mathcal{A}_{leaf}$ of RUN is complete, the only tableau rule which could become applicable due to the ABox extension is the ∀-tableau rule. But since we assume $info^{\forall}_{\mathcal{T}}(R) = \emptyset$, we can conclude, by Proposition 3.7, that the individual a cannot be labeled with a ∀-constraint on role R. Thus, the ∀-tableau rule is not applicable either. In particular, the $TBox$-tableau rule cannot become applicable, since $a \in Ind(\mathcal{A}_{leaf})$ and $b \in Ind(\mathcal{A}_{leaf})$.

The leaf ABox of $RUN^{+\{R(a,b)\}}$ is $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. It is easy to see that this addition yields no immediate clash in $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. Thus, $RUN^{+\{R(a,b)\}}$ is satisfying and $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. □

Next, we discuss consistency-preserving ABox splits with role assertions, such that only direct contradictions are propagated, i.e. given an $\downarrow^{R(a,b)}_{c,d}$, we have $info^{\forall}_{\mathcal{T}}(R) = \{\bot\}$.

Lemma 3.7 (Clash-Propagation ABox Splits)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is a consistency-preserving ABox split for $\mathcal{O}$ if $info_{\mathcal{T}}^{\forall}(R) = \{\bot\}$.

Proof of Lemma 3.7. We have to show that for all atomic concept descriptions C and all individuals $e \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. Assume that $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent. Thus, there exists a satisfying tableau run RUN for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$. It is easy to see that $RUN^{+\{R(a,b)\}}$ is a tableau run for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$. All tableau rules are still applicable on the respective ABox in the tableau run $RUN^{+\{R(a,b)\}}$. Please note that especially the $\exists$-tableau rules remain applicable, since the applicability condition requires an anonymous successor individual (a and b are named individuals). In the remaining part, we only discuss the completeness of the leaf ABox of $RUN^{+\{R(a,b)\}}$.

Since the leaf ABox $\mathcal{A}_{leaf}$ of RUN is complete, the only tableau rule which could become applicable due to the ABox extension is the $\forall$-tableau rule. However, if the $\forall$-tableau rule becomes applicable for $R(a,b)$, then it must have been already applicable in RUN for the role assertion $R(a,d)$. Since RUN is satisfying, d does not contain a direct clash, and thus the $\forall$-tableau rule was not applicable to $R(a,d)$ in RUN and it cannot be applicable to $R(a,b)$ either. Since, by Proposition 3.7, only $\bot$ is propagated over R, the $\forall$-tableau rule is not applicable. The $TBox$-tableau rule cannot become applicable, since we have $a \in Ind(\mathcal{A}_{leaf})$ and $b \in Ind(\mathcal{A}_{leaf})$.

The leaf ABox of $RUN^{+\{R(a,b)\}}$ is $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. It is easy to see that this addition yields no immediate clash in $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. Thus, $RUN^{+\{R(a,b)\}}$ is satisfying and $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. □

In the following, we discuss completeness of ABox splits with role assertions, such that only particular atomic concepts are propagated. These atomic concepts are special in such a way that they will either only propagate redundant information or yield a direct clash during the application of a tableau algorithm. First, we discuss the propagation of redundant information. The terminological knowledge can be used to avoid the worst-case propagation over the role assertion of concern.

Lemma 3.8 (Redundant Propagation ABox Splits)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is a consistency-preserving ABox split for $\mathcal{O}$ if $info_{\mathcal{T}}^{\forall}(R) = \{C_1\}$ and there exists a concept description C_2, with $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C_2 \sqsubseteq C_1$.

Proof of Lemma 3.8. We have to show that for all atomic concept descriptions C and all individuals $e \in NInd(\mathcal{A})$, we have that $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. Assume that $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent. Thus, there exists a satisfying tableau run RUN for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)}$

$(\mathcal{A}) \cup \{\neg C(e)\}\rangle$. It is easy to see that $RUN^{+\{R(a,b)\}}$ is a tableau run for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\}\rangle$. All tableau rules are still applicable on the respective ABox in the tableau run $RUN^{+\{R(a,b)\}}$. Please note that especially the $\exists$-tableau rules remain applicable, since the applicability condition requires an anonymous successor individual (a and b are named individuals). In the remaining part, we only discuss the completeness of the leaf ABox of $RUN^{+\{R(a,b)\}}$.

Since the leaf ABox $\mathcal{A}_{leaf}$ of RUN is complete, the only tableau rule which could become applicable due to the ABox extension is the $\forall$-tableau rule. Please note that we must have $C_1(b) \in \mathcal{A}_{leaf}$ (since $\mathcal{A}_{leaf}$ is a complete ABox), and thus the $\forall$-tableau rule cannot become applicable for the new role assertion $R(a,b)$ and concept description $\forall R.C_1$. In particular the $TBox$-tableau rule cannot become applicable, since we have $a \in Ind(\mathcal{A}_{leaf})$ and $b \in Ind(\mathcal{A}_{leaf})$.

The leaf ABox of $RUN^{+\{R(a,b)\}}$ is $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. It is easy to see that this addition yields no immediate clash in $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. Thus, $RUN^{+\{R(a,b)\}}$ is satisfying and $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\}\rangle$ is consistent. □

We discuss the propagation of directly contradicting information next. If a propagation will only yield a direct clash due to disjointness information, we can break up the role assertion as well.

Lemma 3.9 (Redundant Contradiction-Propagation ABox Splits)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is a consistency-preserving ABox split for $\mathcal{O}$ if $info_{\mathcal{T}}^{\forall}(R) = \{C_1\}$ and there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C_1 \sqcap C_2 \sqsubseteq \bot$.

Proof of Lemma 3.9. We have to show that for all atomic concept descriptions C and all individuals $e \in NInd(\mathcal{A})$, we have that $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\}\rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\}\rangle$ is consistent. Assume that $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\}\rangle$ is consistent. Thus, there exists a satisfying tableau run RUN for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\}\rangle$. It is easy to see that $RUN^{+\{R(a,b)\}}$ is a tableau run for $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e) \cup \{R(a,b)\}\rangle$. All tableau rules are still applicable on the respective ABox in the tableau run $RUN^{+\{R(a,b)\}}$. Please note that especially the $\exists$-tableau rules remain applicable, since the applicability condition requires an anonymous successor individual (a and b are named individuals). In the remaining part, we only discuss the completeness of the leaf ABox of $RUN^{+\{R(a,b)\}}$.

Since the leaf ABox $\mathcal{A}_{leaf}$ of RUN is complete, the only tableau rule which could become applicable due to the ABox extension is the $\forall$-tableau rule. However, if the $\forall$-tableau rule becomes applicable for $R(a,b)$, then it must have been already applicable in RUN for the role assertion $R(a,d)$ and we must have $C_1(d) \in \mathcal{A}_{leaf}$. This must have yielded a clash, since $\mathcal{T} \models C_1 \sqcap C_2 \sqsubseteq \bot$ and $C_2(d) \in \mathcal{A}_{leaf}$.

Since RUN is satisfying, d does not contain that clash, and thus the $\forall$-tableau rule was not applicable to $R(a,d)$ in RUN and it cannot be applicable to $R(a,b)$ either. Thus the

$\forall$-tableau rule is not applicable. The $TBox$-tableau rule cannot become applicable, since we have $a \in Ind(\mathcal{A}_{leaf})$ and $b \in Ind(\mathcal{A}_{leaf})$.

The leaf ABox of $RUN^{+\{R(a,b)\}}$ is $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. It is easy to see that this addition yields no immediate clash in $\mathcal{A}_{leaf} \cup \{R(a,b)\}$. Thus, $RUN^{+\{R(a,b)\}}$ is satisfying and $\langle \mathcal{T}, \mathcal{R}, \downarrow_{c,d}^{R(a,b)} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\}\rangle$ is consistent. □

In Theorem 3.2, we summarize the above results about decision criteria for ABox splits over $\mathcal{ALC}$-ontologies.

Theorem 3.2 (Decision Criteria for ABox Splits in $\mathcal{ALC}$-ontologies)**:**
Given an $\mathcal{ALC}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is valid for $\mathcal{O}$ if for each $C \in info_{\mathcal{T}}^{\forall}(R)$

- $C = \bot$ or
- there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C_2 \sqsubseteq C$ or
- there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C \sqcap C_2 \sqsubseteq \bot$.

Proof of Theorem 3.2. Direct consequence of Lemma 3.4 (soundness), Lemma 3.5, Lemma 3.6, Lemma 3.7, Lemma 3.8 and Lemma 3.9. □

3.3.3 Consistency-preserving ABox Splits for $\mathcal{ALCH}$

In the following, we extend our results for valid ABox splits step-by-step from $\mathcal{ALC}$-ontologies to $\mathcal{SHI}$-ontologies. First, we add role hierarchies to $\mathcal{ALC}$.

In presence of role hierarchies, the $\forall$-info structure needs to be extended in order to handle role subsumptions, because propagations of concept descriptions can now occur over more role descriptions (all super roles).

Definition 3.17 (Extended $\forall$-info Structure)**:**
Given a TBox $\mathcal{T}$ in normal form and an RBox $\mathcal{R}$, an *extended $\forall$-info structure for $\mathcal{T}$ and $\mathcal{R}$* is a function $extinfo_{\mathcal{T},\mathcal{R}}^{\forall} : \mathbf{Rol} \rightarrow \wp(\mathbf{Con})$, such that we have $C \in extinfo_{\mathcal{T},\mathcal{R}}^{\forall}(R)$ if and only if there exists a role $R_2 \in \mathbf{Rol}$, such that $\mathcal{O} \vDash R \sqsubseteq R_2$ and $\forall R_2.C \in clos(\mathcal{T})$.

Example 3.11 (Example for an Extended $\forall$-info Structure)**:**
Let

$$\mathcal{T}_{Ex3.11} = \{ Chair \sqsubseteq \forall headOf.Department, \exists memberOf.\top \sqsubseteq Person, GraduateStudent \sqsubseteq Student \}$$

and

$$\mathcal{R}_{Ex3.11} = \{headOf \sqsubseteq memberOf\},$$

then the TBox in normal form is

$$\begin{aligned}\mathcal{T}_{Ex3.11norm} = \{ & \\ & \top \sqsubseteq \neg Chair \sqcup \forall headOf.Department, \\ & \top \sqsubseteq \forall memberOf.\bot \sqcup Person, \\ & \top \sqsubseteq \neg GraduateStudent \sqcup Student \\ \}&\end{aligned}$$

and the extended ∀-info structure for $\mathcal{T}_{Ex3.11norm}$ and $\mathcal{R}_{Ex3.11}$ is:

$$extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R) = \begin{cases} \{Department, \bot\} & \text{if } R = headOf, \\ \{\bot\} & \text{if } R = memberOf, \\ \emptyset & \text{otherwise.} \end{cases}$$

The extended ∀-info structure allows us to check which concept descriptions are (in the worst case) propagated over role assertions in $\mathcal{ALCH}$-ontologies. Before presenting the formal proof, we revisit the neighborship of individuals in $\mathcal{ALCH}$-ontologies.

Proposition 3.8 (Role Assertion Neighbor Impact in $\mathcal{ALCH}$)**:**
Given an $\mathcal{ALCH}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a role description $R \in \mathbf{Rol}$, and two individuals $a_1 \in Ind(\mathcal{A})$ and $a_2 \in Ind(\mathcal{A})$, a_2 is an R-neighbor of a_1 if and only if there exists a role description R_2, such that $\mathcal{O} \vDash R_2 \sqsubseteq R$, and $R_2(a_1, a_2) \in \mathcal{A}$.

Proof of Proposition 3.8. By Definition 2.31, Definition 2.30 and the absence of inverse roles in $\mathcal{ALCH}$. □

Lemma 3.10 (Extended ∀-Info Structure and Tableau Rule Applications in $\mathcal{ALCH}$)**:**
Given a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$ for an $\mathcal{ALCH}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an extended ∀-info structure $extinfo^{\forall}_{\mathcal{T},\mathcal{R}}$ for $\mathcal{T}$ and $\mathcal{R}$, the ∀-tableau rule is applicable to a role assertion $R(a_1, a_2)$ and a concept description $\forall R_2.C$ for the ABox $\phi^{abox}(n)$ of a node $n \in \mathbf{N}$, only if $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$.

Proof of Lemma 3.10. If the ∀-tableau rule is applicable to a role assertion $R(a_1, a_2)$ and a concept description $\forall R_2.C$, then, by Proposition 3.8, we must have $\forall R_2.C(a_1) \in \phi^{abox}(n)$, such that $\mathcal{O} \vDash R \sqsubseteq R_2$. By Proposition 3.5 we conclude that $\forall R_2.C \in clos(\mathcal{T})$ and by Definition 3.17, we have $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$. □

We extend Theorem 3.2 to $\mathcal{ALCH}$-ontologies in Theorem 3.3.

Theorem 3.3 (Decision Criteria for ABox Splits in $\mathcal{ALCH}$-ontologies)**:**
Given an $\mathcal{ALCH}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow^{R(a,b)}_{c,d}$, $\downarrow^{R(a,b)}_{c,d}$ is valid with respect to $\mathcal{O}$ if for each $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$

- $C = \bot$ or

- there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C_2 \sqsubseteq C$ or
- there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C \sqcap C_2 \sqsubseteq \bot$.

Proof of Theorem 3.3. Since the tableau rules for $\mathcal{ALCH}$ do not change compared to $\mathcal{ALC}$, but only the definition of neighbor relationships, the proof is a direct consequence of Lemma 3.10 and the results for $\mathcal{ALC}$ (Theorem 3.2). □

3.3.4 Consistency-preserving ABox Splits for $\mathcal{ALCHI}$

In presence of inverse roles, we have to adapt our approach a little bit, since concept descriptions can be propagated in two directions over a role assertion $R(a_1, a_2)$. We refine our definitions below.

Proposition 3.9 (Role Assertion Neighbor Impact in $\mathcal{ALCHI}$)**:**
Given an $\mathcal{ALCHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, two individuals $a_1 \in Ind(\mathcal{A})$ and $a_2 \in Ind(\mathcal{A})$, and a role description $R \in \mathbf{Rol}$, a_2 is an R-neighbor of a_1 if and only if

- there exists a role description R_2, such that $\mathcal{O} \models R_2 \sqsubseteq R$, and $R_2(a_1, a_2) \in \mathcal{A}$ or
- there exists a role description R_2, such that $\mathcal{O} \models R_2 \sqsubseteq R^-$, and $R_2(a_2, a_1) \in \mathcal{A}$.

Proof of Proposition 3.9. By Definition 2.30 and Definition 2.31. □

Lemma 3.11 (Extended ∀-Info Structure and Tableau Rule Applications in $\mathcal{ALCHI}$)**:**
Given a tableau run $RUN = \langle \mathbf{N}, root, children, \phi^{abox}, \phi^{ruleapp} \rangle$ for an $\mathcal{ALCHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an extended ∀-info structure $extinfo^{\forall}_{\mathcal{T},\mathcal{R}}$ for $\mathcal{T}$ and $\mathcal{R}$, the ∀-tableau rule is applicable to a role assertion $R(a_1, a_2)$ and a concept description $\forall R_2.C$ in the ABox $\phi^{abox}(n)$ of a node $n \in \mathbf{N}$ only if $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$ or $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$.

Proof. If the ∀-tableau rule is applicable to a role assertion $R(a_1, a_2)$ and a concept description $\forall R_2.C$, then, by Proposition 3.9, we must have one of two cases (depending on whether the ∀-tableau rule is applicable to individual a_1 or individual a_2):

- $\forall R_2.C(a_1) \in \phi^{abox}(n)$, for a role description R_2, such that $\mathcal{O} \models R \sqsubseteq R_2$. Since $\forall R_2.C$ is a non-atomic concept description, we can conclude, by Proposition 3.5, that $\forall R_2.C \in clos(\mathcal{T})$ and, by Definition 3.17, $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R)$.
- $\forall R_2.C(a_2) \in \phi^{abox}(n)$, for one role description R_2, such that $\mathcal{O} \models R \sqsubseteq R_2^-$. Since $\forall R_2.C$ is a non-atomic concept description, we can conclude, by Proposition 3.5, that $\forall R_2.C \in clos(\mathcal{T})$ and, by Definition 3.17, $C \in extinfo^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$.

□

The extension of Theorem 3.3 to $\mathcal{ALCHI}$-ontologies is shown in Theorem 3.4.

Theorem 3.4 (Decision Criteria for ABox Splits in $\mathcal{ALCHI}$-ontologies)**:**
Given an $\mathcal{ALCHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is valid with respect to $\mathcal{O}$ if

1. for each $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R)$
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C_2 \sqsubseteq C$ or
   - there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \models C \sqcap C_2 \sqsubseteq \bot$

   and
2. for each $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \models C_2 \sqsubseteq C$ or
   - there exists a concept description C_2, such that $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \models C \sqcap C_2 \sqsubseteq \bot$.

Proof of Theorem 3.4. Since the tableau rules for $\mathcal{ALCHI}$ do not change compared to $\mathcal{ALCH}$, but only the definition of neighbor relationships, the proof is a direct consequence of Lemma 3.11 and the results for $\mathcal{ALCH}$ (Theorem 3.3). □

3.3.5 Consistency-preserving ABox Splits for $\mathcal{SHI}$

We discuss the extension to transitive roles next. Please note that the additional $\forall_+$-tableau rule can only become applicable for role assertions with transitive roles. Although this insight is obvious, we formally declare this fact in Proposition 3.10.

Proposition 3.10 ($\mathcal{SHI}$-Susceptibility and Tableau Rule Applications in $\mathcal{SHI}$)**:**
Given a tableau run $\mathit{RUN} = \langle \mathbf{N}, \mathit{root}, \mathit{children}, \phi^{\mathit{abox}}, \phi^{\mathit{ruleapp}} \rangle$ for a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, the $\forall_+$-tableau rule is applicable to a role assertion $R(a_1, a_2)$ and a concept description C, in the ABox $\phi^{\mathit{abox}}(n)$ of a node $n \in \mathbf{N}$ only if there exists a role description R_2, such that $\mathcal{O} \models R \sqsubseteq R_2$ and R_2 is transitive with respect to $\mathcal{O}$.

Proof of Proposition 3.10. By Definition 2.38. □

We formally define a class of $\mathcal{SHI}$-splittable role assertions, and prove that each ABox split with respect to these role assertions is valid in $\mathcal{SHI}$-ontologies.

Definition 3.18 ($\mathcal{SHI}$-splittability of Role Assertions)**:**
Given a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a role assertion $R(a, b)$, we say that $R(a, b)$ is *$\mathcal{SHI}$-splittable with respect to* $\mathcal{O}$ if

1. there exists no transitive role R_2 with respect to $\mathcal{O}$, such that $\mathcal{O} \models R \sqsubseteq R_2$,

2. for each $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R)$
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C_2 \sqsubseteq C$ or
   - there exists a concept description C_2, such that $C_2(b) \in \mathcal{A}$ and $\mathcal{T} \vDash C \sqcap C_2 \sqsubseteq \bot$

   and
3. for each $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \vDash C_2 \sqsubseteq C$ or
   - there exists a concept description C_2, such that $C_2(a) \in \mathcal{A}$ and $\mathcal{T} \vDash C \sqcap C_2 \sqsubseteq \bot$.

Please note that the first criterion in Definition 3.18 is conservative. Although we believe that the criterion can be extended, for the remaining part of the thesis we investigate this simple criterion.

Lemma 3.12 (Consistency-preserving $\mathcal{SHI}$-ABox Splits)**:**
Given a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow^{R(a,b)}_{c,d}$, $\downarrow^{R(a,b)}_{c,d}$ is a consistency-preserving ABox split for $\mathcal{O}$ if $R(a,b)$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$.

Proof of Lemma 3.12. We have to show that for all atomic concept descriptions C and all individuals $e \in \mathit{Ind}(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent $\implies$ $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. Assume that $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$ is consistent. Thus, there exists a satisfying tableau run RUN for $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \rangle$. It is easy to see that $\mathit{RUN}^{+\{R(a,b)\}}$ is a tableau run for $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$. All tableau rules are still applicable on the respective ABox in the tableau run $\mathit{RUN}^{+\{R(a,b)\}}$. Please note that especially the $\exists$-tableau rules remain applicable, since the applicability condition requires an anonymous successor individual. In the remaining part, we only discuss the completeness of the leaf ABox of $\mathit{RUN}^{+\{R(a,b)\}}$.

By Theorem 3.4 we know that the only rule which could become applicable is the $\forall_+$-tableau rule. However, since we assume that R does not have a transitive subsuming role ($\mathcal{SHI}$-splittability of R) we can conclude by Proposition 3.10 that the $\forall_+$-tableau rule is not applicable either. The same argumentation as before ($\mathcal{ALCHI}$) is true for non-applicability of the $\forall$-tableau rule.

The leaf ABox of $\mathit{RUN}^{+\{R(a,b)\}}$ is $\mathcal{A}_{\mathit{leaf}} \cup \{R(a,b)\}$. It is easy to see that this addition yields no immediate clash in $\mathcal{A}_{\mathit{leaf}} \cup \{R(a,b)\}$. Thus, $\mathit{RUN}^{+\{R(a,b)\}}$ is satisfying and $\langle \mathcal{T}, \mathcal{R}, \downarrow^{R(a,b)}_{c,d} (\mathcal{A}) \cup \{\neg C(e)\} \cup \{R(a,b)\} \rangle$ is consistent. □

Theorem 3.5 is the extension of Theorem 3.4 from $\mathcal{ALCHI}$ to $\mathcal{SHI}$-ontologies.

Theorem 3.5 (Decision Criteria for ABox Splits in $\mathcal{SHI}$-ontologies)**:**
Given a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an ABox split $\downarrow_{c,d}^{R(a,b)}$, $\downarrow_{c,d}^{R(a,b)}$ is valid with respect to $\mathcal{O}$ if $R(a,b)$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$.

Proof of Theorem 3.5. By Lemma 3.2 we have soundness and by Lemma 3.12 we have completeness. □

In Example 3.12, we define an example ontology and then derive one intensional-based ABox modularization step by step.

Example 3.12 (Example Ontology for Intensional Modularization)**:**
The example ontology $\mathcal{O}_{Ex3.12} = \langle \mathcal{T}_{Ex3.12}, \mathcal{R}_{Ex3.12}, \mathcal{A}_{Ex3.12} \rangle$ is defined as follows

$$\begin{aligned}\mathcal{T}_{Ex3.12} = \{ & \\ & Chair \equiv \exists headOf.Department, Student \equiv \exists takes.Course, \\ & UndergraduateCourse \sqsubseteq Course, \\ & Course \sqcap Chair \sqsubseteq \bot, \top \sqsubseteq \forall takes.Course, \\ & \top \sqsubseteq \forall teaches.Course, \exists memberOf.\top \sqsubseteq Professor \\ & \}\end{aligned}$$

$$\mathcal{R}_{Ex3.12} = \{headOf \sqsubseteq memberOf, teaches \equiv isTaughtBy^{-}, Trans(suborgOf)\}$$

$$\begin{aligned}\mathcal{A}_{Ex3.12} = \{ & \\ & Department(cs), Department(ee), \\ & Professor(ann), Professor(eve), Professor(mae), \\ & UndergraduateCourse(c1), UndergraduateCourse(c4), \\ & UndergraduateCourse(c5), \\ & GraduateCourse(c2), GraduateCourse(c3), \\ & Student(ani), Student(ean), Student(eva), Student(noa), \\ & Student(sam), Student(sue), Student(zoe), \\ & headOf(ann, cs), memberOf(eve, cs), headOf(mae, ee), \\ & teaches(ann, c1), teaches(eve, c2), teaches(eve, c3), \\ & teaches(mae, c4), teaches(mae, c5), \\ & suborgOf(r, cs), suborgOf(cs, u1), suborgOf(ee, u1), \\ & takes(ani, c1), takes(ean, c1), takes(ean, c2), takes(eva, c3), \\ & takes(noa, c3), takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\ & \}.\end{aligned}$$

Please note the absence of the concept inclusion axiom $GraduateCourse \sqsubseteq Course$ in $\mathcal{T}_{Ex3.12}$. Absence and presence of that axiom makes the impact of TBox modeling for $\mathcal{SHI}$-splittability clear. We add the axiom later again.

The extended ∀-info structure for $\mathcal{T}_{Ex3.12}$ and $\mathcal{R}_{Ex3.12}$ is:

$$extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(R) = \begin{cases} \{\neg Department, \bot\} & \text{if } R = headOf, \\ \{Course\} & \text{if } R = isTaughtBy^{-}, \\ \{\bot\} & \text{if } R = memberOf, \\ \{\neg Course, Course\} & \text{if } R = takes, \\ \{Course\} & \text{if } R = teaches, \\ \emptyset & \text{otherwise.} \end{cases}$$

We have for instance $\bot \in extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(headOf)$ because of the subsumption relationship between $headOf$ and $memberOf$ in the RBox $\mathcal{R}_{Ex3.12}$. Given the extended ∀-info structure for $\mathcal{O}_{Ex3.12}$, we can decide $\mathcal{SHI}$-splittability for each role assertion in $\mathcal{A}_{Ex3.12}$. For instance, the role assertion $memberOf(eve, cs)$ is $\mathcal{SHI}$-splittable because of

- $extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(memberOf) = \{\bot\}$ and
- $extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(memberOf^{-}) = \{\}$.

The role assertion $takes(noa, c3)$ is not $\mathcal{SHI}$-splittable because of

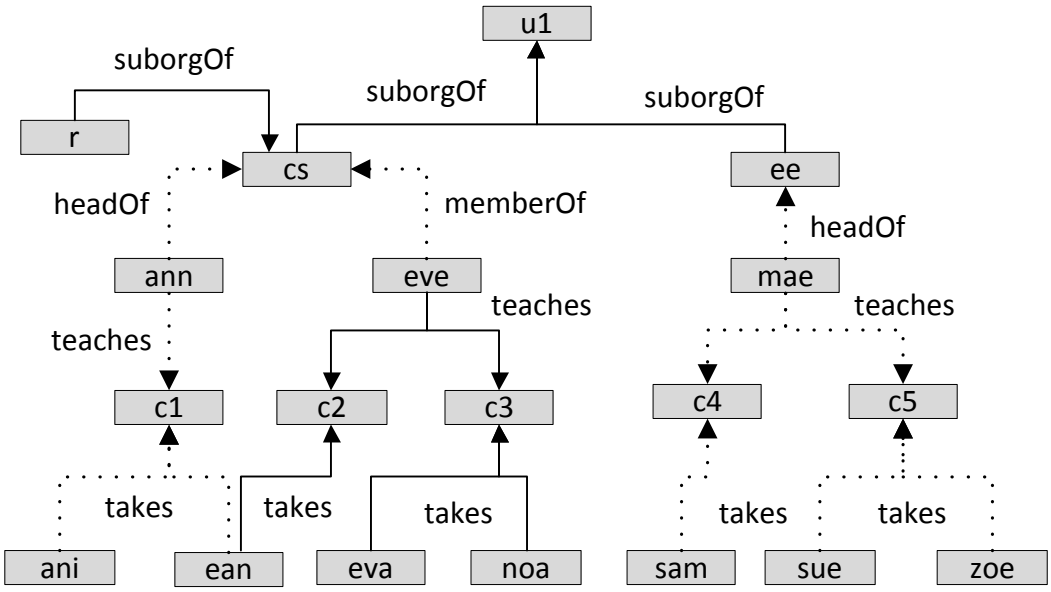
- $extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(takes) = \{\neg Course, Course\}$ and
- $extinfo^{\forall}_{\mathcal{T}_{Ex3.12},\mathcal{R}_{Ex3.12}}(takes^{-}) = \{\}$.

The problem is that the concept description $\neg Course$ can be propagated via role description $takes$. Since we only know that individual $c3$ is an instance of the concept description $GraduateCourse$, we cannot find an obvious propagation and neither a direct clash. Please note that this role assertion would be $\mathcal{SHI}$-splittable if we had a subsumption axiom between $GraduateCourse$ and $Course$. Then, the propagation of $\neg Course$ will be identified as a direct clash. Furthermore, all transitive $suborgOf$-role assertions are not $\mathcal{SHI}$-splittable.

In Figure 3.4, we show all role assertions in $\mathcal{A}_{Ex3.12}$ and their $\mathcal{SHI}$-splittability. All $\mathcal{SHI}$-splittable role assertions are shown with dashed lines and all $\mathcal{SHI}$-unsplittable role assertions are shown with normal lines.

In Figure 3.5, we show all role assertions in $\mathcal{A}_{Ex3.12}$ and their $\mathcal{SHI}$-splittability, if the concept inclusion axiom $GraduateCourse \sqsubseteq Course$ was present. With the axiom included, all role assertions except role assertions involving transitive roles in $\mathcal{A}_{Ex3.12}$ become $\mathcal{SHI}$-splittable. This simple example shows how important the correct modeling of (maybe obvious) domain knowledge can be for intensional modularization. Our experiments in Chapter 6 show similar results for real world ontologies.

Figure 3.4 $\mathcal{SHI}$-splittability for Example 3.12



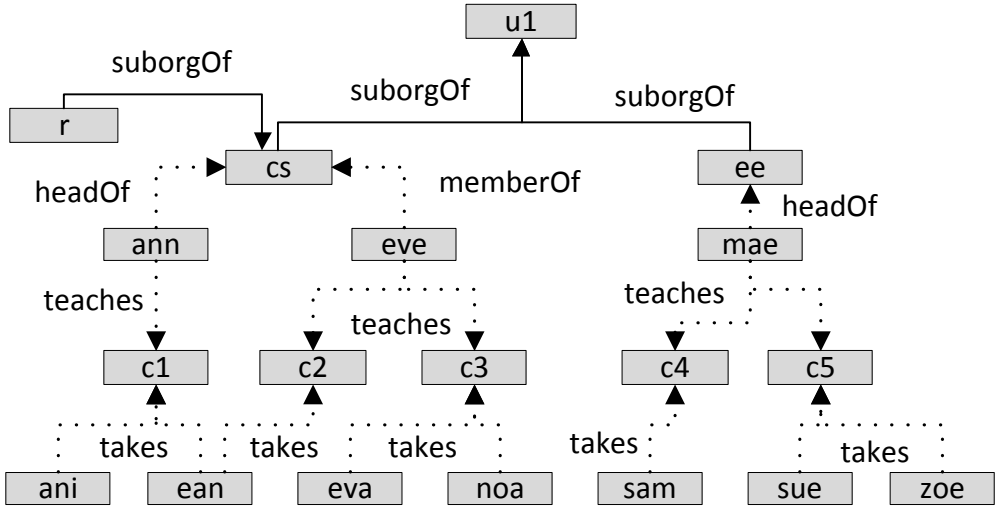
Informally speaking, a more detailed look at ontology $\mathcal{O}_{Ex3.12}$ shows that the $suborgOf$-role assertions have an influence on relation checking and retrieval only, since there is no $\forall$-propagation over $suborgOf$ possible/required. Thus, for instance checking and retrieval, even transitive role assertions could be split up here. We discuss this special case in Chapter 6 again.

3.4 Concluding Remarks

In Chapter 3, we have defined approaches for modularization of the assertional part of an ontology. First, a graph component-based modularization was proposed. In order to improve the granularity of graph component-based modularizations, we have proposed ABox splits. For ontologies with expressivity up to $\mathcal{SHI}$ we propose necessary criteria to ensure soundness and completeness of reasoning after applying ABox splits.

We conjecture that ABox modularizations based on ABox splits can be extended to $\mathcal{SHIQ}$-ontologies, although our proof techniques are not directly applicable to description logics with cardinality restrictions. It is possible to perform a syntactical analysis of the terminological knowledge to decide whether a role description can be target of a maximum cardinality restriction. All role assertions with these role descriptions should be kept unsplittable. This analysis is somehow similar to what we did with transitive roles.

Figure 3.5 $\mathcal{SHI}$-splittability for Example 3.12 with subsumption



Extensions to $\mathcal{SHOIQ}$ could be possible by following the line of [RPZ10]. The idea is to determine all individuals (nominals) in a TBox which are related to each other. This can be done by syntactical analysis of the terminological part again. Given the outcome one would need to avoid ABox splits over individuals which are related by TBox axioms. However, there is no doubt that there is a lot of open work in order to extend our modularization techniques to $\mathcal{SHOIQ}$.

The effectiveness of our modularization techniques can be further improved. For instance, TBox modularization techniques can contribute to smaller ABox modularizations. If we are able to split up the TBox into different modules, we could create an ABox modularization for each TBox module. Since each TBox module only contains a subset of assertions from the original TBox, it is clear that more role assertions become $\mathcal{SHI}$-splittable. However, it needs to be shown, whether the overhead of several ABox modularizations in parallel, one for each TBox module, actually pays off.

In the same line, if the set of possible query concepts for instance checking and retrieval are known beforehand then it might be possible to further split up more role assertions. This could be investigated in further work as well.

In the next chapter, we use ABox modularizations in order to define algorithms for efficient instance checking and instance retrieval over $\mathcal{SHI}$-ontologies.

Chapter 4: Islands, Simulations and One-Step Nodes

So far, we have introduced approaches to modularize the assertional part of an ontology. In Chapter 4, we use these modularization techniques to define structures for efficient reasoning over ontologies in the average case.

In Section 4.1, we define an optimized way to perform instance checking for a given individual and a given atomic concept description. We formally define a subset of assertions of an ABox, called individual island, which is worst-case sufficient in order to have sound and complete instance checking. Informally speaking, we take the graph view of an ABox and, starting from the given individual, follow all role assertions in the graph until we reach a $\mathcal{SHI}$-splittable role assertion. We show that this strategy is sufficient for entailment of atomic concepts. We also propose a naive way for instance retrieval, by performing instance checks on each island separately.

In Section 4.2, we discuss how to optimize instance retrieval over islands. The main insight is that islands might be either equivalent or at least similar to each other. We describe criteria for the similarity of islands, in order to reduce the number of instance checks necessary for instance retrieval.

In Section 4.3, we introduce a new data structure called one-step nodes. These one-step nodes are partially motivated by the need for efficient similarity measures for individual islands. Furthermore, one-step nodes can be used as a kind of proxy to answer queries immediately in a sound (and often complete) way.

In Section 4.4, we describe how individual islands, simulations, and one-step nodes can be used for optimized instance checking and instance retrieval over ontologies.

This chapter is concluded with Section 4.5.

4.1 Islands for Individuals

In the following section, we define an optimized way to perform instance checking for a given named individual and a given atomic concept description. Usually, instance checking over ontologies is performed on the whole TBox, RBox, and ABox. Our goal is to formally identify a subset of assertions, called *individual island*, which is worst-case sufficient to perform sound and complete instance checking for a given individual. The formal foundations for these subsets of assertions have been set up in Chapter 3, where we show that, under some conditions, role assertions can be broken up while preserving

soundness and completeness of instance checking algorithms. First, in Definition 4.1, we formally define an individual island candidate with an arbitrary subset of the original ABox. The concrete computation of the subset is then further defined below.

Definition 4.1 (Individual Island Candidate)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a named individual $a \in Ind(\mathcal{A})$, an *individual island candidate*, is a tuple $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, such that $\mathcal{A}^{isl} \subseteq \mathcal{A}$.

We formally define interpretations and entailment over individual island candidates next.

Definition 4.2 (Individual Island Interpretation)**:**
Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$ and an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a *model of* ISL_a, denoted $\mathcal{I} \vDash ISL_a$, if $\mathcal{I} \vDash \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle$.

Definition 4.3 (Individual Island Candidate Entailment)**:**
Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, we say that ISL_a *entails a concept assertion* $C(a)$, denoted $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle \vDash C(a)$, if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash ISL_a \implies \mathcal{I} \vDash C(a)$. We say that ISL_a *entails a role assertion* $R(a_1, a_2)$, denoted $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle \vDash R(a_1, a_2)$, if for all interpretations $\mathcal{I}$, we have $\mathcal{I} \vDash ISL_a \implies \mathcal{I} \vDash R(a_1, a_2)$.

Please note that entailment of concept and role assertions can be directly reformulated as a decision problem over ontologies, i.e. $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle \vDash C(a) \iff \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle \vDash C(a)$. In order to evaluate the quality of an individual island candidate, we define soundness and completeness criteria for individual island candidates.

Definition 4.4 (Soundness and Completeness for Island Candidates)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, we say that ISL_a is *sound for instance checking in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$, $ISL_a \vDash C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash C(a)$. ISL_a is *complete for instance checking in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash C(a) \implies ISL_a \vDash C(a)$.

We say that ISL_a is *sound for relation checking in ontology* $\mathcal{O}$ if for all role descriptions $R \in \mathbf{Rol}$ and all individuals $a_2 \in NInd(\mathcal{A})$

- $ISL_a \vDash R(a, a_2) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash R(a, a_2)$ and
- $ISL_a \vDash R(a_2, a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash R(a_2, a)$.

ISL_a is *complete for relation checking in ontology* $\mathcal{O}$ if for all role descriptions $R \in \mathbf{Rol}$ and all individuals $a_2 \in NInd(\mathcal{A})$

- $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash R(a, a_2) \implies ISL_a \vDash R(a, a_2)$ and
- $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \vDash R(a_2, a) \implies ISL_a \vDash R(a_2, a)$.

We say that ISL_a is *sound for reasoning in ontology* $\mathcal{O}$ if ISL_a is sound for instance and relation checking in $\mathcal{O}$. We say that ISL_a is *complete for reasoning in ontology* $\mathcal{O}$ if ISL_a is complete for instance and relation checking in $\mathcal{O}$.

Definition 4.5 (Individual Island)**:**
Given an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$ for an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, ISL_a is called *individual island for* $\mathcal{O}$ if ISL_a is sound and complete for reasoning in $\mathcal{O}$.

An individual island candidate becomes an individual island if it can be used for sound and complete reasoning. It is easy to see that each individual island candidate is sound for reasoning since it contains a subset of the original ABox assertions.

In Figure 4.1, we define an algorithm which computes an individual island starting from a given named individual a. The set **agenda** manages the individuals which have to be visited. The set **seen** collects already visited individuals. Individuals are visited if they are connected by a chain of $\mathcal{SHI}$-unsplittable role assertions to a. We add the role assertions of all visited individuals and all concept assertions for visited individuals and their direct neighbors. Please note that we do not explicitly perform ABox splits here, i.e. we do not introduce renamings, but use the original individual names from the ABox. The reason is that we focus on single islands here.

Figure 4.1 Algorithm for computing an individual island

Input: Ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, individual $a \in NInd(\mathcal{A})$

Output: Individual island $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$

Algorithm:

Let agenda $= a$
Let seen $= \emptyset$
Let $\mathcal{A}^{isl} = \emptyset$
While agenda $\neq \emptyset$ **do**
  Remove a_1 from **agenda**
  Add a_1 to **seen**
  Let $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_1) \mid C(a_1) \in \mathcal{A}\}$
  For each $R(a_1, a_2) \in \mathcal{A}$
    $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{R(a_1, a_2) \in \mathcal{A}\}$
    If $R(a_1, a_2) \in \mathcal{A}$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$ **then**
      $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_2) \mid C(a_2) \in \mathcal{A}\}$
    else agenda $=$ **agenda** $\cup (\{a_2\} \setminus$ **seen**$)$
  For each $R(a_2, a_1) \in \mathcal{A}$
    $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{R(a_2, a_1) \in \mathcal{A}\}$
    If $R(a_2, a_1) \in \mathcal{A}$ is $\mathcal{SHI}$-splittable with respect to $\mathcal{O}$ **then**
      $\mathcal{A}^{isl} = \mathcal{A}^{isl} \cup \{C(a_2) \mid C(a_2) \in \mathcal{A}\}$
    else agenda $=$ **agenda** $\cup (\{a_2\} \setminus$ **seen**$)$

Proposition 4.1 (Island Computation yields a Subset of the Input ABox)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual $a \in NInd(\mathcal{A})$, the algorithm in Figure 4.1 computes an individual island candidate $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$ for a.

Proof of Proposition 4.1. Easy to see, since each addition to $\mathcal{A}^{isl}$ comes from $\mathcal{A}$. □

In Lemma 4.1, we show that the individual island of an individual suffices to decide entailment of atomic concept assertions for an individual.

Lemma 4.1 (Individual Island Dependencies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for all named individuals $a \in NInd(\mathcal{A})$ and atomic concept descriptions C, if $ISL_a \not\models C(a)$ then there exists no individual $diff \in NInd(\mathcal{A})$, such that $ISL_{diff} \models C(a)$.

Proof of Lemma 4.1. By contradiction: Assume that $ISL_a \not\models C(a)$ and there exists an individual island $ISL_{diff} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{diff}, diff \rangle$, such that $ISL_{diff} \models C(a)$. It is easy to see that $diff \neq a$ and $ISL_{diff} \neq ISL_a$ (ABox is not structurally equivalent). We know that all role assertions for individual a in ISL_{diff} are $\mathcal{SHI}$-splittable. Therefore, the role assertions for individual a can only be used to derive/propagate obvious concept descriptions. Since all the individual islands are consistent initially, we must have $ISL_{diff} \models C(a)$ only because of the presence of role assertions for individual a, concept assertions for a and its direct neighbors, and TBox axioms. Since all these axioms occur in ISL_a, we must have $ISL_a \models C(a)$ as well. Contradiction. □

Below, we show in Theorem 4.1 that the computed set of assertions is indeed sufficient for complete reasoning.

Theorem 4.1 (Island Computation yields Individual Island for Ontologies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual $a \in NInd(\mathcal{A})$, the algorithm in Figure 4.1 computes an individual island $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$ for a.

Proof of Theorem 4.1. The proof is done in four steps, following Definition 4.4:

- ISL_a is sound for instance checking in $\mathcal{O}$: We have to show that we have for all atomic concept descriptions $C \in \mathbf{AtCon}$ that $ISL_a \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. Assuming $ISL_a \models C(a)$, it follows that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle \models C(a)$, and thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \cup \{\neg C(a)\} \rangle$ is inconsistent. By Proposition 4.1 we know that $\mathcal{A}^{isl} \cup \{\neg C(a)\} \subseteq \mathcal{A} \cup \{\neg C(a)\}$. We can conclude that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{\neg C(a)\} \rangle$ is inconsistent, and thus $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$.
- ISL_a is sound for relation checking in $\mathcal{O}$: We have to show that we have for all role descriptions $R \in \mathbf{Rol}$ and all individuals $a_2 \in NInd(\mathcal{A})$ that
  - $ISL_a \models R(a, a_2) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a, a_2)$: By contraposition: We obtain $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models R(a, a_2) \implies ISL_a \not\models R(a, a_2)$. Assuming $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models R(a, a_2)$, we know that there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, but

$\mathcal{I} \not\models R(a, a_2)$. By Proposition 4.1, we know that $\mathcal{A}^{isl} \subseteq \mathcal{A}$, and thus $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle$. By $\mathcal{I} \not\models R(a, a_2)$ we can then conclude that $ISL_a \not\models R(a, a_2)$.

- $ISL_a \models R(a_2, a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a_2, a)$: By contraposition: We obtain $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models R(a_2, a) \implies ISL_a \not\models R(a_2, a)$. Assuming $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models R(a_2, a)$, we know that there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, but $\mathcal{I} \not\models R(a_2, a)$. By Proposition 4.1, we know that $\mathcal{A}^{isl} \subseteq \mathcal{A}$, and thus $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl} \rangle$. By $\mathcal{I} \not\models R(a_2, a)$ we can then conclude that $ISL_a \not\models R(a_2, a)$.

- ISL_a is complete for instance checking in $\mathcal{O}$: We have to show that for all atomic concept descriptions $C \in \mathbf{AtCon}$ and all individuals $a \in NInd(\mathcal{A})$ that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a) \implies ISL_a \models C(a)$. By contraposition: We have to show that $ISL_a \not\models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a)$. Assume that $ISL_a \not\models C(a)$. By Lemma 4.1, we know that no other individual island entails $C(a)$. Please note that the set of all individual islands can be rewritten to a component-based ABox modularization. Thus, by Definition 3.15 and Theorem 3.5, we know that $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a)$.

- ISL_a is complete for relation checking in $\mathcal{O}$: We have to show that for all role descriptions $R \in \mathbf{Rol}$ and all individuals $a_2 \in NInd(\mathcal{A})$ that

  - $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a, a_2) \implies ISL_a \models R(a, a_2)$: There are three (combinations of) reasons for entailment of a role assertion $R(a, a_2)$ in a $\mathcal{SHI}$-ontology:

    * $R_2(a, a_2) \in \mathcal{A}$ and $\mathcal{O} \models R_2 \sqsubseteq R$: It is easy to see that all potentially useful role assertions $R_2(a, a_2)$ are in $\mathcal{A}^{isl}$, since, by the computation of islands in Figure 4.1, all role assertions for a are added to $\mathcal{A}^{isl}$.
    * $R_2(a_2, a) \in \mathcal{A}$ and $\mathcal{O} \models R_2^- \sqsubseteq R$: It is easy to see that all potentially useful role assertions $R_2(a_2, a)$ are in $\mathcal{A}^{isl}$, since, by the computation of islands in Figure 4.1, all role assertions for a are added to $\mathcal{A}^{isl}$.
    * a and a_2 are connected by a chain of (subroles of) transitive roles: By the definition of valid ABox splits and $\mathcal{SHI}$-splittability, each role assertion with a transitive superrole connected to an individual is not $\mathcal{SHI}$-splittable, and thus will end up in the $\mathcal{A}^{isl}$ computed by the algorithm in Figure 4.1.

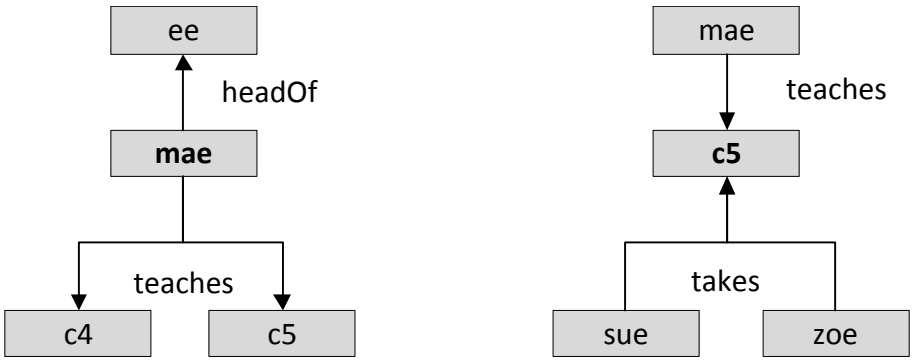
  - $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a_2, a) \implies ISL_a \models R(a_2, a)$: symmetric to the previous case.

□

In Figure 4.2, we show two example individual islands for individual *mae* and individual *c*5 from Example 3.12.

In the following, we define similarity measures for individual islands. First, a special case called *island identity* is introduced. Then we take a different view on similarity with a measure called *island similarity*. We try to find similar individual islands, for which we only have to perform one instance check after all.

Figure 4.2 Example individual island for *mae* and *c*5 in Example 3.12



Definition 4.6 (Individual Island Identity)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an individual island $ISL_{a_1} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1^{isl}, a_1 \rangle$ for individual a_1, and an individual island $ISL_{a_2} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl}, a_2 \rangle$ for individual a_2, we say that ISL_{a_1} and ISL_{a_2} are *identical*, if $\mathcal{A}_1^{isl} = \mathcal{A}_2^{isl}$.

It is easy to see that any application of the algorithm in Figure 4.1 yields the same individual island. This is formally shown in Proposition 4.2.

Proposition 4.2 (Individual Islands Are Unique)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an individual $a \in NInd(\mathcal{A})$, and two individual islands computed by the algorithm in Figure 4.1, both islands are identical.

Proof of Proposition 4.2. It is easy to see that by definition of the algorithm in Figure 4.1, the values of $\mathcal{T}$, $\mathcal{R}$, and a for both islands are identical, since these structures are deterministically added to the result island. Furthermore, the non-determinism introduced by the for loops, i.e. the order of iterating the neighbors of individuals does only affect the order adding ABox assertions to $\mathcal{A}^{isl}$. The result ABox $\mathcal{A}^{isl}$, as a set, is the same for any execution of the extraction algorithm. □

By uniqueness of individual islands, we refer to *the* individual island of a given named individual of an ontology below.

We have shown that an individual island can be used for sound and complete instance checks. In the average case, the size of the individual island (with respect to the number of assertion in its ABox) is considerably smaller than the original ABox. In our experiments the size is usually orders of magnitudes smaller. For qualitative and quantitative results, we refer to Chapter 6.

In the following, we extend our island-based approach from instance checking to instance retrieval. The first naive approach to solve an instance retrieval problem is to perform instance checking for each named individual in the ABox by using its individual island. In Definition 4.7, we define an individual island map, which assigns to each named individual its individual island.

Definition 4.7 (Individual Island Map)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a function $islandmap^{\mathcal{O}}$ is an *individual island map for* $\mathcal{O}$ if $islandmap^{\mathcal{O}}$ assigns the individual island ISL_a to each individual $a \in NInd(\mathcal{A})$.

In Definition 4.8, we define formal entailment of islands, and in Definition 4.9, we define the formal criteria for soundness and completeness with respect to reasoning over individual island maps.

Definition 4.8 (Individual Island Map Entailment)**:**
Given an individual island map $islandmap^{\mathcal{O}}$, we say that $islandmap^{\mathcal{O}}$ *entails a concept assertion* $C(a)$, denoted $islandmap^{\mathcal{O}} \models C(a)$, if $islandmap^{\mathcal{O}}(a) \models C(a)$. We say that $islandmap^{\mathcal{O}}$ *entails a role assertion* $R(a_1, a_2)$, denoted $islandmap^{\mathcal{O}} \models R(a_1, a_2)$, if $islandmap^{\mathcal{O}}(a_1) \models R(a_1, a_2)$.

Definition 4.9 (General Soundness and Completeness of Individual Island Maps)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual island map $islandmap^{\mathcal{O}}$, we say that $islandmap^{\mathcal{O}}$ is *sound for instance retrieval in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$ and all individuals $a \in NInd(\mathcal{A})$, $islandmap^{\mathcal{O}} \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. $islandmap^{\mathcal{O}}$ is *complete for instance retrieval in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$ and all individuals $a \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a) \implies islandmap^{\mathcal{O}} \models C(a)$.

We say that $islandmap^{\mathcal{O}}$ is *sound for relation retrieval in ontology* $\mathcal{O}$ if for all role descriptions $R \in \mathbf{Rol}$ and all pairs of individuals $a_1, a_2 \in NInd(\mathcal{A})$, $islandmap^{\mathcal{O}} \models R(a_1, a_2) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a_1, a_1)$. $islandmap^{\mathcal{O}}$ is *complete for relation retrieval in ontology* $\mathcal{O}$ if for all role descriptions $R \in \mathbf{Rol}$ and all pairs of individuals $a_1, a_2 \in NInd(\mathcal{A})$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models R(a_1, a_2) \implies islandmap^{\mathcal{O}} \models R(a_1, a_2)$.

We say that $islandmap^{\mathcal{O}}$ is *sound for reasoning in ontology* $\mathcal{O}$ if $islandmap^{\mathcal{O}}$ is sound for instance and relation retrieval in $\mathcal{O}$. We say that $islandmap^{\mathcal{O}}$ is *complete for reasoning in ontology* $\mathcal{O}$ if $islandmap^{\mathcal{O}}$ is complete for instance and relation retrieval in $\mathcal{O}$.

In Theorem 4.2, we show that an individual island map for an ontology $\mathcal{O}$ is indeed sound and complete for reasoning in $\mathcal{O}$. The proof of Theorem 4.2 is straightforward by reduction of instance retrieval over named individuals to instance checking.

Theorem 4.2 (Individual Island Maps are Sound and Complete for $\mathcal{SHI}$-Ontologies)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual island map $islandmap^{\mathcal{O}}$, $islandmap^{\mathcal{O}}$ is sound and complete for reasoning in ontology $\mathcal{O}$.

Proof of Theorem 4.2. By Theorem 4.1 and the fact that $islandmap^{\mathcal{O}}$ is a total function. □

The individual island map allows optimized instance checking, but can still be too naive for instance retrieval since we have to perform instance checks for each named individual in the ontology. In the following section we discuss how to further optimize instance retrieval over individual islands.

4.2 Simulation over Individual Islands

The computation of individual islands already has reduced the size of the (main memory) input for reasoning over ontologies. Another possibility for optimization is the reduction of the number of instance tests. In the literature, this approach is known, for instance, as binary instance retrieval [HM08].

In order to perform instance retrieval over n individuals, the idea is to group individuals for a combined instance check. For each individual a in a group g, the concept assertion axiom $\neg C(a)$ is added to the ontology. If the ontology is still consistent, then no individual from group g is an instance of concept description C. If the ontology is inconsistent, then at least one individual from the group g must be an instance of C. The approach is called binary instance retrieval, because one tries to split up sets of individuals step by step, until one has a consistent ontology or only one individual is left. The critical point here is obviously the process of grouping of individuals. This problem is closely related to finding a similarity measure for individuals.

From the computation of individual islands in Figure 4.1, it is clear that the individual islands for two individuals have the same individual island ABox if both individuals are connected by a (chain of) $\mathcal{SHI}$-unsplittable role assertion(s). However, this approach might be too naive for instance retrieval, especially if the number of individuals shared by one individual island is quite small.

Therefore, we propose a similarity measure. Graphs of different individual island ABoxes might be „structurally equivalent“, and thus, entail the same set of atomic concept descriptions for the root node (individual). This idea is formalized below. First, we define a graph view of individual islands.

Definition 4.10 (Individual Island Graph)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual island $ISL_a = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}^{isl}, a \rangle$, the *individual island graph* (for ISL_a) is a tuple $\mathbb{IIG}_a = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma, root \rangle$, such that $\langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$ is the ABox graph of $\mathcal{A}^{isl}$ and $root = a$.

Proposition 4.3 (Individual Island Graphs Are Unique)**:**
Given a $\mathcal{SHI}$-ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and two individual island graphs for an individual $a \in NInd(\mathcal{A})$, $\langle \mathbf{N}_1, \mathbf{E}_1, \phi_1, \sigma_1, root_1 \rangle$ and $\langle \mathbf{N}_2, \mathbf{E}_2, \phi_2, \sigma_2, root_2 \rangle$, both individual island graphs are identical.

Proof of Proposition 4.3. Easy to see, in particular, we have $root_1 = root_2$, since, by Definition 4.10, we have two islands for the same individual. □

Given the result in Proposition 4.3, we refer to *the* individual island graph of an individual island in the remaining part of this thesis. Our similarity measure over individual islands is based on the definition of homomorphism over labeled graphs. First, we define the notion of successors in Definition 4.11.

Definition 4.11 (ABox Graph Neighbor)**:**
Given an ABox graph $\mathbb{G}^{\mathcal{A}} = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$ and an individual node $a_1 \in \mathbf{N}$, then an individual node $a_2 \in \mathbf{N}$ is called *ρ-neighbor of* a_1, denoted $a_1 \xrightarrow{\rho}_{\mathbb{G}^{\mathcal{A}}} a_2$, if

$$\rho = \{R \mid R \in \sigma(a_1, a_2)\} \cup \{R^- \mid R \in \sigma(a_2, a_1)\}.$$

Given an individual island graph $\mathbb{IIG}_{a_1} = \langle \mathbf{N}, \mathbf{E}, \phi, \sigma, root \rangle$ and an individual node $a_1 \in \mathbf{N}$, then an individual node $a_2 \in \mathbf{N}$ is called *ρ-neighbor of* a_1, denoted $a_1 \xrightarrow{\rho}_{\mathbb{IIG}_a} a_2$, if $a_2 \in \mathbf{N}$ is a *ρ-neighbor of* a_1 with respect to $\langle \mathbf{N}, \mathbf{E}, \phi, \sigma \rangle$.

The formal notion of homomorphisms between individual island graphs is introduced in Definition 4.12. The criterion for homomorphisms is that the set of assigned concept descriptions is equal for nodes and their mapped nodes, and furthermore, that all nodes have the same kind of successors as their mapped nodes.

Definition 4.12 (Individual Island Graph Homomorphism)**:**
Given an individual island graph $\mathbb{IIG}_{a_1} = \langle \mathbf{N}_1, \mathbf{E}_1, \phi_1, \sigma_1, root_1 \rangle$ and an individual island graph $\mathbb{IIG}_{a_2} = \langle \mathbf{N}_2, \mathbf{E}_2, \phi_2, \sigma_2, root_2 \rangle$, an *individual island graph homomorphism from* $\mathbb{IIG}_{a_1}$ *to* $\mathbb{IIG}_{a_2}$ is a total function $\theta : \mathbf{N}_1 \to \mathbf{N}_2$, such that

- $\theta(root_1) = root_2$,
- for all nodes $n \in \mathbf{N}_1$, we have $\phi_1(n) = \phi_2(\theta(n))$, and
- $\forall n_1, n_{1*} \in \mathbf{N}_1$ we have that, if $n_1 \xrightarrow{\rho}_{\mathbb{IIG}_{a_1}} n_{1*}$, then $\theta(n_1) \xrightarrow{\rho}_{\mathbb{IIG}_{a_2}} \theta(n_{1*})$.

We use the definition of homomorphisms over individual islands graphs to formally define similarity over individual island graphs.

Definition 4.13 (Individual Island Graph Similarity)**:**
Given an individual island graph $\mathbb{IIG}_{a_1} = \langle \mathbf{N}_1, \mathbf{E}_1, \phi_1, \sigma_1, root_1 \rangle$ and an individual island graph $\mathbb{IIG}_{a_2} = \langle \mathbf{N}_2, \mathbf{E}_2, \phi_2, \sigma_2, root_2 \rangle$, we say that $\mathbb{IIG}_{a_1}$ and $\mathbb{IIG}_{a_2}$ are *similar* if there exists a homomorphism from $\mathbb{IIG}_{a_1}$ to $\mathbb{IIG}_{a_2}$ and there exists a homomorphism from $\mathbb{IIG}_{a_2}$ to $\mathbb{IIG}_{a_1}$.

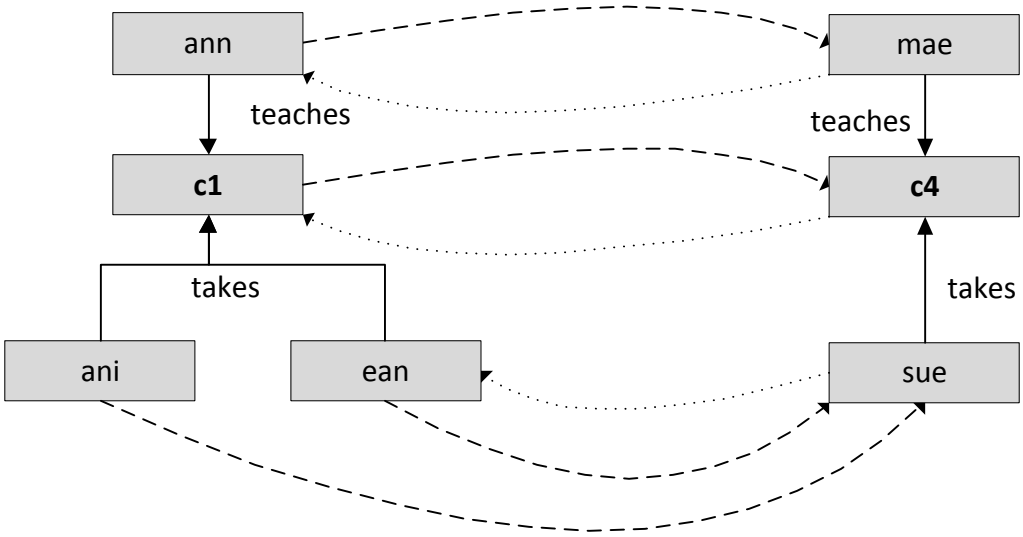
In Figure 4.3, we show two similar individual islands, one individual island for individual $c1$ and one individual island for individual $c4$ from Example 3.12. In addition, we show a homomorphism from $\mathbb{IIG}_{c1}$ to $\mathbb{IIG}_{c4}$, indicated by dashed lines, and a homomorphism from $\mathbb{IIG}_{c4}$ to $\mathbb{IIG}_{c1}$, indicated by dotted lines

For the sake of completeness, in Definition 4.14, we formally extend our similarity measure over individual islands.

Definition 4.14 (Individual Island Similarity)**:**
Given an individual island ISL_{a_1} with $\mathbb{IIG}_{a_1}$ and an individual island ISL_{a_2} with $\mathbb{IIG}_{a_2}$, we say that ISL_{a_1} and ISL_{a_2} are *similar* if $\mathbb{IIG}_{a_1}$ and $\mathbb{IIG}_{a_2}$ are similar.

In the following, we show that similar individual islands indeed entail the same set of atomic concept descriptions for their root nodes. To show this, we formally define the

Figure 4.3 Example individual islands for $c1$ and $c4$ (plus homomorphism) in Example 3.12



outcome of applying a homomorphism to an interpretation in Definition 4.15, and show in Lemma 4.2 that using the homomorphism a model for an individual island can be rewritten to a model for each similar individual island.

Definition 4.15 (Homomorphism Interpretation)**:**
Given an interpretation $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ and a homomorphism θ, the *homomorphism interpretation of $\mathcal{I}$ under θ*, denoted with $\mathcal{I}_\theta$, is defined as the interpretation $\mathcal{I}_\theta = \langle \Delta_{\mathcal{I}_\theta}, \cdot^{\mathcal{I}_\theta} \rangle$, such that:

- $\Delta_{\mathcal{I}_\theta} = \Delta_{\mathcal{I}}$ and
- $\cdot^{\mathcal{I}_\theta}(x) =$

$$\begin{cases} \cdot^{\mathcal{I}}(x) & \text{if } x \in \mathbf{CN}, \\ \cdot^{\mathcal{I}}(x) & \text{if } x \in \mathbf{RN}, \\ \cdot^{\mathcal{I}}(\theta(x)) & \text{if } x \in \mathbf{IN}. \end{cases}$$

Lemma 4.2 (Homomorphism Interpretation on Individual Islands)**:**
Given two individual islands $ISL_{a_1} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1^{isl}, a_1 \rangle$ and $ISL_{a_2} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl}, a_2 \rangle$, a homomorphism θ from ISL_{a_2} to ISL_{a_1}, and an interpretation $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, such that $\mathcal{I} \vDash ISL_{a_1}$, we have $\mathcal{I}_\theta \vDash ISL_{a_2}$.

Proof of Lemma 4.2. We know that $\mathcal{I}_\theta \vDash \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl} \rangle$ if $\mathcal{I}_\theta$ is a model for $\mathcal{T}$, $\mathcal{R}$, and $\mathcal{A}_2^{isl}$. This is shown for each component:

- $\mathcal{I}_\theta \vDash \mathcal{T}$: By contradiction: Assume there exists a concept inclusion axiom $C_1 \sqsubseteq C_2 \in \mathcal{T}$, such that $\mathcal{I}_\theta \nvDash C_1 \sqsubseteq C_2$. From $\mathcal{I} \vDash \mathcal{T}$, we know that for all $\delta \in \Delta_\mathcal{I}$, we have $\delta \in C_1^\mathcal{I} \implies \delta \in C_2^\mathcal{I}$. Since, by Definition 4.15, we have $\Delta_\mathcal{I} = \Delta_{\mathcal{I}_\theta}$ and an unchanged set of concept description labels for each domain element, we conclude that $\delta \in C_1^{\mathcal{I}_\theta} \implies \delta \in C_2^{\mathcal{I}_\theta}$ for all $\delta \in \Delta_{\mathcal{I}_\theta}$. Contradiction with $\mathcal{I}_\theta \nvDash C_1 \sqsubseteq C_2$.

- $\mathcal{I}_\theta \vDash \mathcal{R}$: We need to show that $\mathcal{I}_\theta$ satisfies all role inclusions axioms and role transitivity axioms in $\mathcal{R}$.

  - Role inclusion axioms: By contradiction: Assume there exists a role inclusion axiom $R_1 \sqsubseteq R_2$, such that $\mathcal{I}_\theta \nvDash R_1 \sqsubseteq R_2$. Thus, we have that $R_1^{\mathcal{I}_\theta} \nsubseteq R_2^{\mathcal{I}_\theta}$, which means that there exists two domain elements $\delta_1, \delta_2 \in \Delta_{\mathcal{I}_\theta}$, such that $(\delta_1, \delta_2) \in R_1^{\mathcal{I}_\theta}$ and $(\delta_1, \delta_2) \notin R_2^{\mathcal{I}_\theta}$. By Definition 4.15, we know that $\delta_1, \delta_2 \in \Delta_\mathcal{I}$. Since the role description labels are unchanged by Definition 4.15, we have that $(\delta_1, \delta_2) \in R_1^\mathcal{I}$ and $(\delta_1, \delta_2) \notin R_2^\mathcal{I}$. Contradiction, since we assume that $\mathcal{I} \vDash \mathcal{R}$ and thus $\mathcal{I} \vDash R_1 \sqsubseteq R_2$.

  - Role transitivity axioms: Same argumentation as before, since the transformation from Definition 4.15 does not change the role labels of any domain element, transitive connections are retained.

- $\mathcal{I}_\theta \vDash \mathcal{A}_2^{isl}$: We need to show that $\mathcal{I}_\theta$ satisfies all concept assertion axioms and role assertion axioms in $\mathcal{A}_2^{isl}$. By contradiction:

  - Assume $\mathcal{I}_\theta \nvDash C(a_3)$, such that $C(a_3) \in \mathcal{A}_2^{isl}$: We know $a_3^{\mathcal{I}_\theta} \notin C^{\mathcal{I}_\theta}$, and thus, by Definition 4.15, $(\theta(a_3))^\mathcal{I} \notin C^\mathcal{I}$. This yields $\mathcal{I} \nvDash C(\theta(a_3))$. However, by Definition 4.12, we know $C(\theta(a_3)) \in \mathcal{A}_1^{isl}$ and hence, by $\mathcal{I} \vDash \mathcal{A}_1^{isl}$, we must have $\mathcal{I} \vDash C(\theta(a_3))$. Contradiction.

  - Assume $\mathcal{I}_\theta \nvDash R(a_3, a_4)$, such that $R(a_3, a_4) \in \mathcal{A}_2^{isl}$: We know $(a_3^{\mathcal{I}_\theta}, a_4^{\mathcal{I}_\theta}) \notin R^{\mathcal{I}_\theta}$, and thus, by Definition 4.15, $((\theta(a_3))^\mathcal{I}, (\theta(a_4))^\mathcal{I}) \notin R^\mathcal{I}$. This yields $\mathcal{I} \nvDash R(\theta(a_3), \theta(a_4))$. However, by Definition 4.12, we know $R(\theta(a_3), \theta(a_4)) \in \mathcal{A}_1^{isl}$ and hence, by $\mathcal{I} \vDash \mathcal{A}_1^{isl}$, we must have $\mathcal{I} \vDash R(\theta(a_3), \theta(a_4))$. Contradiction.

□

In Lemma 4.3, we show that we can perform instance checking on one individual island and then, informally speaking, transfer the results to all similar individual islands.

Lemma 4.3 (Homomorphic Individual Islands Instance Checking)**:**
Given two individual islands, $ISL_{a_1} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1^{isl}, a_1 \rangle$ and $ISL_{a_2} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl}, a_2 \rangle$, such that ISL_{a_1} is similar to ISL_{a_2}, then, for all atomic concept descriptions C, $ISL_{a_1} \vDash C(a_1) \implies ISL_{a_2} \vDash C(a_2)$.

Proof of Lemma 4.3. By contraposition: We have to show that $ISL_{a_2} \nvDash C(a_2) \implies ISL_{a_1} \nvDash C(a_1)$. Assume that $ISL_{a_2} \nvDash C(a_2)$. Thus, there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \vDash ISL_{a_2}$ and $\mathcal{I} \nvDash C(a_2)$. Without loss of generality, let θ be any homomorphism

between ISL_{a_1} and ISL_{a_2}. By Lemma 4.2, we know that $\mathcal{I}_\theta \models ISL_{a_1}$ since ISL_{a_1} is similar to ISL_{a_2}. Then it is easy to see that $\mathcal{I}_\theta \not\models C(a_1)$, since by Definition 4.12 we have $\theta(a_1) = a_2$. Thus $ISL_{a_1} \not\models C(a_1)$. □

In Theorem 4.3, we summarize the main result about individual island similarity.

Theorem 4.3 (Similar Individual Islands Instance Checking)**:**
Given two similar individual islands ISL_{a_1} and ISL_{a_2}, for all atomic concept descriptions C, $ISL_{a_1} \models C(a_1)$ if and only if $ISL_{a_2} \models C(a_2)$.

Proof of Theorem 4.3. By application of Lemma 4.3. □

In the following, we discuss the computation of similarity of islands. In general, the problem of deciding whether a homomorphism between two graphs exists is a hard problem. If the decision whether two graphs are similar becomes too complex, then the similarity measure over islands does not have any performance gain. Although we already have one important advantage, since we know, during homomorphism search for individual islands, that one root node should be mapped to the other root node, we propose another idea how to further overcome performance problems as follows.

We try to approximate (non-)similarity of islands with a set of criteria. We focus on domain-specific approaches. First, we introduce a simple integrity constraint for similar islands.

Proposition 4.4 (Similar Islands Integrity Constraint)**:**
Given an individual island $ISL_{a_1} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1^{isl}, a_1 \rangle$ and an individual island $ISL_{a_2} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl}, a_2 \rangle$, we have that ISL_{a_1} and ISL_{a_2} are similar to each other only if

$$\{C \mid C(a_1) \in \mathcal{A}_1^{isl}\} = \{C \mid C(a_2) \in \mathcal{A}_2^{isl}\}.$$

Proof of Proposition 4.4. If ISL_{a_1} and ISL_{a_2} are similar to each other, then there exists a homomorphism θ from ISL_{a_1} to ISL_{a_2}, and vice versa. By Definition 4.12, we must have $\theta(a_1) = a_2$ and therefore $\phi_1(a_1) = \phi_2(a_2)$. By construction of individual island graphs in Definition 4.10, we can conclude that $\{C \mid C(a_1) \in \mathcal{A}_1^{isl}\} = \{C \mid C(a_2) \in \mathcal{A}_2^{isl}\}$. □

In practice, the integrity constraint of Proposition 4.4 rejects lots of islands immediately as dissimilar, because their root individuals are labeled with different concept sets. The integrity constraint can be further extended by making sure that, in addition, both root nodes have the same kinds of direct successors. This idea is further discussed below, in Section 4.3.

Before we continue, we would like to relate similarity of individual islands with the results from [DFK+09]. The similarity of individual islands gives rise to a similarity relation among individuals. Usually, reasoning about similar individuals can be performed on one representative individual directly.

In [DFK+09], the idea is to initially *merge all* individuals which have the same set of concept assertions. Once this merging yields inconsistencies - because the individuals turn out to be dissimilar - then the individual merging is undone. The similarity criteria in [DFK+09] is (for the sake of heuristics) chosen to be based on the set of concept assertions for each individual. Our individual island similarity approach can be seen as a generalization of these results. We have shown a formal foundation on *why* sets of individuals are similar and how to use the similarity during reasoning. Furthermore, we do not need a refinement step, since we identify dissimilar individuals as dissimilar immediately.

4.3 One-Step Nodes

In this section, we discuss a data structure which allows us to quickly decide, whether two individual islands are similar or not. Although we can use the homomorphism approach directly, we would like to introduce a simpler data structure. To be more precise, the data structure is used to detect several dissimilar islands immediately, and only if the data structure fails to show dissimilarity, then we apply further techniques to find a homomorphism.

The basic idea is to define a notion of so-called pseudo node neighbors, which represent the directly asserted successors of a named individual in an ABox. Then, for each individual in the ABox, the information about all pseudo node successors plus the information about the original individual is combined, to obtain so-called one-step nodes. In addition to similarity detection, these one-step nodes can be used to answer instance checking and instance retrieval queries directly.

First, in Definition 4.16, we formally define a pseudo node successor for an individual with respect to an ABox.

Definition 4.16 (Pseudo Node Successor)**:**
Given an ABox $\mathcal{A}$, a *pseudo node successor* of an individual $a \in NInd(\mathcal{A})$ is a pair $pns^{a,\mathcal{A}} = \langle \mathbf{rs}, \mathbf{cs} \rangle$, such that $\exists a_2 \in Ind(\mathcal{A})$ with

1. $\forall R \in \mathbf{rs}.(R(a, a_2) \in \mathcal{A} \vee R^-(a_2, a) \in \mathcal{A})$,
2. $\forall C \in \mathbf{cs}.C(a_2) \in \mathcal{A}$, and
3. **rs** and **cs** are maximal.

The third criteria (maximality) is important to ensure that for each pair of named individuals $\langle a, a_2 \rangle \in NInd(\mathcal{A}) \times NInd(\mathcal{A})$, we have that individual a_2 is *exactly* one pseudo node successor for individual a.

Example 4.1 (Example for Pseudo Node Successors)**:**
Given an ontology $\mathcal{O}_{Ex4.1} = \langle \mathcal{T}_{Ex4.1}, \mathcal{R}_{Ex4.1}, \mathcal{A}_{Ex4.1} \rangle$ as follows

$$\begin{aligned}
\mathcal{T}_{Ex4.1} &= \{GraduateStudent \sqsubseteq Student\} \\
\mathcal{R}_{Ex4.1} &= \{headOf \sqsubseteq memberOf\} \\
\mathcal{A}_{Ex4.1} &= \{ \\
&\quad Department(ee), Professor(mae), \\
&\quad UndergraduateCourse(c4), UndergraduateCourse(c5), \\
&\quad Student(sam), Student(sue), Student(zoe), \\
&\quad headOf(mae, ee), \\
&\quad teaches(mae, c4), teaches(mae, c5), \\
&\quad takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
&\},
\end{aligned}$$

the following are pseudo node successors:

- $pns_1^{mae,\mathcal{A}_{Ex4.1}} = \langle \{headOf\}, \{Department\} \rangle$,
- $pns_2^{mae,\mathcal{A}_{Ex4.1}} = \langle \{teaches\}, \{UndergraduateCourse\} \rangle$, and
- $pns_3^{c5,\mathcal{A}_{Ex4.1}} = \langle \{teaches^-\}, \{Professor\} \rangle$.

Next, we combine all pseudo node successors of a named individual a in an ABox $\mathcal{A}$, the reflexive role assertions for a, and the directly asserted concepts of a, in order to create a summarization representative, called one-step node.

Definition 4.17 (One-Step Node)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual $a \in NInd(\mathcal{A})$, the *one-step node of a for* $\mathcal{A}$, denoted $osn^{a,\mathcal{A}}$, is a tuple $osn^{a,\mathcal{A}} = \langle \mathbf{rootconset}, \mathbf{reflset}, \mathbf{pnsset} \rangle$, such that $\mathbf{rootconset} = \{C | C(a) \in \mathcal{A}\}$, $\mathbf{reflset} = \{R | R(a,a) \in \mathcal{A} \vee R^-(a,a) \in \mathcal{A}\}$, and $\mathbf{pnsset}$ is the set of all pseudo node successors of individual a. The set of all possible one-step nodes is denoted **OSN**.

Example 4.2 (Example for One-Step Nodes)**:**
Given ontology $\mathcal{O}_{Ex4.1} = \langle \mathcal{T}_{Ex4.1}, \mathcal{R}_{Ex4.1}, \mathcal{A}_{Ex4.1} \rangle$ from Example 4.1, one-step nodes are:

$$\begin{aligned}
osn^{mae,\mathcal{A}_{Ex4.1}} =& \langle \{Professor\}, \emptyset, \{\langle \{headOf\}, \{Department\} \rangle, \\
& \langle \{teaches\}, \{UndergraduateCourse\} \rangle\} \rangle \\
osn^{c5,\mathcal{A}_{Ex4.1}} =& \langle \{Student\}, \emptyset, \\
& \{\langle \{takes^-\}, \{Student\} \rangle, \langle \{teaches^-\}, \{Professor\} \rangle\} \rangle.
\end{aligned}$$

Definition 4.18 (One-Step Node Similarity)**:**
Two individuals a_1 and a_2 are called *one-step node similar* for an ABox $\mathcal{A}$ if $osn^{a_1,\mathcal{A}} = osn^{a_2,\mathcal{A}}$.

In order to apply tableau-based reasoning techniques to one-step nodes, we need to have some kind of serialization for one-step nodes into an ontology-based representation. This serialization is formally defined in Definition 4.19 for pseudo node successors and in Definition 4.20 for one-step nodes. Please note that we cannot use the original individual names of the pseudo node successors. Our intention is to abstract away from these individuals names for the sake of similarity among one-step nodes.

Definition 4.19 (Pseudo Node Successor ABox Realization)**:**
Given a pseudo node successor $pns^{a,\mathcal{A}} = \langle \mathbf{rs}, \mathbf{cs} \rangle$ and an individual a_2, the *ABox realization of* $pns^{a,\mathcal{A}}$ *with respect to* a_2, denoted $ABox^{a_2}(pns^{a,\mathcal{A}})$, is

$$ABox^{a_2}(pns^{a,\mathcal{A}}) = \bigcup_{C \in \mathbf{cs}} \{C(a_2)\} \cup \bigcup_{R \in \mathbf{rs}} \{R(a, a_2)\}.$$

Definition 4.20 (One-Step Node ABox Realization)**:**
Given a one-step node $osn^{a,\mathcal{A}} = \langle \mathbf{rootconset}, \mathbf{reflset}, \mathbf{pnsset} \rangle$, let $a_1, ..., a_n$ be individuals distinct with a, such that $n = |\mathbf{pnsset}|$. Furthermore, let f be a bijective function from **pnsset** to $\{a_1, ..., a_n\}$. An *ABox realization of* $osn^{a,\mathcal{A}}$, denoted $ABox(osn^{a,\mathcal{A}})$, is

$$\begin{aligned} ABox(osn^{a,\mathcal{A}}) = &\{C(a) \mid C \in \mathbf{rootconset}\} \cup \\ &\{R(a,a) \mid R \in \mathbf{reflset}\} \cup \\ &\bigcup_{pns^{a,\mathcal{A}} \in \mathbf{pnsset}} ABox^{f(pns^{a,\mathcal{A}})}(pns^{a,\mathcal{A}}). \end{aligned}$$

Example 4.3 (Example for One-Step Node Serialization)**:**
An example serialization for the one-step node $osn^{mae,\mathcal{A}_{Ex4.1}}$ from Example 4.2 is

$$\begin{aligned} ABox(osn^{mae,\mathcal{A}_{Ex4.1}}) = \{&Professor(mae), \\ &headOf(mae, newa), teaches(mae, newb)\}. \end{aligned}$$

Based on this one-step node serialization technique, we define entailment over one-step nodes in Definition 4.21 and soundness and completeness with respect to instance checking in Definition 4.22.

Definition 4.21 (One-Step Node Entailment)**:**
Given a TBox $\mathcal{T}$, a RBox $\mathcal{R}$, and a one-step node $osn^{a,\mathcal{A}}$ for an individual $a \in NInd(\mathcal{A})$, we say that $osn^{a,\mathcal{A}}$ *entails a concept assertion* $C(a)$, denoted $osn^{a,\mathcal{A}} \models_{\mathcal{T},\mathcal{R}} C(a)$, if we $\langle \mathcal{T}, \mathcal{R}, ABox(osn^{a,\mathcal{A}}) \rangle \models C(a)$. If $\mathcal{T}$ and $\mathcal{R}$ are clear from the context, then both are omitted and we denote concept assertion entailment with $osn^{a,\mathcal{A}} \models C(a)$.

Definition 4.22 (Instance Checking Soundness and Completeness for One-Step Nodes)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a one-step node $osn^{a,\mathcal{A}}$ for individual $a \in NInd(\mathcal{A})$, we say that $osn^{a,\mathcal{A}}$ is *sound for instance checking in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$, $osn^{a,\mathcal{A}} \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. We say that $osn^{a,\mathcal{A}}$ is *complete for instance checking in ontology* $\mathcal{O}$ if for all atomic concept descriptions $C \in \mathbf{AtCon}$, $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a) \implies osn^{a,\mathcal{A}} \models C(a)$.

The important instrument to prove soundness and completeness is the definition of a homomorphism into the source ABox. This homomorphism is formally defined in Definition 4.23.

Definition 4.23 (One-Step Node Homomorphism)**:**
Given

- an ABox $\mathcal{A}$ and its ABox graph $\mathbb{G}^{\mathcal{A}} = \langle \mathbf{N}_1, \mathbf{E}_1, \phi_1, \sigma_1 \rangle$,
- an individual $a \in NInd(\mathcal{A})$, and
- a one-step node $osn^{a,\mathcal{A}}$ with $\mathbb{G}^{ABox(osn^{a,\mathcal{A}})} = \langle \mathbf{N}_2, \mathbf{E}_2, \phi_2, \sigma_2 \rangle$,

a *one-step node homomorphism from* $osn^{a,\mathcal{A}}$ *to* $\mathcal{A}$ is a total function $\theta : \mathbf{N}_2 \to \mathbf{N}_1$, such that

- $\theta(a) = a$,
- for all nodes $n \in \mathbf{N}_2$, we have $\phi_2(n) = \phi_1(\theta(n))$, and
- $\forall n_1, n_{1*} \in \mathbf{N}_2$, we have that if $n_1 \xrightarrow{\rho}_{\mathbb{G}^{ABox(osn^{a,\mathcal{A}})}} n_{1*}$, then $\theta(n_1) \xrightarrow{\rho}_{\mathbb{G}^{\mathcal{A}}} \theta(n_{1*})$.

In Lemma 4.4, we show that, whenever such a homomorphism exists, then any model of the source ontology can be rewritten (see Definition 4.15) to a model over the one-step node serialization.

Lemma 4.4 (Homomorphism Interpretation on One-Step Nodes)**:**
Given

- an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$,
- a one-step node $osn^{a,\mathcal{A}}$ for an individual $a \in NInd(\mathcal{A})$,
- a homomorphism θ from $osn^{a,\mathcal{A}}$ to $\mathcal{A}$, and
- an interpretation $\mathcal{I} = \langle \Delta_{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, such that $\mathcal{I} \vDash \mathcal{O}$,

$\mathcal{I}_\theta \vDash \langle \mathcal{T}, \mathcal{R}, ABox(osn^{a,\mathcal{A}}) \rangle$.

Proof of Lemma 4.4. We have to show that $\mathcal{I}_\theta \vDash \langle \mathcal{T}, \mathcal{R}, ABox(osn^{a,\mathcal{A}}) \rangle$. This entailment is true if $\mathcal{I}_\theta$ is a model for $\mathcal{T}$, $\mathcal{R}$, and $ABox(osn^{a,\mathcal{A}})$. It is easily seen that the proof can be shown in the same style as the proof for Lemma 4.2. □

Using Lemma 4.4, it is straightforward to prove soundness of instance checking in Lemma 4.5.

Lemma 4.5 (One-Step Node Entailment is Sound for Instance Checking)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and a one-step node $osn^{a,\mathcal{A}}$ for an individual $a \in NInd(\mathcal{A})$, $osn^{a,\mathcal{A}}$ is sound for instance checking in ontology $\mathcal{O}$.

Proof of Lemma 4.5. We have to show that $osn^{a,\mathcal{A}} \models C(a) \implies \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. By contraposition: We obtain $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a) \implies osn^{a,\mathcal{A}} \not\models C(a)$. We assume $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \not\models C(a)$. Thus, there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, but $\mathcal{I} \not\models C(a)$. By Definition 4.16, Definition 4.17 and Definition 4.20, there must exist a homomorphism θ from $ABox(osn^{a,\mathcal{A}})$ to $\mathcal{A}$. By Lemma 4.4, we can conclude that $\mathcal{I}_\theta \models osn^{a,\mathcal{A}}$. By $\mathcal{I}_\theta \not\models C(a)$ (which follows from $\mathcal{I} \not\models C(a)$, $\theta(a) = a$, and the unchanged concept labels of the domain elements of $\mathcal{I}$ in $\mathcal{I}_\theta$), we obtain $osn^{a,\mathcal{A}} \not\models C(a)$. □

It is clear that not every one-step node is complete for instance checking. However, in case the one-step node coincides with the individual island, then we can show that instance checking over the one-step node is even complete. For this, we define so-called splittable one-step nodes, for which each role assertion to a direct neighbor is $\mathcal{SHI}$-splittable.

Definition 4.24 (Splittable One-Step Node)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an individual $a \in NInd(\mathcal{A})$, and a one-step node $osn^{a,\mathcal{A}} = \langle \mathbf{rootconset}, \mathbf{reflset}, \mathbf{pnsset} \rangle$, we say that $osn^{a,\mathcal{A}}$ is *splittable* if for each $\langle \mathbf{rs}, \mathbf{cs} \rangle \in \mathbf{pnsset}$, a fresh individual $a_2 \notin Ind(\mathcal{A})$, and for each $R \in \mathbf{rs}$, the role assertion axiom $R(a, a_2)$ is $\mathcal{SHI}$-splittable with respect to ontology $\mathcal{O}_2 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2 \rangle$ with

$$\mathcal{A}_2 = \{C(a) \mid C \in \mathbf{rootconset}\} \cup \{C(a_2) \mid C \in \mathbf{cs}\} \cup \{R(a, a_2)\}.$$

In Lemma 4.6 and Proposition 4.5, we show the actual proofs of completeness, by using coincidence of one-step nodes with individual islands.

Lemma 4.6 (Splittable One-Step Node Entailment Is Complete for Instance Checking)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ and an individual $a \in NInd(\mathcal{A})$ with a splittable one-step node $osn^{a,\mathcal{A}} = \langle \mathbf{rootconset}, \mathbf{reflset}, \mathbf{pnsset} \rangle$, $osn^{a,\mathcal{A}}$ is complete for instance checking in ontology $\mathcal{O}$.

Proof of Lemma 4.6. We have to show that, given a concept description $C \in \mathbf{AtCon}$, we have $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a) \implies osn^{a,\mathcal{A}} \models C(a)$. Assuming $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$, by Theorem 4.1, we know that $ISL_a \models C(a)$ from $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \models C(a)$. Thus, it is enough to show that $ISL_a \models C(a) \implies osn^{a,\mathcal{A}} \models C(a)$. By contraposition: We obtain $osn^{a,\mathcal{A}} \not\models C(a) \implies ISL_a \not\models C(a)$. Assuming $osn^{a,\mathcal{A}} \not\models C(a)$, we know that there exists an interpretation $\mathcal{I}$, such that $\mathcal{I} \models osn^{a,\mathcal{A}}$ and $\mathcal{I} \not\models C(a)$. By Definition 4.17 and since $osn^{a,\mathcal{A}}$ is splittable, there exists a homomorphism θ from $ABox(osn^{a,\mathcal{A}})$ to $\mathcal{A}^{isl}$ of ISL_a, and vice versa. Since there exists a θ from $ABox(osn^{a,\mathcal{A}})$ to $\mathcal{A}^{isl}$, we know that $\mathcal{I}_\theta \models ISL_a$. From $\mathcal{I}_\theta \not\models C(a)$ (unchanged concept sets for domain elements), we can conclude $ISL_a \not\models C(a)$. □

One-step nodes cannot only be used as a compact representation for instance checking and retrieval, but also for the problem of relation checking. Since all direct neighbors

are represented as pseudo node successors, we can easily define a special completeness criterion for relation entailment in Proposition 4.5. In addition, such a criterion can be important as a filter when answering conjunctive queries [GHLS07].

Proposition 4.5 (One-Step Node Entailment is Complete for Relation Checking)**:**
Given

- an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$,
- a role description R,
- two individuals $a_1 \in NInd(\mathcal{A})$ and $a_2 \in NInd(\mathcal{A})$,
- a one-step node $osn^{a_1,\mathcal{A}}$ for individual a_1, and
- a one-step node $osn^{a_2,\mathcal{A}}$ for individual a_2,

$\mathcal{O} \models R(a_1, a_2) \implies (osn^{a_1,\mathcal{A}} \models \exists R.\top(a_1) \wedge osn^{a_2,\mathcal{A}} \models \exists R^-.\top(a_2))$.

Proof of Proposition 4.5. Easy to see, since all role assertions for a_1 and its direct neighbors are represented as a pseudo node successor. □

In Example 4.4 we show two one step nodes and their intuition.

Example 4.4 (Example for One-Step Node Relation Entailment)**:**
Given the two one-step nodes $osn^{mae,\mathcal{A}_{Ex4.1}}$ and $osn^{c5,\mathcal{A}_{Ex4.1}}$, as defined in

$$\begin{aligned} osn^{mae,\mathcal{A}_{Ex4.1}} =& \langle \{Professor\}, \emptyset, \{\langle \{headOf\}, \{Department\}\rangle, \\ & \langle \{teaches\}, \{UndergraduateCourse\}\rangle\}\rangle \\ osn^{c5,\mathcal{A}_{Ex4.1}} =& \langle \{Student\}, \emptyset, \\ & \{\langle \{takes^-\}, \{Student\}\rangle, \langle \{teaches^-\}, \{Professor\}\rangle\}\rangle, \end{aligned}$$

we have for instance that individual *mae* must have at least one *teaches*-successor and individual $c5$ cannot have any named *teaches*-successor (but has a *teaches*-predecessor).

We come back to the original intention of one-step nodes, the use as integrity check for individual island similarity. In Proposition 4.6, we show that two individual islands are only similar to each other if the one-step nodes for their individuals are similar.

Definition 4.25 (Similar One-Step Nodes)**:**
Given two one-step nodes $osn^{a_1,\mathcal{A}} = \langle \mathbf{rootconset}_1, \mathbf{reflset}_1, \mathbf{pnsset}_1 \rangle$ and $osn^{a_2,\mathcal{A}} = \langle \mathbf{rootconset}_2, \mathbf{reflset}_2, \mathbf{pnsset}_2 \rangle$, we say that $osn^{a_1,\mathcal{A}}$ is *similar to* $osn^{a_2,\mathcal{A}}$ if and only if $\mathbf{rootconset}_1 = \mathbf{rootconset}_2$, $\mathbf{reflset}_1 = \mathbf{reflset}_2$, and $\mathbf{pnsset}_1 = \mathbf{pnsset}_2$.

Proposition 4.6 (Similar Islands Integrity with One-Step Nodes)**:**
Given an individual island $ISL_{a_1} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_1^{isl}, a_1 \rangle$ and an individual island $ISL_{a_2} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2^{isl}, a_2 \rangle$, plus the one-step nodes $osn^{a_1,\mathcal{A}} = \langle \mathbf{rootconset}_1, \mathbf{reflset}_1, \mathbf{pnsset}_1 \rangle$ and $osn^{a_2,\mathcal{A}} = \langle \mathbf{rootconset}_2, \mathbf{reflset}_2, \mathbf{pnsset}_2 \rangle$, we have that ISL_{a_1} and ISL_{a_2} are similar to each other only if $osn^{a_1,\mathcal{A}}$ and $osn^{a_2,\mathcal{A}}$ are similar to each other.

Proof of Proposition 4.6. If ISL_{a_1} and ISL_{a_2} are similar to each other, then there exists a homomorphism θ from ISL_{a_1} to ISL_{a_2}, and vice versa. By Definition 4.12, it is easy to see that we have $\mathbf{rootconset}_1 = \mathbf{rootconset}_2$, $\mathbf{reflset}_1 = \mathbf{reflset}_2$, and $\mathbf{pnsset}_1 = \mathbf{pnsset}_2$. □

Given Proposition 4.6, we can avoid computing homomorphisms for pairs of individual islands (individuals), if their one-step nodes are not similar.

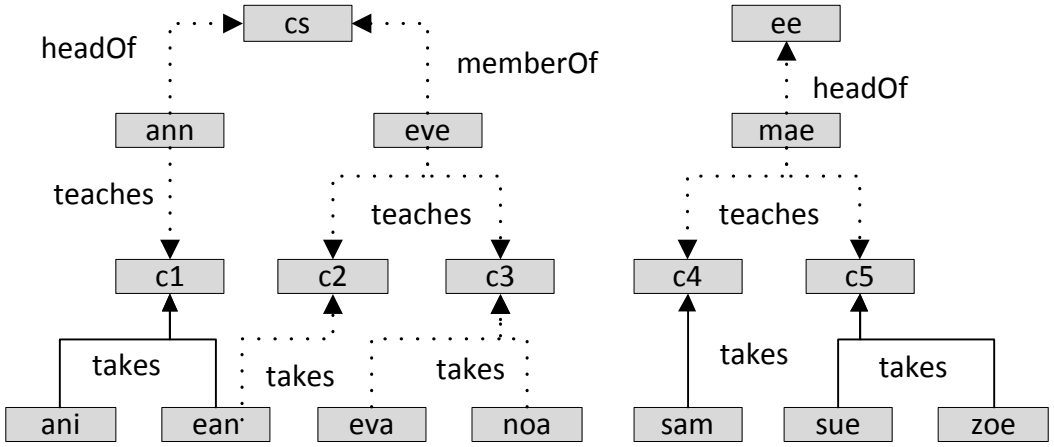
4.4 Reasoning Optimization

In the following, we look at an example to discuss the optimization of instance checking and instance retrieval by the techniques introduced in this chapter.

Example 4.5 (Example Ontology for Island Reasoning)**:**
The example ontology $\mathcal{O}_{Ex4.5} = \langle \mathcal{T}_{Ex4.5}, \mathcal{R}_{Ex4.5}, \mathcal{A}_{Ex4.5} \rangle$ is defined as follows

$$
\begin{aligned}
\mathcal{T}_{Ex4.5} = \{ & \\
& Chair \equiv \exists headOf.Department, Student \equiv \exists takes.Course, \\
& GraduateStudent \sqsubseteq \forall takes.GraduateCourse, \\
& UndergraduateCourse \sqcap Chair \sqsubseteq \bot, GraduateCourse \sqcap Chair \sqsubseteq \bot, \\
& UndergraduateCourse \sqsubseteq Course, GraduateCourse \sqsubseteq Course, \\
& Student \sqcap Chair \sqsubseteq \bot, \top \sqsubseteq \forall takes.Course \\
\} & \\
\mathcal{R}_{Ex4.5} = \{ & headOf \sqsubseteq memberOf, teaches \equiv isTaughtBy^{-} \} \\
\mathcal{A}_{Ex4.5} = \{ & \\
& Department(cs), Department(ee), \\
& Professor(ann), Professor(eve), Professor(mae), \\
& UndergraduateCourse(c1), UndergraduateCourse(c4), \\
& UndergraduateCourse(c5), \\
& GraduateCourse(c2), GraduateCourse(c3), \\
& Student(ani), Student(ean), Student(eva), Student(noa), \\
& Student(sam), Student(sue), Student(zoe), \\
& headOf(ann, cs), memberOf(eve, cs), headOf(mae, ee), \\
& teaches(ann, c1), teaches(eve, c2), teaches(eve, c3), \\
& teaches(mae, c4), teaches(mae, c5), \\
& takes(ani, c1), takes(ean, c1), takes(ean, c2), takes(eva, c3), \\
& takes(noa, c3), takes(sam, c4), takes(sue, c5), takes(zoe, c5) \\
& \}.
\end{aligned}
$$

Figure 4.4 Individual relationships and splittability for Example 4.5



The extended $\forall$-info structure for $\mathcal{T}_{Ex4.5}$ and $\mathcal{R}_{Ex4.5}$ is:

$$extinfo^{\forall}_{\mathcal{T}_{Ex4.5},\mathcal{R}_{Ex4.5}}(R) = \begin{cases} \{\neg Department\} & \text{if } R = headOf, \\ \{\neg Course, Course, GraduateCourse\} & \text{if } R = takes, \\ \emptyset & \text{otherwise.} \end{cases}$$

The relationships among individuals of $\mathcal{A}_{Ex4.5}$ are depicted in Figure 4.4. Please note that only role assertions are used to build the graph, since we only want to emphasize the relationship between the ABox individuals. $\mathcal{SHI}$-splittable role assertions are indicated with a dashed line. For instance, the role assertion $takes(ani, c1)$ is not $\mathcal{SHI}$-splittable because the concept description $GraduateCourse$ can be propagated via role description $takes$. Please note that all these role assertions would be $\mathcal{SHI}$-splittable if we had a disjointness axiom for $GraduateCourse$ and $UndergraduateCourse$. However, to show the behavior of reasoning in case of $\mathcal{SHI}$-unsplittability, we omitted the disjointness axiom here.

4.4.1 Instance Checking

For instance checking, we are given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, an atomic concept description C, and an individual $a \in NInd(\mathcal{A})$, and we would like to find out, whether $\mathcal{O} \models C(a)$. The process of instance checking is done in two steps. First, we take the one-step node $osn^{a,\mathcal{A}}$ of individual a and check, whether $osn^{a,\mathcal{A}} \models C(a)$. If yes, then we are done, since by Lemma 4.5, we know that one-step nodes are sound for instance checking with respect to the input ontology $\mathcal{O}$. If $osn^{a,\mathcal{A}} \not\models C(a)$, then we distinguish two cases. First, if $osn^{a,\mathcal{A}}$ is splittable, then we know, by Lemma 4.6, that we have $\mathcal{O} \not\models C(a)$. Otherwise, if $osn^{a,\mathcal{A}}$

is not splittable, then we load the individual island ISL_a for individual a and perform instance checking over ISL_a.

As an example for instance checking, we would like to check, whether the individual ann is an instance of concept description $Chair$ with respect to the ontology $\mathcal{O}_{Ex4.5}$. The one-step node $osn^{ann,\mathcal{A}_{Ex4.5}}$ is defined as follows:

$$\begin{aligned} osn^{ann,\mathcal{A}_{Ex4.5}} =& \langle \{Professor\}, \emptyset, \{\langle \{headOf\}, \{Department\}\rangle, \\ & \langle \{teaches\}, \{UndergraduateCourse\}\rangle\}\rangle. \end{aligned}$$

One possible one-step node realization of $osn^{ann,\mathcal{A}_{Ex4.5}}$ is

$$ABox(osn^{ann,\mathcal{A}_{Ex4.5}}) = \{Professor(ann), headOf(ann, a_1), teaches(ann, a_2)\}.$$

It is easy to see that we have $\langle \mathcal{T}, \mathcal{R}, ABox(osn^{ann,\mathcal{A}_{Ex4.5}})\rangle \models Chair(ann)$, and thus we have $ABox(osn^{ann,\mathcal{A}_{Ex4.5}}) \models_{\mathcal{T}_{Ex4.5},\mathcal{R}_{Ex4.5}} Chair(ann)$ and by soundness of one-step node reasoning $\mathcal{O}_{Ex4.5} \models Chair(ann)$.

As a second example for instance checking, we would like to check, whether the individual $c1$ is an instance of concept description $Chair$ with respect to the ontology $\mathcal{O}_{Ex4.5}$. The one-step node $osn^{c1,\mathcal{A}_{Ex4.5}}$ is as follows:

$$\begin{aligned} osn^{c1,\mathcal{A}_{Ex4.5}} =& \langle \{UndergraduateCourse\}, \emptyset, \{\langle \{teaches^-\}, \{Professor\}\rangle, \\ & \langle \{takes^-\}, \{Student\}\rangle\}\rangle. \end{aligned}$$

One possible one-step node realization of $osn^{c1,\mathcal{A}_{Ex4.5}}$ is

$$ABox(osn^{c1,\mathcal{A}_{Ex4.5}}) = \{UndergraduateCourse(c1), teaches(a_1, c1,), takes(a_2, c1)\}.$$

It is easy to see that we have $\langle \mathcal{T}, \mathcal{R}, ABox(osn^{c1,\mathcal{A}_{Ex4.5}})\rangle \not\models Chair(c1)$. In this case, the one-step node does not indicate entailment, and since $osn^{c1,\mathcal{A}_{Ex4.5}}$ is not a splittable one-step node, we should refer to the individual island of individual $c1$. However, another simple instance check can help us to avoid using the individual island here. It is easy to see that we have $\langle \mathcal{T}, \mathcal{R}, ABox(osn^{c1,\mathcal{A}_{Ex4.5}})\rangle \models \neg Chair(c1)$, by disjointness of $UndergraduateCourse$ and $Chair$. And this means, by Lemma 4.5, that we have $\mathcal{O}_{Ex4.5} \models \neg Chair(c1)$. Thus, in some cases, the „negated instance check“ for one-step nodes can also help us to avoid performing reasoning on (more complex) individual islands. However, if the negated instance check fails, and the one-step node is unsplittable, then we really have to use sound and complete individual islands.

4.4.2 Instance Retrieval

In the following, we discuss instance retrieval optimization over ontologies. This is a direct extension of instance checking, by using one-step node similarity in addition. The first naive approach would be to apply instance checking techniques to each named individual

in the ABox. For ontology $\mathcal{O}_{Ex4.5}$, we would have to perform 17 instance checks in that case. However, in Section 4.3, we have introduced the notion of one-step node similarity. The idea is that similar one-step nodes entail the same set of concept descriptions for the named root individual. Given the set of all one-step nodes for an input ontology, we can reduce the number of instance checks.

For example, assume that we would like to perform instance retrieval for the concept description $Chair$ with respect to ontology $\mathcal{O}_{Ex4.5}$. First, we retrieve the one-step node for each named individual in $\mathcal{A}_{Ex4.5}$. The resulting one-step nodes are shown below:

$$osn^{ani,\mathcal{A}_{Ex4.5}} = osn^{sam,\mathcal{A}_{Ex4.5}} = osn^{sue,\mathcal{A}_{Ex4.5}} = osn^{zoe,\mathcal{A}_{Ex4.5}} = \langle \{Student\}, \emptyset, \{\langle \{takes\}, \{UndergraduateCourse\}\rangle\}\rangle$$

$$osn^{ean,\mathcal{A}_{Ex4.5}} = \langle \{Student\}, \emptyset, \{\langle \{takes\}, \{UndergraduateCourse\}\rangle, \langle \{takes\}, \{GraduateCourse\}\rangle\}\rangle$$

$$osn^{eva,\mathcal{A}_{Ex4.5}} = osn^{noa,\mathcal{A}_{Ex4.5}} = \langle \{Student\}, \emptyset, \{\langle \{takes\}, \{GraduateCourse\}\rangle\}\rangle$$

$$osn^{c1,\mathcal{A}_{Ex4.5}} = osn^{c4,\mathcal{A}_{Ex4.5}} = osn^{c5,\mathcal{A}_{Ex4.5}} = \langle \{UndergraduateCourse\}, \emptyset, \{\langle \{teaches^-\}, \{Professor\}\rangle, \langle \{takes^-\}, \{Student\}\rangle\}\rangle$$

$$osn^{c2,\mathcal{A}_{Ex4.5}} = osn^{c3,\mathcal{A}_{Ex4.5}} = \langle \{GraduateCourse\}, \emptyset, \{\langle \{teaches^-\}, \{Professor\}\rangle, \langle \{takes^-\}, \{Student\}\rangle\}\rangle$$

$$osn^{ann,\mathcal{A}_{Ex4.5}} = osn^{mae,\mathcal{A}_{Ex4.5}} = \langle \{Professor\}, \emptyset, \{\langle \{headOf\}, \{Department\}\rangle, \langle \{teaches^-\}, \{UndergraduateCourse\}\rangle\}\rangle$$

$$osn^{eve,\mathcal{A}_{Ex4.5}} = \langle \{Professor\}, \emptyset, \{\langle \{memberOf\}, \{Department\}\rangle, \langle \{teaches^-\}, \{GraduateCourse\}\rangle\}\rangle$$

$$osn^{cs,\mathcal{A}_{Ex4.5}} = \langle \{Department\}, \emptyset, \{\langle \{headOf^-\}, \{Professor\}\rangle, \langle \{memberOf^-\}, \{Professor\}\rangle\}\rangle$$

$$osn^{ee,\mathcal{A}_{Ex4.5}} = \langle \{Department\}, \emptyset, \{\langle \{headOf^-\}, \{Professor\}\rangle\}\rangle.$$

Instead of 17 instance checks for 17 named individuals, we are left with 9 instance checks over 9 similar one-step nodes. For ontologies with a larger assertional part, similarity of one-step nodes reduces the number of instance checks usually by orders of magnitudes. These results are shown in Chapter 6.

By performing instance checks for concept description $Chair$ over the 9 one-step nodes, we can conclude that individual ann and individual mae are instances of $Chair$. Additional instance checks for concept description $\neg Chair$ yields that $c1$, $c2$, $c3$, $c4$, $c5$, ani, ean, eva, noa, sam, sue and zoe are instances of concept description $\neg Chair$, and therefore are not instances of concept description $Chair$ if the input ontology is consistent. After one-step node retrieval, we are left to check three individuals for being an instance of concept description $Chair$, or not: cs, ee and eve. Usually, one would have to perform instance checks over the three individual islands. However, since the corresponding one-step nodes for these three individuals are splittable, we do not need to do any further checks, since the one-step nodes are already sound and complete for reasoning in $\mathcal{O}_{Ex4.5}$.

4.5 Concluding Remarks

Given our results about ABox modularizations from Chapter 3, we have defined an optimized way to perform instance checking for a given individual and a given atomic concept description. Given an input individual, we extract a subset of ABox assertions, such that the subset is worst-case sufficient to ensure sound and complete instance checking.

Tableau-based description logic reasoning systems are usually implemented based on efficient main memory data structures. Our island extraction can release the main memory burden for solving the instance checking problem. The average size of individual islands is evaluated in Chapter 6.

In addition, we have discussed how to optimize instance retrieval over islands. We have introduced a similarity measure, in order to reduce the number of instance checks necessary to perform instance retrieval.

In order to determine individual island similarity, we have introduced a data structure called one-step nodes. These one-step nodes can be used as a kind of proxy to answer queries immediately in a sound (and in many cases complete) way, as well.

Chapter 5: Updates

In order to support stream-like processing of ontologies, incremental reasoning and updatable data structures will become increasingly important in the future. In the literature, there exist different semantics for updates over description logic ontologies. This is similar to research results in the belief revision literature, see for instance [Neb94], where the authors differentiate between belief bases and belief sets. For ontologies, there exists a lot of research about semantic updates, addressing directly the underlying model(s). Here, we focus on so-called syntactic updates in the style of [HWPS06]. Syntactic updates do not address the model(s) directly, but only the assertions and axioms describing the model(s).

In Section 5.1, we formalize a set of syntactic update operations over ontologies. In particular, we define operations to add and retract assertions from the TBox, RBox, and ABox.

In Section 5.2, we define an abstract split decision system, which helps us to decide $\mathcal{SHI}$-splittability with respect to ontologies. This abstract split decision system is motivated by the data structures introduced for more efficient instance checking and instance retrieval in Chapter 4.

In Section 5.3, we define updatable data structures for reasoning over updatable ontologies. In detail, we introduce sound structures for TBox classification and TBox disjointness, complete structures for $\forall$-propagation management and role management. These structures are gradually adjusted under syntactic ontology updates.

Based on the updatable structures from Section 5.3, we propose a concrete implementation of a split decision system in Section 5.4.

In Section 5.5, we use the above results to propose an updatable one-step node map and an updatable island map.

This chapter is concluded with Section 5.6.

5.1 Syntactic Update Definitions

In the following, we define updates on ontologies. We distinguish updates of axioms in TBoxes, RBoxes, and ABoxes, and define an update for each kind of axiom by using update functions over ontologies. The input of an update function is an ontology, and the output is a modified ontology, such that either an assertion is added or retracted. The intuition of these update definitions is not hard to understand. However, in order to to ease proofs and implementation of our algorithms, we provide rigorous definitions at this point.

The definition of updates as a function is convenient, since we would like to compose several updates and could even discuss histories, i.e., past states of an ontology. We start with the addition and retraction of ABox assertion axioms in Definition 5.1, 5.2 and 5.3.

Definition 5.1 (Syntactic ABox Concept Assertion Addition, Retraction and Update)**:** Given a concept assertion $C_u(a_u)$, a *syntactic ABox concept assertion update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that

- $upd = upd \parallel\!+\!C_u(a_u)\parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic ABox concept assertion addition*), defined as $upd \parallel\!+\!C_u(a_u)\parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle) = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{C_u(a_u)\}\rangle$, or
- $upd = upd \parallel\!-\!C_u(a_u)\parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic ABox concept assertion retraction*), defined as $upd \parallel\!-\!C_u(a_u)\parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle) = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \setminus \{C_u(a_u)\}\rangle$.

In Definition 5.2 and below, we use *upd* as a placeholder for any update function.

Definition 5.2 (Syntactic ABox Role Assertion Addition, Retraction and Update)**:** Given a role assertion $R_u(a_{u1}, a_{u2})$, a *syntactic ABox role assertion update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that

- $upd = upd \parallel\!+\!R_u(a_{u1}, a_{u2})\parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic ABox role assertion addition*), defined as $upd \parallel\!+\!R_u(a_{u1}, a_{u2})\parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle) = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{R_u(a_{u1}, a_{u2})\}\rangle$, or
- $upd = upd \parallel\!-\!R_u(a_{u1}, a_{u2})\parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic ABox role assertion retraction*), defined as $upd \parallel\!-\!R_u(a_{u1}, a_{u2})\parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle) = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \setminus \{R_u(a_{u1}, a_{u2})\}\rangle$.

Definition 5.3 (Syntactic ABox Update)**:**
A *syntactic ABox update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that upd is one of $upd \parallel\!+\!C_u(a_u)\parallel$, $upd \parallel\!-\!C_u(a_u)\parallel$, $upd \parallel\!+\!R_u(a_{u1}, a_{u2})\parallel$, or $upd \parallel\!-\!R_u(a_{u1}, a_{u2})\parallel$.

In Example 5.1, we show the effect of syntactic ABox updates in practice.

Example 5.1 (Syntactic ABox Update)**:**
Assume that we are given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle$, with

$$\mathcal{A} = \{UndergraduateStudent(ani), takes(ani, c1), Course(c1)\}.$$

If *ani* graduates, i.e. we have to represent that she is not an $UndergraduateStudent$ anymore, but a $GraduateStudent$, we can perform the following two syntactic ABox updates on $\mathcal{O}$: $upd \parallel\!-\!UndergraduateStudent(ani)\parallel$ and $upd \parallel\!+\!GraduateStudent(ani)\parallel$. The ontology obtained by applying both updates is $\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_2\rangle =$

$$upd \parallel\!+\!GraduateStudent(ani)\parallel (upd \parallel\!-\!UndergraduateStudent(ani)\parallel (\mathcal{O})),$$

where

$$\mathcal{A}_2 = \{GraduateStudent(ani), takes(ani, c1), Course(c1)\}.$$

Next, we define update functions on the RBox of an ontology. In Definition 5.4, role inclusion updates are introduced and in Definition 5.5 we define role transitivity updates. These two updates are summarized as syntactic RBox updates in Definition 5.6.

Definition 5.4 (Syntactic RBox Role Inclusion Addition, Retraction and Update)**:**
Given a role inclusion $R_{u1} \sqsubseteq R_{u2}$, a *syntactic RBox role inclusion update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that

- $upd = upd \parallel\!+\! R_{u1} \sqsubseteq R_{u2} \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic RBox role inclusion addition*), defined as $upd \parallel\!+\! R_{u1} \sqsubseteq R_{u2} \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T}, \mathcal{R} \cup \{R_{u1} \sqsubseteq R_{u2}\}, \mathcal{A} \rangle$, or
- $upd = upd \parallel\!-\! R_{u1} \sqsubseteq R_{u2} \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic RBox role inclusion retraction*), defined as $upd \parallel\!-\! R_{u1} \sqsubseteq R_{u2} \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T}, \mathcal{R} \setminus \{R_{u1} \sqsubseteq R_{u2}\}, \mathcal{A} \rangle$.

Definition 5.5 (Syntactic RBox Role Transitivity Addition, Retraction and Update)**:**
A *syntactic RBox role transitivity update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that

- $upd = upd \parallel\!+\! Trans(R_u) \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic RBox role transitivity addition*), defined as $upd \parallel\!+\! Trans(R_u) \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T}, \mathcal{R} \cup \{Trans(R_u)\}, \mathcal{A} \rangle$, or
- $upd = upd \parallel\!-\! Trans(R_u) \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic RBox role transitivity retraction*), defined as $upd \parallel\!-\! Trans(R_u) \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T}, \mathcal{R} \setminus \{Trans(R_u)\}, \mathcal{A} \rangle$.

Definition 5.6 (Syntactic RBox Update)**:**
A *syntactic RBox update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that upd is one of: $upd \parallel\!+\! R_{u1} \sqsubseteq R_{u2} \parallel$, $upd \parallel\!-\! R_{u1} \sqsubseteq R_{u2} \parallel$, $upd \parallel\!+\! Trans(R_u) \parallel$, or $upd \parallel\!-\! Trans(R_u) \parallel$.

We proceed with the formal definition of update functions on the TBox of an ontology. In Definition 5.7, TBox concept inclusion updates are introduced.

Definition 5.7 (Syntactic TBox Concept Inclusion Addition, Retraction and Update)**:**
Given a general concept inclusion $C_{u1} \sqsubseteq C_{u2}$, a *syntactic TBox concept inclusion update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that

- $upd = upd \parallel\!+\! C_{u1} \sqsubseteq C_{u2} \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic TBox concept inclusion addition*), defined as $upd \parallel\!+\! C_{u1} \sqsubseteq C_{u2} \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T} \cup \{C_{u1} \sqsubseteq C_{u2}\}, \mathcal{R}, \mathcal{A} \rangle$, or
- $upd = upd \parallel\!-\! C_{u1} \sqsubseteq C_{u2} \parallel : \mathbf{SO} \rightarrow \mathbf{SO}$ (called *syntactic TBox concept inclusion retraction*), defined as $upd \parallel\!-\! C_{u1} \sqsubseteq C_{u2} \parallel (\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle) = \langle \mathcal{T} \setminus \{C_{u1} \sqsubseteq C_{u2}\}, \mathcal{R}, \mathcal{A} \rangle$.

Definition 5.8 (Syntactic TBox Update)**:**
A *syntactic TBox update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that the update function upd is either $upd \parallel\!+\! C_{u1} \sqsubseteq C_{u2} \parallel$ or $upd \parallel\!-\! C_{u1} \sqsubseteq C_{u2} \parallel$.

In Definition 5.9, we combine all updates to syntactic ontology updates.

Definition 5.9 (Syntactic Ontology Update)**:**
A *syntactic ontology update* is a function $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, such that upd is a syntactic TBox update, a syntactic RBox update, or a syntactic ABox update.

Definition 5.10 (Applicability of Syntactic Ontology Updates)**:**
A syntactic ontology update $upd : \mathbf{SO} \rightarrow \mathbf{SO}$ is *applicable* to an ontology $\mathcal{O}$ if $\mathcal{O} \neq upd(\mathcal{O})$.

By Definition 5.10, we only allow updates which really change the ontology. For instance, removal of an axiom is only allowed if the axiom is present in $\mathcal{O}$, and addition of an axiom is only allowed if the axiom is not yet present in $\mathcal{O}$. In the following we only consider applicable syntactic ontology updates.

Given the syntactic ontology updates introduced above we define a so-called ontology state. Informally speaking, an ontology state describes a snapshot of an ontology, together with the information how that snapshot was obtained from an empty ontology, i.e. by a history of syntactic ontology updates. Although we do not make explicit use of the history below and hence could define our algorithms directly over ontologies, we think that the abstraction towards an ontology state points out that: in general we would like touch/look at the assertions in an ontology anymore, but define (general) data structures only based on an initial (empty) ontology and the update functions. This makes the design of incremental algorithms easier.

Definition 5.11 (Ontology State)**:**
An *ontology state* $\mathcal{OS} = \langle \mathcal{O}, history \rangle$ is inductively defined as follows:

1. The pair $\langle \mathcal{O}, history \rangle$ is an *initial* ontology state if $\mathcal{O} = \langle \emptyset, \emptyset, \emptyset \rangle$ and $history = \square$ is an empty list of syntactic ontology updates.
2. The pair $\langle upd(\mathcal{O}), upd \circ history \rangle$ is an ontology state if
   - $\langle \mathcal{O}, history \rangle$ is an ontology state and
   - upd is a syntactic ontology update applicable to $\mathcal{O}$.

In Definition 5.12, we extend syntactic update functions from ontologies to ontology states.

Definition 5.12 (Ontology State Update)**:**
Given an ontology state $\langle \mathcal{O}, history \rangle$ and a syntactic ontology update upd, an *ontology state update*, denoted $\langle \mathcal{O}, history \rangle \uparrow upd$, is defined as

$$\langle \mathcal{O}, history \rangle \uparrow upd = \langle upd(\mathcal{O}), upd \circ history \rangle.$$

In Example 5.2, we show two examples for ontology states.

Example 5.2 (Ontology State)**:**
Given the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, where

$$\begin{aligned} \mathcal{T} =& \{\top \sqsubseteq \forall memberOf^{-}.Person\}, \\ \mathcal{R} =& \{headOf \sqsubseteq memberOf\}, \\ \mathcal{A} =& \{headOf(ann, cs), Department(cs)\}, \end{aligned}$$

the following are ontology states:

- $\langle \mathcal{O}, history_1 \rangle$, such that

$$\begin{aligned} history_1 = & upd \, \|{+}Department(cs)\| \\ & \circ \, upd \, \|{+}headOf \sqsubseteq memberOf\| \\ & \circ \, upd \, \big\|{+}\top \sqsubseteq \forall memberOf^{-}.Person\big\| \\ & \circ \, upd \, \|{+}headOf(ann, cs)\| \\ & \circ \, \square \end{aligned}$$

- $\langle \mathcal{O}, history_2 \rangle$, such that

$$\begin{aligned} history_2 = & upd \, \|{+}Department(cs)\| \\ & \circ \, upd \, \|{-}Professor(ann)\| \\ & \circ \, upd \, \|{+}headOf \sqsubseteq memberOf\| \\ & \circ \, upd \, \big\|{+}\top \sqsubseteq \forall memberOf^{-}.Person\big\| \\ & \circ \, upd \, \|{+}headOf(ann, cs)\| \\ & \circ \, upd \, \|{+}Professor(ann)\| \\ & \circ \, \square. \end{aligned}$$

5.2 Abstract Split Decision System

In Chapter 4, we identified a set of data structures for optimized instance checking and retrieval. The two main structures are

- *one-step node maps* for fast query answering/individual island similarity measure and
- *individual island maps* for fast main memory instance checking.

In addition, we recapitulate auxiliary structures for the determination of $\mathcal{SHI}$-splittability as follows:

- TBox classification structure,
- TBox disjointness structure,
- $\forall$-info structure,
- RBox classification structure, and
- RBox transitivity structure.

In the following, we explicitly define these data structures over ontology states and then propose updatable instantiations. We start with the TBox classification structure. The definition of $\mathcal{SHI}$-splittable ABox splits, in Definition 3.18, shows that we actually need a sound TBox classification structure, i.e. it is only important that we can show that all used concept subsumptions really hold in an ontology state. If we disregard some subsumptions, then we only miss additional ABox splits. Therefore, and since the computation of a sound TBox classification can be done more efficiently, we focus on the computation of a sound TBox classification only.

Definition 5.13 (Sound TBox Classification Structure)**:**
Given an ontology state $\mathcal{OS} = \langle \mathcal{O}, \mathit{history} \rangle$, a set $\alpha^{stcs}_{\mathcal{OS}} \subseteq \mathbf{GCIs}$ is a *sound TBox classification structure for* $\mathcal{OS}$ if

$$C_1 \sqsubseteq C_2 \in \alpha^{stcs}_{\mathcal{OS}} \implies O \vDash C_1 \sqsubseteq C_2.$$

We only need soundness for the TBox disjointness structure as well. If we disregard some entailed TBox disjointness axioms, we might miss some valid ABox splits. However, for the sake of efficiency, it might be worth in practice. We investigate this assumption below. Here, we just state that the TBox disjointness structure should be sound. Of course, to detect more ABox splits, in an actual implementation one could enforce completeness as well.

Definition 5.14 (Sound TBox Disjointness Structure)**:**
Given an ontology state $\mathcal{OS} = \langle \mathcal{O}, \mathit{history} \rangle$, a set $\alpha^{stds}_{\mathcal{OS}} \subseteq \mathbf{Con} \times \mathbf{Con}$ is a *sound TBox disjointness structure for* $\mathcal{OS}$ if $(C_1, C_2) \in \alpha^{stds}_{\mathcal{OS}} \implies \mathcal{O} \vDash C_1 \oslash C_2$.

In Chapter 4, the basic structure for managing propagations in the description logic $\mathcal{ALC}$ is the $\forall$-info structure. In opposition to the TBox structures above, here completeness is important. If the $\forall$-info structure has more entries than necessary, then we might only miss some ABox splits again because we apply the check for too many role assertions in the ABox. A perfect $\forall$-info structure would be sound and complete (and easy to compute) . But in order to follow the style of sound or complete data structures, we focus on complete $\forall$-info structures.

Definition 5.15 (Complete $\forall$-info Structure)**:**
Given an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, \mathit{history} \rangle$, a set $\alpha^{cfis}_{\mathcal{OS}} \subseteq \mathbf{Rol} \times \mathbf{Con}$ is a *complete* $\forall$*-info structure for* $\mathcal{OS}$ if $\forall R.C \in \mathit{clos}(\mathcal{T}) \implies (R, C) \in \alpha^{cfis}_{\mathcal{OS}}$.

Since we only want to have a complete $\forall$-info structure, we can also release soundness for our definitions of RBox classification and transitivity, which are used for the extension to the description logic $\mathcal{SHI}$.

Definition 5.16 (Complete RBox Classification Structure)**:**
Given an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, \mathit{history} \rangle$, a set $\alpha^{crcs}_{\mathcal{OS}} \subseteq \mathbf{Rol} \times \mathbf{Rol}$ is a *complete RBox classification structure for* $\mathcal{OS}$ if $\underline{rc}^{\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle} \subseteq \alpha^{crcs}_{\mathcal{OS}}$.

Definition 5.17 (Complete RBox Transitivity Structure)**:**
Given an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, \mathit{history} \rangle$, a set $\alpha^{crts}_{\mathcal{OS}} \subseteq \mathbf{Rol} \times \mathbf{Rol}$ is a *complete RBox transitivity structure for* $\mathcal{OS}$ if $\underline{rtc}^{\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle} \subseteq \alpha^{crts}_{\mathcal{OS}}$.

Last, but not least, we define the main updatable structures for split management in ontology states: one-step node maps and island maps.

Definition 5.18 (Sound and Complete One-Step Node Map Structure)**:**
Given an ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle$, a function $f_{\mathcal{OS}} : \mathbf{IN} \to \mathbf{OSN}$ is a *sound and complete one-step node map structure for* $\mathcal{OS}$ if $a \in NInd(\mathcal{A}) \iff f_{\mathcal{OS}}(a) = osn^{a,\mathcal{A}}$.

The complete split dependency structure defined in Definition 5.19 below is an intermediate structure which manages all dependencies between individuals in an ontology, i.e. a pair of two individuals is in the split dependency structure if these two individuals are related by a $\mathcal{SHI}$-unsplittable role assertion in the ontology state.

Definition 5.19 (Complete Split Dependency Structure)**:**
Given an ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle$, a set $\mathbf{S}_{\mathcal{OS}} \subseteq \mathbf{IN} \times \mathbf{IN}$ is a *complete split dependency structure for* $\mathcal{OS}$ if $\forall a_1, a_2.((\exists R \in \mathbf{Rol}.(R(a_1,a_2) \in \mathcal{A} \wedge R(a_1,a_2)$ is not $\mathcal{SHI}$-splittable with respect to $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle$ $)) \implies (a_1,a_2) \in \mathbf{S}_{\mathcal{OS}})$.

Definition 5.20 (Sound and Complete Island Map Structure)**:**
Given an ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle$, a function $f_{\mathcal{OS}} : \mathbf{IN} \to \wp(\mathbf{IN})$ is a *sound and complete island map structure for* $\mathcal{OS}$ if for each named individual $a \in NInd(\mathcal{A})$, $\langle\mathcal{T},\mathcal{R}, LoadAs(f_{\mathcal{OS}}(a)), a\rangle$ is an individual island for $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle$, where

$$\begin{aligned} LoadAs(\mathbf{inds}) =& \{C(a) \in \mathcal{A} \mid a \in \mathbf{inds}\} \cup \\ & \{R(a_1,a_2) \in \mathcal{A} \mid \{a_1,a_2\} \subseteq \mathbf{inds}\}. \end{aligned}$$

In Chapter 3, we have defined $\mathcal{SHI}$-splittability with respect to ontologies. In this chapter, we have introduced abstractions, by allowing for incompleteness and unsoundness, for the sake of efficiency. In the following, we define $\mathcal{SHI}$-splittability with respect to these incomplete or unsound data structures, and show that each split in our abstraction is also a split with respect to the ontology. First, we define an abstract split decision system, which is constructed from the basic abstract data structures introduced above.

Definition 5.21 (Abstract Split Decision System)**:**
Given

- an ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle$,
- a sound TBox classification structure $\alpha^{stcs}_{\mathcal{OS}}$ for $\mathcal{OS}$,
- a sound TBox disjointness structure $\alpha^{stds}_{\mathcal{OS}}$ for $\mathcal{OS}$,
- a complete $\forall$-info structure $\alpha^{cfis}_{\mathcal{OS}}$ for,
- a complete RBox classification structure $\alpha^{crcs}_{\mathcal{OS}}$ for $\mathcal{OS}$, and
- a complete RBox transitivity structure $\alpha^{crts}_{\mathcal{OS}}$ for $\mathcal{OS}$,

the tuple $asds_{\mathcal{OS}} = \langle\alpha^{stcs}_{\mathcal{OS}}, \alpha^{stds}_{\mathcal{OS}}, \alpha^{cfis}_{\mathcal{OS}}, \alpha^{crcs}_{\mathcal{OS}}, \alpha^{crts}_{\mathcal{OS}}, \mathcal{A}\rangle$ is called *abstract split decision system for* $\mathcal{OS}$.

For an abstract split decision system, we define split-safe role assertions in Definition 5.22. The intuition is that any split-safe role assertion for an ontology state is for sure $\mathcal{SHI}$-splittable with respect to the underlying ontology. In addition, there might exist other $\mathcal{SHI}$-splittable role assertions which are not identified as split-safe, due to unsoundness or incompleteness. Definition 5.22, follows the pattern introduced in Definition 3.18.

Definition 5.22 (Split-Safe Role Assertion)**:**
Given an abstract split decision system $asds_{\mathcal{OS}} = \langle \alpha^{stcs}_{\mathcal{OS}}, \alpha^{stds}_{\mathcal{OS}}, \alpha^{cfis}_{\mathcal{OS}}, \alpha^{crcs}_{\mathcal{OS}}, \alpha^{crts}_{\mathcal{OS}}, \mathcal{A} \rangle$ for an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, history \rangle$ and a role assertion $R(a_1, a_2) \in \mathcal{A}$, the role assertion $R(a_1, a_2)$ is called *split-safe with respect to* $asds_{\mathcal{OS}}$ if

1. for each $(R, R_2) \in \alpha^{crcs}_{\mathcal{OS}}$, $R_2 \notin \alpha^{crts}_{\mathcal{OS}}$,
2. for each $(R_2, C) \in \alpha^{cfis}_{\mathcal{OS}}$, such that $(R, R_2) \in \alpha^{crcs}_{\mathcal{OS}}$,
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a_2) \in \mathcal{A}$ and $C_2 \sqsubseteq C \in \alpha^{stcs}_{\mathcal{OS}}$ or
   - there exists a concept description C_2, such that $C_2(a_2) \in \mathcal{A}$ and $(C_2, C) \in \alpha^{stds}_{\mathcal{OS}}$,

   and
3. for each $(R_2, C) \in \alpha^{cfis}_{\mathcal{OS}}$, such that $(R^-, R_2) \in \alpha^{crcs}_{\mathcal{OS}}$,
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a_1) \in \mathcal{A}$ and $C_2 \sqsubseteq C \in \alpha^{stcs}_{\mathcal{OS}}$ or
   - there exists a concept description C_2, such that $C_2(a_1) \in \mathcal{A}$ and $(C_2, C) \in \alpha^{stds}_{\mathcal{OS}}$.

In Lemma 5.1, we show that any split-safe role assertion for an abstract split decision system is $\mathcal{SHI}$-splittable with respect to the underlying ontology.

Lemma 5.1 (Split Safe Role Assertion)**:**
Given an abstract split decision system $asds_{\mathcal{OS}} = \langle \alpha^{stcs}_{\mathcal{OS}}, \alpha^{stds}_{\mathcal{OS}}, \alpha^{cfis}_{\mathcal{OS}}, \alpha^{crcs}_{\mathcal{OS}}, \alpha^{crts}_{\mathcal{OS}}, \mathcal{A} \rangle$ for an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, history \rangle$ and a role assertion $R(a_1, a_2) \in \mathcal{A}$, $R(a_1, a_2)$ is $\mathcal{SHI}$-splittable with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$ if $R(a_1, a_2)$ is split-safe with respect to $asds_{\mathcal{OS}}$.

Proof of Lemma 5.1. By contradiction: Assume that role assertion $R(a_1, a_2)$ is split-safe with respect to $asds_{\mathcal{OS}}$ and $R(a_1, a_2)$ is not $\mathcal{SHI}$-splittable with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$. Then, by Definition 3.18, we have three cases:

1. There exists a role R_2, such that $\mathcal{O} \models R \sqsubseteq R_2$ and R_2 is transitive with respect to $\mathcal{O}$:

   Since $R(a_1, a_2)$ is split-safe with respect to $asds_{\mathcal{OS}}$, and $\alpha^{crts}_{\mathcal{OS}}$ is a complete RBox transitivity structure for $\mathcal{OS}$, we conclude that R cannot have a subsuming transitive role. Contradiction.

2. There exists a $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R)$ such that
   - $C \neq \bot$ and
   - for all concept descriptions C_2, such that $C_2(a_2) \in \mathcal{A}$, we have $\mathcal{T} \not\models C_2 \sqsubseteq C$ and
   - for all concept descriptions C_2, such that $C_2(a_2) \in \mathcal{A}$, we have $\mathcal{T} \not\models C \sqcap C_2 \sqsubseteq \bot$.

   Since $\alpha^{cfis}_{\mathcal{OS}}$ is a complete $\forall$-info structure for $\mathcal{OS}$ and $\alpha^{crcs}_{\mathcal{OS}}$ is a complete RBox classification structure for $\mathcal{OS}$, there must exist a $R_2 \in \mathbf{Rol}$ such that $(R_2, C) \in \alpha^{cfis}_{\mathcal{OS}}$ and $(R, R_2) \in \alpha^{crcs}_{\mathcal{OS}}$. However, since $R(a_1, a_2)$ is split-safe with respect to $asds_{\mathcal{OS}}$, we have that
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a_2) \in \mathcal{A}$ and $C_2 \sqsubseteq C \in \alpha^{stcs}_{\mathcal{OS}}$ or
   - there exists a concept description C_2, such that $C_2(a_2) \in \mathcal{A}$ and $(C_2, C) \in \alpha^{stds}_{\mathcal{OS}}$.

   Contradiction by soundness of $\alpha^{stcs}_{\mathcal{OS}}$ and $\alpha^{stds}_{\mathcal{OS}}$.
3. There exists a $C \in \mathit{extinfo}^{\forall}_{\mathcal{T},\mathcal{R}}(R^-)$ such that
   - $C \neq \bot$ and
   - for all concept descriptions C_2, such that $C_2(a_1) \in \mathcal{A}$, we have $\mathcal{T} \not\models C_2 \sqsubseteq C$ and
   - for all concept descriptions C_2, such that $C_2(a_1) \in \mathcal{A}$, we have $\mathcal{T} \not\models C \sqcap C_2 \sqsubseteq \bot$.

   Since $\alpha^{cfis}_{\mathcal{OS}}$ is a complete $\forall$-info structure for $\mathcal{OS}$ and $\alpha^{crcs}_{\mathcal{OS}}$ is a complete RBox classification structure for $\mathcal{OS}$, there must exist a $R_2 \in \mathbf{Rol}$ such that $(R_2, C) \in \alpha^{cfis}_{\mathcal{OS}}$ and $(R^-, R_2) \in \alpha^{crcs}_{\mathcal{OS}}$. However, since $R(a_1, a_2)$ is split-safe with respect to $asds_{\mathcal{OS}}$, we have that
   - $C = \bot$ or
   - there exists a concept description C_2, such that $C_2(a_1) \in \mathcal{A}$ and $C_2 \sqsubseteq C \in \alpha^{stcs}_{\mathcal{OS}}$ or
   - there exists a concept description C_2, such that $C_2(a_1) \in \mathcal{A}$ and $(C_2, C) \in \alpha^{stds}_{\mathcal{OS}}$.

   Contradiction by soundness of $\alpha^{stcs}_{\mathcal{OS}}$ and $\alpha^{stds}_{\mathcal{OS}}$.

□

5.3 Syntactic Update Structures

We have defined necessary criteria for all update structures over ontology states, and in the remaining part we formally define these structures in an updatable way. The idea usually is to define a structure for the empty ontology state and then define a transition for each syntactic ontology update.

5.3.1 Updatable Sound TBox Classification Structure

In the following, we define an updatable sound TBox classification structure. The idea is to read off all obvious subsumptions from a general concept inclusion axiom, without the need for complex reasoning. In Definition 5.23, we define how to determine these obvious classification consequences of a concept inclusion.

Definition 5.23 (Obvious Classification Consequences of a General Concept Inclusion)**:** Given a general concept inclusion $C_1 \sqsubseteq C_2$, the set of *obvious classification consequences*, denoted $\mathbf{occ}\,[C_1 \sqsubseteq C_2]$, is defined as a set, such that $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$ if and only if

- $C_1 = C_{a,1} \sqcup ... \sqcup C_{a,l}$ and $C_2 = C_{b,1} \sqcap ... \sqcap C_{b,m}$, and $\exists i \in \{1, ..., l\}.\exists j \in \{1, ..., m\}.C_3 = C_{a,i} \wedge C_4 = C_{b,j}$ or
- $C_1 = C_{a,1} \sqcup ... \sqcup C_{a,l}$ and $C_2 = C_{b,1} \sqcap ... \sqcap C_{b,m}$, and $\exists i \in \{1, ..., l\}.\exists j \in \{1, ..., m\}.C_4 = nnf(\neg C_{a,i}) \wedge C_3 = nnf(\neg C_{b,j})$.

Please note that in Definition 5.23, we also include the case of conjunctions/disjunctions with one element. In general, it is possible to capture more sound consequences without additional reasoning, by more sophisticated syntactical analysis, for instance *anytime classification* shown in [SBK+07]. Thus, more sophisticated algorithms can increase the number of $\mathcal{SHI}$-splittable role assertions. However, in our work we have followed the simple approach for determining obvious classification consequences.

An example for obvious classification consequences of a general concept inclusion is given in Example 5.3.

Example 5.3 (Obvious Classification Consequences)**:**
We have, for instance,

$$\begin{aligned}&\mathbf{occ}\,[Chair \sqsubseteq Person \sqcap \exists headOf.Department] = \\ &\{ \\ &Chair \sqsubseteq Person, \neg Person \sqsubseteq \neg Chair, \\ &Chair \sqsubseteq \exists headOf.Department, \\ &\forall headOf.\neg Department \sqsubseteq \neg Chair \\ &\}.\end{aligned}$$

In Proposition 5.2, we show that the results from Definition 5.23 are indeed sound consequences. But first, we show in Proposition 5.1 that interpretations carry over from concept inclusions to their obvious classification consequences.

Proposition 5.1 (Obvious Classification Consequences of a General Concept Inclusion)**:** Given a general concept inclusion $C_1 \sqsubseteq C_2$ and the set of obvious classification consequences $\mathbf{occ}\,[C_1 \sqsubseteq C_2]$ for $C_1 \sqsubseteq C_2$, for every interpretation $\mathcal{I}$ and each $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$, $\mathcal{I} \vDash C_1 \sqsubseteq C_2 \implies \mathcal{I} \vDash C_3 \sqsubseteq C_4$.

Proof of Proposition 5.1. Since $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$, we can distinguish two cases:

- $C_1 = C_{a,1} \sqcup ... \sqcup C_{a,l}$ and $C_2 = C_{b,1} \sqcap ... \sqcap C_{b,m}$, and $\exists i \in \{1, ..., l\}.\exists j \in \{1, ..., m\}.C_3 = C_{a,i} \wedge C_4 = C_{b,j}$: By $\mathcal{I} \vDash C_1 \sqsubseteq C_2$, we have $(C_{a,1} \sqcup ... \sqcup C_{a,l})^{\mathcal{I}} \subseteq (C_{b,1} \sqcap ... \sqcap C_{b,l})^{\mathcal{I}}$. Since we have $C_3 = C_{a,i}$ and $C_4 = C_{b,j}$, we can conclude $C_3^{\mathcal{I}} \subseteq C_4^{\mathcal{I}}$. This yields $\mathcal{I} \vDash C_3 \sqsubseteq C_4$.
- $C_1 = C_{a,1} \sqcup ... \sqcup C_{a,l}$ and $C_2 = C_{b,1} \sqcap ... \sqcap C_{b,m}$, and $\exists i \in \{1, ..., l\}.\exists j \in \{1, ..., m\}.C_4 = nnf(\neg C_{a,i}) \wedge C_3 = nnf(\neg C_{b,j})$: Analogously to the first case.

□

Proposition 5.2 (Obvious Classification Consequences of an Ontology)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for all $C_1 \sqsubseteq C_2 \in \mathcal{T}$ and for all $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$, $\mathcal{O} \vDash C_3 \sqsubseteq C_4$.

Proof of Proposition 5.2. For each interpretation $\mathcal{I}$, such that $\mathcal{I} \vDash \mathcal{O}$, we know that for each $C_1 \sqsubseteq C_2 \in \mathcal{T}$, we have $\mathcal{I} \vDash C_1 \sqsubseteq C_2$, and by Proposition 5.1, we know that $\mathcal{I} \vDash C_3 \sqsubseteq C_4$ for each $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$. Thus, for each $C_3 \sqsubseteq C_4 \in \mathbf{occ}\,[C_1 \sqsubseteq C_2]$, we have $\mathcal{O} \vDash C_3 \sqsubseteq C_4$. □

In the following, we use obvious classification consequences of a concept inclusion axiom to define an updatable sound TBox classification structure. First, we define an updatable sound classification snapshot in Definition 5.24.

For the initial ontology state there exist no classification consequences, and for each syntactic TBox update we adjust the sound classification snapshot. Please note the usage of multisets in Definition 5.24. This is necessary, since some concept subsumptions might be entailed by several TBox inclusion axioms, which has to be taken into account when retracting a TBox axiom from the ontology state, i.e. we have to know, whether an obvious classification consequence is still entailed, or not.

Definition 5.24 (Updatable Sound TBox Classification Snapshot)**:**
An *updatable sound TBox classification snapshot for an ontology state* $\mathcal{OS} = \langle \mathcal{O}, history \rangle$, denoted $stclss\,[\mathcal{OS}]$, is a multiset over **GCIs**, defined inductively as follows:

- For the initial ontology state $\langle \langle \emptyset, \emptyset, \emptyset \rangle, \square \rangle$, let $stclss\,[\langle \langle \emptyset, \emptyset, \emptyset \rangle, \square \rangle] = \Xi$.
- Given an updatable sound TBox classification snapshot $stclss\,[\langle \mathcal{O}, history \rangle]$ and an syntactic ontology update upd, we let $stclss\,[\mathcal{OS} \uparrow upd] =$

$$\begin{cases} stclss\,[\langle \mathcal{O}, history \rangle] \biguplus \uparrow_S^M (\mathbf{occ}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{+}C_{u1} \sqsubseteq C_{u2}\|\,, \\ stclss\,[\langle \mathcal{O}, history \rangle] \setminus \uparrow_S^M (\mathbf{occ}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{-}C_{u1} \sqsubseteq C_{u2}\|\,, \\ stclss\,[\langle \mathcal{O}, history \rangle] & \text{otherwise.} \end{cases}$$

Lemma 5.2 (Updatable Sound TBox Classification Snapshot Constraint)**:**
Given an updatable sound TBox classification snapshot $stclss\,[\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle]$ and two atomic concept descriptions $C_1 \in \mathbf{AtCon}$ and $C_2 \in \mathbf{AtCon}$,

$$stclss\,[\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle]\,(C_1 \sqsubseteq C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T} \mid C_1 \sqsubseteq C_2 \in \mathbf{occ}\,[C_3 \sqsubseteq C_4]\}|.$$

Proof of Lemma 5.2. By induction on the construction of ontology states:

- Induction base:

  For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\square\rangle$, we have $stclss\,[\langle\langle\emptyset,\emptyset,\emptyset\rangle,\square\rangle] = \Xi$, and thus $stclss\,[\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle]\,(C_1 \sqsubseteq C_2) = 0$. Since $\mathcal{T} = \emptyset$, we have $|\{C_3 \sqsubseteq C_4 \in \mathcal{T} \mid C_1 \sqsubseteq C_2 \in \mathbf{occ}\,[C_3 \sqsubseteq C_4]\}| = 0$. We obtain $0 = 0$, for each C_1 and C_2.

- Induction step:

  Let $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle = upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle)$. We assume

  $$stclss\,[\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle]\,(C_1 \sqsubseteq C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T} \mid C_1 \sqsubseteq C_2 \in \mathbf{occ}\,[C_3 \sqsubseteq C_4]\}|$$

  and have to show that

  $$stclss\,[\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle), upd \circ history\rangle]\,(C_1 \sqsubseteq C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T}_2 \mid C_1 \sqsubseteq C_2 \in \mathbf{occ}\,[C_3 \sqsubseteq C_4]\}|.$$

  It is easily seen by case analysis on the syntactic ontology updates that the value of $stclss\,[\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle), upd \circ history\rangle]\,(C_1 \sqsubseteq C_2)$ is increased (or decreased) by one, whenever a new general concept inclusion axiom entails $C_1 \sqsubseteq C_2$ as an obvious classification consequence, by Proposition 5.2. The same argumentation is true for syntactic TBox retractions.

□

Given a sound TBox classification snapshot, we would like to define an updatable sound TBox classification over ontology states. Formally, we compute the transitive closure over all obvious classification consequences and adapt the transitive closure during updates, e.g., if a new general concept inclusion is added, we saturate the transitive closure. In order to keep the transitive closure updatable we need some kind of dependency structure to keep track which obvious classification consequences are used for computing transitive conclusions. This is formally defined in Definition 5.25.

Definition 5.25 (Updatable Sound TBox Classification Structure)**:**
An *updatable sound TBox classification structure for an ontology state* $\mathcal{OS}$, denoted $\beta^{stcs}_{\mathcal{OS}}$, is a function $\beta^{stcs}_{\mathcal{OS}} : \mathbf{GCIs} \to \wp(\mathbf{GCIs})$, defined inductively as follows:

Figure 5.1 Updating sound TBox classification structures

Input: Updatable sound TBox classification structure $\beta^{stcs}_{\mathcal{OS}}$ and an syntactic ontology update *upd*

Output: $\beta^{stcs}_{\mathcal{OS}\uparrow upd}$

Algorithm:

Let $\beta^{stcs}_{\mathcal{OS}\uparrow upd} = \beta^{stcs}_{\mathcal{OS}}$
Let minus $= \downarrow^{M}_{S} (stclss\,[\mathcal{OS}]) \setminus \downarrow^{M}_{S} (stclss\,[\mathcal{OS} \uparrow upd])$
Let plus $= \downarrow^{M}_{S} (stclss\,[\mathcal{OS} \uparrow upd]) \setminus \downarrow^{M}_{S} (stclss\,[\mathcal{OS}])$
For $C_1 \sqsubseteq C_2 \in$ **minus do**
  Set $\beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_3 \sqsubseteq C_4) = n.d.$, if $C_1 \sqsubseteq C_2 \in \beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_3 \sqsubseteq C_4)$
For $C_1 \sqsubseteq C_2 \in$ **plus do**
  Set $\beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_1 \sqsubseteq C_2) = \{C_1 \sqsubseteq C_2\}$
While there exists three concept descriptions C_1, C_2, C_3, and $y_a \subseteq \wp($**GCIs**$)$, and $y_b \subseteq \wp($**GCIs**$)$, such that $\beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_1 \sqsubseteq C_2) = x_a$, $\beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_2 \sqsubseteq C_3) = x_b$ and $C_1 \sqsubseteq C_3 \notin FD(\beta^{stcs}_{\mathcal{OS}\uparrow upd})$,
  Set $\beta^{stcs}_{\mathcal{OS}\uparrow upd}(C_1 \sqsubseteq C_3) = x_a \cup x_b$

- For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle$, let $\beta^{stcs}_{\mathcal{OS}} = \emptyset$.
- For each other ontology state, $\beta^{stcs}_{\mathcal{OS}}$ is computed as in Figure 5.1.

The algorithm in Figure 5.1 can be explained as follows. First, the set of new and obsolete classification consequences is computed, respectively, and stored in the sets **plus** and **minus**. All inferred general inclusion axioms depending on obsolete classification consequences are removed and the new basic classification consequences are added. At last, a saturation phase is executed, which completes the transitive closure over the remaining obvious classification consequences.

In Proposition 5.3, we show that the updatable sound TBox classification structure from Definition 5.25 can be used as a sound TBox classification structure for each ontology state.

Proposition 5.3 (Updatable Sound TBox Classification Structure)**:**
For each ontology state $\mathcal{OS}$, the set $FD(\beta^{stcs}_{\mathcal{OS}})$ is a sound TBox classification structure for $\mathcal{OS}$.

Proof of Proposition 5.3. We have to show that for all ontology states $\mathcal{OS} = \langle\mathcal{O}, history\rangle$, we have $C_1 \sqsubseteq C_2 \in FD(\beta^{stcs}_{\mathcal{OS}}) \implies \mathcal{O} \vDash C_1 \sqsubseteq C_2$.

By construction of the algorithm in Figure 5.1, we know that each $C_1 \sqsubseteq C_2 \in FD(\beta^{stcs}_{\mathcal{OS}})$ is an obvious classification consequence of a general concept inclusion axiom in the TBox

(or derived from a set of obvious classification consequences and their transitive closure). By Lemma 5.2, we can conclude that $\mathcal{O} \models C_1 \sqsubseteq C_2$. □

Example 5.4 (Sound TBox Classification Update)**:**
Given an ontology state $\mathcal{OS} = \langle \mathcal{O}_{Ex5.4}, history \rangle$, with an example ontology $\mathcal{O}_{Ex5.4} = \langle \mathcal{T}_{Ex5.4}, \mathcal{R}_{Ex5.4}, \mathcal{A}_{Ex5.4} \rangle$ defined as

$$\begin{aligned} \mathcal{T}_{Ex5.4} &= \{B \sqsubseteq A \sqcap \exists has.D, C \sqsubseteq B\}, \\ \mathcal{R}_{Ex5.4} &= \{\}, \\ \mathcal{A}_{Ex5.4} &= \{\}, \end{aligned}$$

we have that

$$\begin{aligned} stclss\,[\langle \mathcal{O}_{Ex5.4}, history \rangle] = \{ & \\ & B \sqsubseteq A \to 1, \neg A \sqsubseteq \neg B \to 1, \\ & B \sqsubseteq \exists has.D \to 1, \forall has.\neg D \sqsubseteq \neg B \to 1, \\ & C \sqsubseteq B \to 1, \neg B \sqsubseteq \neg C \to 1 \\ & \} \end{aligned}$$

and

$$\begin{aligned} \beta^{stcs}_{\langle \mathcal{O}_{Ex5.4}, history \rangle} = \{ & \\ & B \sqsubseteq A \to \{B \sqsubseteq A\}, \\ & \neg A \sqsubseteq \neg B \to \{\neg A \sqsubseteq \neg B\}, \\ & B \sqsubseteq \exists has.D \to \{B \sqsubseteq \exists has.D\}, \\ & \forall has.\neg D \sqsubseteq \neg B \to \{\forall has.\neg D \sqsubseteq \neg B\}, \\ & C \sqsubseteq B \to \{C \sqsubseteq B\}, \\ & \neg B \sqsubseteq \neg C \to \{\neg B \sqsubseteq \neg C\}, \\ & C \sqsubseteq A \to \{C \sqsubseteq B, B \sqsubseteq A\}, \\ & \neg A \sqsubseteq \neg C \to \{\neg A \sqsubseteq \neg B, \neg B \sqsubseteq \neg C\}, \\ & C \sqsubseteq \exists has.D \to \{C \sqsubseteq B, B \sqsubseteq \exists has.D\}, \\ & \forall has.\neg D \sqsubseteq \neg C \to \{\forall has.\neg D \sqsubseteq \neg B, \neg B \sqsubseteq \neg C\} \\ & \}. \end{aligned}$$

If we perform the syntactic ontology update $upd\,\|{-}B \sqsubseteq A \sqcap \exists has.D\|$ on $\mathcal{OS}$, we obtain

$$\begin{aligned} \beta^{stcs}_{\langle \mathcal{O}_{Ex5.4}, history \rangle \uparrow upd\|-B \sqsubseteq A \sqcap \exists has.D\|} = \{ & \\ & C \sqsubseteq B \to \{C \sqsubseteq B\} \\ & \neg B \sqsubseteq \neg C \to \{\neg B \sqsubseteq \neg C\} \\ & \}, \end{aligned}$$

since all entries referring to obvious classification consequences of $B \sqsubseteq A \sqcap \exists has.D$, i.e. referring to $B \sqsubseteq A$, $\neg A \sqsubseteq \neg B$, $B \sqsubseteq \exists has.D$, and $\forall has.\neg D \sqsubseteq \neg B$, are removed.

Please note that the computation of the transitive closure might be quite time consuming, and there is some storage overhead for materializing all sound subsumptions together with the dependencies. But since we would like to compute a *sound* TBox classification structure *only*, we could stop the saturation phase at any time if some memory/runtime constraint is violated. The results obtained so far are still sound.

5.3.2 Updatable Sound TBox Disjointness Structure

In the following, we define an updatable sound TBox disjointness structure. The idea is to extract all obvious disjointness consequences from a concept inclusion axiom, without the need for complex reasoning. In Definition 5.26, we describe how to determine these obvious disjointness consequences from a concept inclusion axiom.

Definition 5.26 (Obvious Disjointness Consequences of a General Concept Inclusion)**:** Given a general concept inclusion $C_1 \sqsubseteq C_2$ and its normal form $\top \sqsubseteq C_{negnorm}$, the set of *obvious disjointness consequences*, denoted $\mathbf{odc}\,[C_1 \sqsubseteq C_2]$, is defined as a set over $\mathbf{Con} \times \mathbf{Con}$, such that $(nnf(C_3), nnf(C_4)) \in \mathbf{odc}\,[C_1 \sqsubseteq C_2]$ if and only if $C_{negnorm} = C_{a,1} \sqcap C_{a,2} \sqcap ... \sqcap C_{a,n}$ and there exists an $i \in \{1, ..., n\}$, such that

- $C_{a,i} = \neg C_3 \sqcup \neg C_4$,
- $C_{a,i} = \neg C_4 \sqcup \neg C_3$,
- $C_{a,i} = \neg C_3 \sqcup \neg C_4 \sqcup \bot$, or
- $C_{a,i} = \neg C_4 \sqcup \neg C_3 \sqcup \bot$.

The definition of obvious disjointness consequences can be changed to capture more sound disjointness consequences without additional reasoning. However, the approach in Definition 5.26 is the one we have used for our evaluation. In Proposition 5.5, we show that the results from Definition 5.26 are indeed sound consequences. But first, we show in Proposition 5.4 that interpretations carry over from concept inclusion axioms to their obvious disjointness consequences.

Proposition 5.4 (Obvious Disjointness Consequences of a General Concept Inclusion)**:** Given a general concept inclusion $C_1 \sqsubseteq C_2$ and the set of obvious disjointness consequences $\mathbf{odc}\,[C_1 \sqsubseteq C_2]$ for $C_1 \sqsubseteq C_2$, for every interpretation $\mathcal{I} = \langle \Delta_\mathcal{I}, \cdot^\mathcal{I} \rangle$ and each $(C_3, C_4) \in \mathbf{odc}\,[C_1 \sqsubseteq C_2]$, $\mathcal{I} \vDash C_1 \sqsubseteq C_2 \implies \mathcal{I} \vDash C_3 \oslash C_4$.

Proof of Proposition 5.4. Let $\top \sqsubseteq C_{negnorm}$ be the normal form of $C_1 \sqsubseteq C_2$. By Definition 5.26, we can distinguish four cases:

- $C_{negnorm} = C_{a,1} \sqcap C_{a,2} \sqcap ... \sqcap C_{a,n}$ and $C_{a,i} = \neg C_3 \sqcup \neg C_4$: From $\mathcal{I} \vDash C_1 \sqsubseteq C_2$, we can conclude $\mathcal{I} \vDash \top \sqsubseteq C_{negnorm}$. Thus, by Definition 2.15, we have $\Delta_\mathcal{I} \subseteq C_{a,1}^\mathcal{I} \cap C_{a,2}^\mathcal{I} \cap ... \cap C_{a,n}^\mathcal{I}$ and especially $\Delta_\mathcal{I} \subseteq C_{a,i}^\mathcal{I}$. This yields $\Delta_\mathcal{I} \subseteq \neg C_3^\mathcal{I} \cup \neg C_4^\mathcal{I}$ and we obtain $C_3^\mathcal{I} \cap C_4^\mathcal{I} \subseteq \emptyset$. Thus, we have $\mathcal{I} \vDash C_3 \oslash C_4$.

- $C_{negnorm} = C_{a,1} \sqcap C_{a,2} \sqcap ... \sqcap C_{a,n}$ and $C_{a,i} = \neg C_4 \sqcup \neg C_3$: analogously.
- $C_{negnorm} = C_{a,1} \sqcap C_{a,2} \sqcap ... \sqcap C_{a,n}$ and $C_{a,i} = \neg C_3 \sqcup \neg C_4 \sqcup \bot$: analogously, since $\bot^{\mathcal{I}} = \emptyset$.
- $C_{negnorm} = C_{a,1} \sqcap C_{a,2} \sqcap ... \sqcap C_{a,n}$ and $C_{a,i} = \neg C_4 \sqcup \neg C_3 \sqcup \bot$: analogously.

□

Proposition 5.5 (Obvious Disjointness Consequences of an Ontology)**:**
Given an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, for all $C_1 \sqsubseteq C_2 \in \mathcal{T}$ and for all $(C_3, C_4) \in \mathbf{odc}\,[C_1 \sqsubseteq C_2]$, $\mathcal{O} \models C_3 \oslash C_4$.

Proof of Proposition 5.5. For each interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{O}$, we know that $\mathcal{I} \models C_1 \sqsubseteq C_2$ for each $C_1 \sqsubseteq C_2 \in \mathcal{T}$, and by Proposition 5.4, we know that $\mathcal{I} \models C_3 \oslash C_4$ for each $(C_3, C_4) \in \mathbf{odc}\,[C_1 \sqsubseteq C_2]$. Thus, $\mathcal{O} \models C_3 \oslash C_4$ for each $(C_3, C_4) \in \mathbf{odc}\,[C_1 \sqsubseteq C_2]$. □

In Definition 5.27, we use obvious disjointness consequences of a general concept inclusion to define an updatable sound TBox disjointness structure. For the initial ontology state there exist no disjointness consequences and for each syntactic TBox update, we adjust the obvious disjointness consequences for the ontology state. For the same reason as in Definition 5.24 we use multisets again, in order to handle consequences entailed by multiple general concept inclusions.

Definition 5.27 (Updatable Sound TBox Disjointness Structure)**:**
Given an ontology state $\mathcal{OS}$, an *updatable sound TBox disjointness structure for* $\mathcal{OS}$, denoted $\beta^{stds}_{\mathcal{OS}}$, is a multiset over $\mathbf{Con} \times \mathbf{Con}$, defined inductively as follows:

- For the initial ontology state $\langle \langle \emptyset, \emptyset, \emptyset \rangle, \Box \rangle$, let $\beta^{stds}_{\langle \langle \emptyset, \emptyset, \emptyset \rangle, \Box \rangle} = \Xi$.
- Given an updatable sound TBox disjointness structure $\beta^{stds}_{\langle \mathcal{O}, history \rangle}$ and an applicable syntactic ontology update upd, we let $\beta^{stds}_{\mathcal{OS} \uparrow upd} =$

$$\begin{cases} \beta^{stds}_{\langle \mathcal{O}, history \rangle} \uplus \uparrow^M_S (\mathbf{odc}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{+}C_{u1} \sqsubseteq C_{u2}\|\,, \\ \beta^{stds}_{\langle \mathcal{O}, history \rangle} \setminus \uparrow^M_S (\mathbf{odc}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{-}C_{u1} \sqsubseteq C_{u2}\|\,, \\ \beta^{stcs}_{\langle \mathcal{O}, history \rangle} & \text{otherwise.} \end{cases}$$

Lemma 5.3 (Updatable Sound TBox Disjointness Structure Constraint)**:**
Given an updatable sound TBox disjointness structure $\beta^{stds}_{\langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, history \rangle}$ and two concept descriptions $C_1 \in \mathbf{Con}$ and $C_2 \in \mathbf{Con}$,

$$\beta^{stds}_{\langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, history \rangle}(C_1, C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T} |\ (C_1, C_2) \in \mathbf{odc}\,[C_3 \sqsubseteq C_4]\}|.$$

Proof of Lemma 5.3. By induction on the construction of ontology states:

- Induction base:

  For $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle$, $\beta^{stds}_{\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle} = \Xi$, and thus we have $\beta^{stds}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}(C_1,C_2) = 0$. Since $\mathcal{T} = \emptyset$, we also have $|\{C_3 \sqsubseteq C_4 \in \mathcal{T} \mid (C_1,C_2) \in \mathbf{odc}\,[C_3 \sqsubseteq C_4]\}| = 0$. We obtain $0 = 0$, for each (C_1,C_2).

- Induction step:

  Let $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle = upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle)$. We assume

  $$\beta^{stds}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}(C_1,C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T} \mid (C_1,C_2) \in \mathbf{odc}\,[C_3 \sqsubseteq C_4]\}|$$

  and have to show that

  $$\beta^{stds}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle}(C_1,C_2) = |\{C_3 \sqsubseteq C_4 \in \mathcal{T}_2 \mid (C_1,C_2) \in \mathbf{odc}\,[C_3 \sqsubseteq C_4]\}|.$$

  It is easily seen by case analysis on the syntactic ontology updates that the value of $\beta^{stds}_{\langle upd([\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle]}(C_1,C_2)$ is increased (or decreased) by one, whenever a new general concept inclusion axiom entails (C_1,C_2) as an obvious disjointness consequence (or an entailing general concept inclusion axiom is removed).

□

In Proposition 5.6, we show that the updatable sound TBox disjointness structure can be used as a sound TBox disjointness structure for each ontology state.

Proposition 5.6 (Updatable Sound TBox Disjointness Structure)**:**
For each ontology state $\mathcal{OS}$, the set $\downarrow^M_S (\beta^{stds}_{\mathcal{OS}})$ is a sound TBox disjointness structure for $\mathcal{OS}$.

Proof of Proposition 5.6. By Definition 5.14, we have to show that for all ontology states $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle$, we have $(C_1,C_2) \in \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}) \implies \langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle \vDash C_1 \oslash C_2$.

By contradiction: Assume that there exists a $(C_1,C_2) \in \beta^{stds}_{\mathcal{OS}}$, such that $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle \nvDash C_1 \oslash C_2$. By Lemma 5.3, we know that $(C_1,C_2) \in \beta^{stds}_{\mathcal{OS}}$ if and only if there exists a $C_3 \sqsubseteq C_4 \in \mathcal{T}$, such that $(C_1,C_2) \in \mathbf{odc}\,[C_3 \sqsubseteq C_4]$. By Proposition 5.5, we can conclude that $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle \vDash C_1 \oslash C_2$. Contradiction. □

5.3.3 Updatable Complete ∀-info Structure

Next, we define an updatable complete ∀-info structure. In Definition 5.28, we describe how to extract all possible ∀-propagations from a given general concept inclusion axiom.

Definition 5.28 (∀-Concept Description Closure of a General Concept Inclusion)**:**
Given a general concept inclusion $C_1 \sqsubseteq C_2$, the *∀-concept description closure of* $C_1 \sqsubseteq C_2$, denoted $\mathbf{fac}\,[C_1 \sqsubseteq C_2]$, is defined as a set, such that $(R,C_3) \in \mathbf{fac}\,[C_1 \sqsubseteq C_2]$ if and only if $\forall R.C_3 \in (clos(\neg C_1) \cup clos(C_2))$.

The extension from concept inclusions to ontology states is straightforward, given Definition 5.28. We define an updatable ∀-info structure for ontology states in Definition 5.29. Again, we make use of multisets as the basic data structure.

Definition 5.29 (Updatable ∀-Info Structure)**:**
An *updatable ∀-info structure for an ontology state* $\mathcal{OS} = \langle \mathcal{O}, history \rangle$, denoted $\beta^{cfis}_{\mathcal{OS}}$, is a multiset over $\mathbf{Rol} \times \mathbf{Con}$, defined inductively as follows:

- For the initial ontology state $\langle \langle \emptyset, \emptyset, \emptyset \rangle, \Box \rangle$, let $\beta^{cfis}_{\langle \langle \emptyset,\emptyset,\emptyset \rangle,\Box \rangle} = \Xi$.
- Given an updatable ∀-info structure $\beta^{cfis}_{\langle \mathcal{O},history \rangle}$ and an syntactic ontology update upd, we let $\beta^{cfis}_{\langle \mathcal{O},history \rangle \uparrow upd} =$

$$\begin{cases} \beta^{cfis}_{\langle \mathcal{O},history \rangle} \uplus \uparrow^M_S (\mathbf{fac}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{+}C_{u1} \sqsubseteq C_{u2}\|\,, \\ \beta^{cfis}_{\langle \mathcal{O},history \rangle} \setminus \uparrow^M_S (\mathbf{fac}\,[C_{u1} \sqsubseteq C_{u2}]) & \text{if } upd = upd\,\|{-}C_{u1} \sqsubseteq C_{u2}\|\,, \\ \beta^{cfis}_{\langle \mathcal{O},history \rangle} & \text{otherwise.} \end{cases}$$

Lemma 5.4 (Updatable ∀-Info Structure Constraint)**:**
Given an updatable ∀-info structure $\beta^{cfis}_{\langle \langle \mathcal{T},\mathcal{R},\mathcal{A} \rangle,history \rangle}$, a concept description $C \in \mathbf{Con}$, and a role description $R \in \mathbf{Rol}$,

$$\beta^{cfis}_{\langle \langle \mathcal{T},\mathcal{R},\mathcal{A} \rangle,history \rangle}(R, C) = |\{C_2 \sqsubseteq C_3 \in \mathcal{T} |\ (R, C) \in \mathbf{fac}\,[C_2 \sqsubseteq C_3]\}|.$$

Proof of Lemma 5.4. By induction on the construction of ontology states:

- Induction base:

  For the initial ontology state $\langle \langle \emptyset, \emptyset, \emptyset \rangle, \Box \rangle$, $\beta^{cfis}_{\langle \langle \emptyset,\emptyset,\emptyset \rangle,\Box \rangle} = \Xi$, and we can conclude $\beta^{cfis}_{\langle \langle \mathcal{T},\mathcal{R},\mathcal{A} \rangle,history \rangle}(R, C) = 0$. Since $\mathcal{T} = \emptyset$, we have $|\{C_2 \sqsubseteq C_3 \in \mathcal{T} |\ (R, C) \in \mathbf{fac}\,[C_2 \sqsubseteq C_3]\}| = 0$. We obtain $0 = 0$, for each (R, C).

- Induction step:

  Let $\langle \mathcal{T}_2, \mathcal{R}_2, \mathcal{A}_2 \rangle = upd(\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle)$. We assume

  $$\beta^{cfis}_{\langle \mathcal{O},history \rangle}(R, C) = |\{C_2 \sqsubseteq C_3 \in \mathcal{T} |\ (R, C) \in \mathbf{fac}\,[C_2 \sqsubseteq C_3]\}|$$

  and have to show that

  $$\beta^{cfis}_{\langle upd(\mathcal{O}),upd \circ history \rangle}(R, C) = |\{C_2 \sqsubseteq C_3 \in \mathcal{T}_2 \mid (R, C) \in \mathbf{fac}\,[C_2 \sqsubseteq C_3]\}|.$$

  It is easily seen by case analysis on the syntactic ontology updates that the value of $\beta^{cfis}_{\langle upd([\langle \mathcal{T},\mathcal{R},\mathcal{A} \rangle)),upd \circ history \rangle]}(R, C)$ is increased (or decreased) by one, whenever a new general concept inclusion axiom has (R, C) in its ∀-concept description closure (or a containing general concept inclusion axiom is removed).

□

The results from Lemma 5.4 are summarized in Proposition 5.7, where we show that an updatable complete ∀-info structure is indeed a complete ∀-info structure for each ontology state.

Proposition 5.7 (Updatable ∀-Info Structure)**:**
For each ontology state $\mathcal{OS} = \langle\langle\mathcal{T}, \mathcal{R}, \mathcal{A}\rangle, \mathit{history}\rangle$, the set $\downarrow_S^M (\beta_{\mathcal{OS}}^{cfis})$ is a complete ∀-info structure for $\mathcal{OS}$.

Proof of Proposition 5.7. We have to show that $\forall R.C \in \mathit{clos}(\mathcal{T}) \implies (R, C) \in \downarrow_S^M (\beta_{\mathcal{OS}}^{cfis})$. Assuming $\forall R.C \in \mathit{clos}(\mathcal{T})$, we know by Definition 2.20 that there exists a $C_2 \sqsubseteq C_3 \in \mathcal{T}$, such that $(R, C) \in \mathbf{fac}\,[C_2 \sqsubseteq C_3]$. By Lemma 5.4, we conclude that $(R, C) \in \downarrow_S^M (\beta_{\mathcal{OS}}^{cfis})$. □

5.3.4 Updatable Complete RBox Structures

In the following, we define RBox-related structures. Since the RBox is usually rather small, and the computation is not so expensive, we stick to sound and complete implementations below. However, it should be kept in mind that actually only completeness is necessary. In Definition 5.30, we define an updatable complete (and sound) RBox classification structure. For each each syntactic TBox and RBox update, we compute the set of role subsumptions from scratch. Please note that this approach is not *updatable* literally. We leave the investigation for real updatable complete RBox structures for future work.

Definition 5.30 (Updatable Complete RBox Classification Structure)**:**
An *updatable complete RBox classification structure for an ontology state* $\mathcal{OS}$, denoted $\beta_{\mathcal{OS}}^{crcs}$, is a multiset over $\mathbf{Rol} \times \mathbf{Rol}$, defined inductively as follows:

- For the initial ontology state $\langle\langle\emptyset, \emptyset, \emptyset\rangle, \Box\rangle$, let $\beta_{\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle}^{crcs} = \Xi$.
- Given an updatable complete RBox classification $\beta_{\langle\mathcal{O},\mathit{history}\rangle}^{crcs}$ and an syntactic ontology update *upd*, we let $\beta_{\mathcal{OS}\uparrow \mathit{upd}}^{crcs}(R_1, R_2) =$

$$\begin{cases} 1 & \text{if } (R_1, R_2) \in \underline{rc}^{\mathit{upd}(\mathcal{O})}, \\ 0 & \text{otherwise.} \end{cases}$$

The proof of completeness in Proposition 5.8 is straightforward.

Proposition 5.8 (Updatable Complete RBox Classification Structure)**:**
For all ontology states $\mathcal{OS} = \langle\mathcal{O}, \mathit{history}\rangle$, we have that $\downarrow_S^M (\beta_{\mathcal{OS}}^{crcs})$ is a complete RBox classification structure for $\mathcal{OS}$.

Proof of Proposition 5.8. We have to show that $\mathcal{O} \models R_1 \sqsubseteq R_2 \implies (R_1, R_2) \in \downarrow_S^M (\beta_{\mathcal{OS}}^{crcs})$. By Definition 5.30, we recompute the whole role hierarchy on each syntactical TBox and RBox update and then add (R_1, R_2) to $\beta_{\mathcal{OS}}^{crcs}$ if and only if $\mathcal{O} \models R_1 \sqsubseteq R_2$. □

Next, we define an updatable complete RBox transitivity structure. Due to the same reason as above, we compute the whole RBox transitivity structure again from scratch for each syntactical TBox and RBox update. We think this is reasonable, since RBoxes are usually rather small.

Definition 5.31 (Updatable Complete RBox Transitivity Structure)**:**
An *updatable complete RBox transitivity structure for an ontology state* $\mathcal{OS}$, denoted $\beta^{crts}_{\mathcal{OS}}$, is a multiset over **Rol**, defined inductively as follows:

- For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle$, let $\beta^{crts}_{\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle} = \Xi$.
- Given an updatable complete RBox transitivity structure $\beta^{crcs}_{\langle\mathcal{O},history\rangle}$ and an syntactic ontology update *upd*, we let $\beta^{crts}_{\mathcal{OS}\uparrow upd}(R_1) =$

$$\begin{cases} 1 & \text{if } R_1 \in \underline{rtc}^{upd(\mathcal{O})}, \\ 0 & \text{otherwise.} \end{cases}$$

The proof of completeness in Proposition 5.9 is again straightforward.

Proposition 5.9 (Updatable Complete RBox Transitivity Structure)**:**
For all ontology states $\mathcal{OS} = \langle\mathcal{O}, history\rangle$, we have that $\downarrow^M_S (\beta^{crts}_{\mathcal{OS}})$ is a complete RBox transitivity structure for $\mathcal{OS}$.

Proof of Proposition 5.9. We have to show that for all R_1, we have $\mathcal{O} \vDash Trans(R_1) \implies R_1 \in \downarrow^M_S (\beta^{crts}_{\mathcal{OS}})$. By Definition 5.31, we recompute all role transitivities on each syntactical TBox and RBox update and then add R_1 to $\beta^{crcs}_{\mathcal{OS}}$ if and only if $\mathcal{O} \vDash Trans(R_1)$. □

At this point we have defined all important basic data structures necessary for determining $\mathcal{SHI}$-splittability in an incremental way.

5.4 Updatable Split Decision System

In the following, we define one concrete implementation of the abstract split decision system from Definition 5.21, by use of the syntactic update structures introduced above.

Definition 5.32 (Updatable Split Decision System)**:**
Given an ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}_{\mathcal{OS}}\rangle, history\rangle$, the tuple

$$updsds(\mathcal{OS}) = \langle FD(\beta^{stcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}), \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}), \mathcal{A}_{\mathcal{OS}}\rangle$$

is called *updatable split decision system for* $\mathcal{OS}$.

Theorem 5.1 (Updatable Split Decision System)**:**
For each ontology state $\mathcal{OS}$, the updatable split decision system $updsds(\mathcal{OS})$ is an abstract split decision system for $\mathcal{OS}$.

Proof of Theorem 5.1. By Proposition 5.3, Proposition 5.6, Proposition 5.7, Proposition 5.8, and Proposition 5.9. □

We introduce the notion of a split set in Definition 5.33. The idea is to capture all split-safe assertions with respect to an ontology state in a set and then define upper bounds on role assertions which can be affected (in terms of split-safety) during a syntactic ontology update. The motivation is to define/reduce the number of checks during the application of an ontology update. This upper bound is further defined in Definition 5.34.

Definition 5.33 (Split Set)**:**
Given an updatable split decision system

$$updsds(\mathcal{OS}) = \langle FD(\beta_{\mathcal{OS}}^{stcs}), \downarrow_S^M (\beta_{\mathcal{OS}}^{stds}), \downarrow_S^M (\beta_{\mathcal{OS}}^{cfis}), \downarrow_S^M (\beta_{\mathcal{OS}}^{crcs}), \downarrow_S^M (\beta_{\mathcal{OS}}^{crts}), \mathcal{A}_{\mathcal{OS}} \rangle$$

for an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}} \rangle, history \rangle$, the *split set for* $updsds(\mathcal{OS})$, denoted $spls(updsds(\mathcal{OS}))$, is defined as

$$spls(updsds(\mathcal{OS})) = \{R(a_1, a_2) \in \mathcal{A}_{\mathcal{OS}} \mid R(a_1, a_2) \texttt{ is split-safe for } updsds(\mathcal{OS})\}.$$

Definition 5.34 (Update Split Difference Bound)**:**
Given an updatable split decision system $updsds(\mathcal{OS})$ for an ontology state $\mathcal{OS}$ and a syntactic ontology update $upd : \mathbf{SO} \rightarrow \mathbf{SO}$, an *update split difference bound*, denoted $sdf(updsds(\mathcal{OS}), upd)$, is a set $\mathbf{S}$ of role assertions, such that

$$\mathbf{S} \supseteq (spls(updsds(\mathcal{OS} \uparrow upd)) \ominus spls(updsds(\mathcal{OS}))).$$

With update split difference bounds, we capture (a superset of) all role assertions, whose split-safety can change during a syntactic ontology update. For all role assertions in the split difference bound, we need to check split-safety after the update. In the following, we derive update split difference bounds for each syntactic ontology update. Please note that the update split difference bounds for most syntactic ontology updates are easy to derive. However, for the sake of completeness, we provide each of these bounds with a lemma and a short proof.

5.4.1 Difference Bounds for Syntactic ABox Updates

Lemma 5.5 (Update Split Difference Bound for Role Assertion Updates)**:**
Given an ontology state $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}} \rangle, history \rangle$, the set $\mathbf{S} = \{R_u(a_{u1}, a_{u2})\}$ is an update split difference bound for $upd \parallel +R_u(a_{u1}, a_{u2}) \parallel$ and $upd \parallel -R_u(a_{u1}, a_{u2}) \parallel$ on $updsds(\mathcal{OS})$.

Proof of Lemma 5.5. Split-safety of a role assertion $R(a_1, a_2)$ can be decided based on the current TBox, RBox and the concept assertions for individual a_1 and and individual a_2. Since $upd \parallel +R_u(a_{u1}, a_{u2}) \parallel$ and $upd \parallel -R_u(a_{u1}, a_{u2}) \parallel$ only change the ABox, i.e. all TBox and RBox based structures are unchanged after the update, and in addition no

concept assertion for any individual is changed, we have that $\mathbf{S} = \{R_u(a_{u1}, a_{u2})\}$ is an update split difference bound for the syntactic ontology updates $upd \,\|{+}R_u(a_{u1}, a_{u2})\|$ and $upd \,\|{-}R_u(a_{u1}, a_{u2})\|$ on $updsds(\mathcal{OS})$. □

Lemma 5.6 (Update Split Difference Bound for Concept Assertion Updates)**:**
Given an ontology state $\mathcal{OS} = \langle\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}}\rangle, history\rangle$, $\mathbf{S} = \{R_x(a_u, a_{x1}) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\} \cup \{R_x(a_{x1}, a_u) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\}$ is an update split difference bound for the syntactic ontology updates $upd \,\|{+}C_u(a_u)\|$ and $upd \,\|{-}C_u(a_u)\|$ on $updsds(\mathcal{OS})$.

Proof of Lemma 5.6. First, please note that C_u is an atomic concept description. Split-safety of a role assertion $R(a_1, a_2)$ can be decided based on the current TBox, RBox and the concept assertions for individual a_1 and and individual a_2. Since $upd \,\|{+}C_u(a_u)\|$ and $upd \,\|{-}C_u(a_u)\|$ only change the ABox, i.e. all TBox and RBox based structures are unchanged after the update, and in addition only the concept assertions for individual a_u are changed, it holds that $\mathbf{S} = \{R_x(a_u, a_{x1}) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\} \cup \{R_x(a_{x1}, a_u) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\}$ is an update split difference bound for $upd \,\|{+}R_u(a_{u1}, a_{u2})\|$ and $upd \,\|{-}R_u(a_{u1}, a_{u2})\|$ on $updsds(\mathcal{OS})$. □

5.4.2 Difference Bounds for Syntactic RBox Updates

Lemma 5.7 (Update Split Difference Bound for Role Inclusion Updates)**:**
Given an ontology state $\mathcal{OS} = \langle\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}}\rangle, history\rangle$, $\mathbf{S} = \{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\}$ is an update split difference bound for the syntactic ontology updates $upd \,\|{+}R_{u1} \sqsubseteq R_{u2}\|$ and $upd \,\|{-}R_{u1} \sqsubseteq R_{u2}\|$ on $updsds(\mathcal{OS}) = \langle FD(\beta^{stcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}), \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}), \mathcal{A}_{\mathcal{OS}}\rangle$, where

$$\begin{aligned}\mathbf{dr} =& \{R \mid \exists R_y \in \mathbf{Rol}.(R, R_y) \in (\downarrow^M_S (\beta^{crcs}_{\mathcal{OS}\uparrow upd}) \ominus \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}))\} \cup \\ & \{R \mid \exists R_y \in \mathbf{Rol}.(R_y, R) \in (\downarrow^M_S (\beta^{crcs}_{\mathcal{OS}\uparrow upd}) \ominus \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}))\} \cup \\ & \{R \mid R \in (\downarrow^M_S (\beta^{crts}_{\mathcal{OS}\uparrow upd}) \ominus \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}))\}.\end{aligned}$$

Proof of Lemma 5.7. Split-safety of a role assertion $R(a_1, a_2)$ can be decided based on the current TBox, RBox, and the concept assertions for individual a_1 and and individual a_2. The syntactic ontology updates $upd \,\|{+}R_{u1} \sqsubseteq R_{u2}\|$ and $upd \,\|{-}R_{u1} \sqsubseteq R_{u2}\|$ change only the RBox, i.e. all TBox based structures and the ABox are unchanged after the update. Thus, we only need to check all role assertions with new/obsolete super/sub roles and new/obsolete transitive roles. □

Lemma 5.8 (Update Split Difference Bound for Role Transitivity Updates)**:**
Given an ontology state $\mathcal{OS} = \langle\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}}\rangle, history\rangle$, the set

$$\mathbf{S} = \{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\}$$

is an update split difference bound for $upd \,\|{+}Trans(R_u)\|$ and $upd \,\|{-}Trans(R_u)\|$ on $updsds(\mathcal{OS}) = \langle FD(\beta^{stcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}), \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}), \mathcal{A}_{\mathcal{OS}}\rangle$, where

$$\mathbf{dr} = \{R \mid \exists R_y \in \mathbf{Rol}.(R_y \in (\downarrow^M_S (\beta^{crts}_{\mathcal{OS}\uparrow upd}) \ominus \downarrow^M_S (\beta^{crts}_{\mathcal{OS}})) \wedge (R, R_y) \in \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}\uparrow upd}))\}.$$

Proof of Lemma 5.8. Split-safety of a role assertion $R(a_1, a_2)$ can be decided based on the current TBox, RBox and the concept assertions for individual a_1 and individual a_2. The syntactic ontology updates $upd \,\|{+}Trans(R_u)\|$ and $upd \,\|{-}Trans(R_u)\|$ change only the RBox, i.e. all TBox based structures and the ABox are unchanged after the update. Thus, we only need to check all role assertions with roles being new/obsolete subroles of transitive roles. □

5.4.3 Difference Bounds for Syntactic TBox Updates

Lemma 5.9 (Update Split Difference Bound for Concept Inclusion Updates)**:**
Given an ontology state $\mathcal{OS} = \langle\langle \mathcal{T}, \mathcal{R}, \mathcal{A}_{\mathcal{OS}}\rangle, history\rangle$, the set

$$\begin{aligned}\mathbf{S} =&\{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\}\cup\\ &\{R_x(a_{x1}, a_{x2}) \mid \{C \mid C(a_{x1}) \in \mathcal{A}_{\mathcal{OS}} \vee C(a_{x2}) \in \mathcal{A}_{\mathcal{OS}}\} \cap \mathbf{dc} \neq \emptyset\}\end{aligned}$$

is an update split difference bound for the updates $upd = upd \,\|{+}C_{u1} \sqsubseteq C_{u2}\|$ and $upd = upd \,\|{-}C_{u1} \sqsubseteq C_{u2}\|$ on $updsds(\mathcal{OS}) = \langle FD(\beta^{stcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}), \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}), \mathcal{A}_{\mathcal{OS}}\rangle$, where

$$\begin{aligned}\mathbf{dc} =&\{C \mid \exists C_y \in \mathbf{AtCon}.C \sqsubseteq C_y \in FVAL(\beta^{stcs}_{\mathcal{OS}\uparrow upd}) \ominus FD(\beta^{stcs}_{\mathcal{OS}})\}\cup\\ &\{C \mid \exists C_y \in \mathbf{AtCon}.(C, C_y) \in\downarrow^M_S (\beta^{stds}_{\mathcal{OS}\uparrow upd})\ominus \downarrow^M_S (\beta^{stds}_{\mathcal{OS}})\}\cup\\ &\{C \mid \exists C_y \in \mathbf{AtCon}.(C_y, C) \in\downarrow^M_S (\beta^{stds}_{\mathcal{OS}\uparrow upd})\ominus \downarrow^M_S (\beta^{stds}_{\mathcal{OS}})\}\\ \mathbf{dr} =&\{R \mid \exists C.(R, C) \in\downarrow^M_S (\beta^{cfis}_{\mathcal{OS}\uparrow upd})\ominus \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}})\}\cup\\ &\{R \mid \exists R_y \in \mathbf{Rol}.(R_y \in (\downarrow^M_S (\beta^{crts}_{\mathcal{OS}\uparrow upd})\ominus \downarrow^M_S (\beta^{crts}_{\mathcal{OS}})) \wedge (R, R_y) \in\downarrow^M_S (\beta^{crcs}_{\mathcal{OS}\uparrow upd}))\}.\end{aligned}$$

Proof of Lemma 5.9. Split-safety of a role assertion $R(a_1, a_2)$ can be decided based on the current TBox, RBox and the concept assertions for individual a_1 and and individual a_2. The syntactic ontology updates $upd \,\|{+}C_{u1} \sqsubseteq C_{u2}\|$ and $upd \,\|{-}C_{u1} \sqsubseteq C_{u2}\|$ only change the TBox, i.e. all the RBox based structures and the ABox are unchanged after the update. The only exception is the introduction/removal of transitive roles. Thus, we only need to check all role assertions, such that

1. new sound concept subsumption relationships exist (or become obsolete),
2. new sound disjointness relationships exist (or become obsolete),
3. new potential ∀-propagations exist (or become obsolete), or
4. new transitive super roles exist (or become obsolete).

□

In Definition 5.35, we define a structure containing all individual pairs which are connected by a non-split-safe role assertion with respect to an updatable split decision system.

Definition 5.35 (Updatable Complete Split Dependency Structure)**:**
An *updatable complete split structure for an ontology state* $\mathcal{OS} = \langle \mathcal{O}, history \rangle$, denoted $\beta_{\mathcal{OS}}^{aboxspl}$, is a multiset over $\mathbf{IN} \times \mathbf{IN}$, defined inductively as follows:

- For the initial ontology state $\langle \langle \emptyset, \emptyset, \emptyset \rangle, \square \rangle$, let $\beta_{\langle \langle \emptyset,\emptyset,\emptyset \rangle,\square \rangle}^{aboxspl}(a_2, a_2) = 0$, for each $a_1 \in \mathbf{IN}$ and for each $a_2 \in \mathbf{IN}$.
- Given an updatable complete split dependency structure $\beta_{\langle \mathcal{O},history \rangle}^{aboxspl}$ and a syntactic ontology update *upd*, we let $\beta_{\mathcal{OS}\uparrow upd}^{aboxspl}$ be the result of the algorithm in Figure 5.2.

The implementation of the algorithm in Figure 5.2 is directly motivated by the upper split difference bounds proved for each syntactic ontology update.

In Theorem 5.2, we show that the proposed updatable complete split dependency structure is a complete split dependency structure for all ontology states.

Theorem 5.2 (Updatable Split Dependency Structure)**:**
For all ontology states $\mathcal{OS} = \langle \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle, history \rangle$, $\downarrow_S^M (\beta_{\mathcal{OS}}^{aboxspl})$ is a complete split dependency structure for $\mathcal{OS}$.

Proof of Theorem 5.2. To show: if $\exists R \in \mathbf{Rol}.R(a_1, a_2) \in \mathcal{A} \wedge R(a_1, a_2)$ is not $\mathcal{SHI}$-splittable with respect to $\langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle \implies (a_1, a_2) \in \downarrow_S^M (\beta_{\mathcal{OS}}^{aboxspl})$. By induction on the construction of ontology states:

- Induction base: For the initial ontology state the statement is true trivially, since $\mathcal{A} = \emptyset$.
- Induction step: By Lemma 5.5, Lemma 5.6, Lemma 5.7, Lemma 5.8 and Lemma 5.9, for each syntactic ontology update we recompute all potentially changed dependencies for the new ontology state.

□

5.5 Updatable Reasoning Structures

5.5.1 Updatable One-Step Node Map Structure

In the following, we define an updatable one-step node map structure, motivated by one-step node maps from Section 4.3. Please recall that one-step nodes are defined as efficient equality measures for individual islands. Furthermore, one-step nodes can be used as a kind of proxy structure to answer queries fast in a sound and possibly complete way.

Figure 5.2 Updating split dependency structures

Input: Updatable complete split dependency structure $\beta^{aboxspl}_{\mathcal{OS}}$, an updatable split decision system $updsds(\mathcal{OS}) = \langle FD(\beta^{stcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{stds}_{\mathcal{OS}}), \downarrow^M_S (\beta^{cfis}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crcs}_{\mathcal{OS}}), \downarrow^M_S (\beta^{crts}_{\mathcal{OS}}), \mathcal{A}_{\mathcal{OS}} \rangle$ syntactic ontology update upd

Output: Updatable complete split dependency structure $\beta^{aboxspl}_{\mathcal{OS}\uparrow upd}$

Algorithm:

Let toupdate $= \emptyset$

If $upd = upd \,\|{+}C_u(a_u)\|$ or $upd = upd \,\|{-}C_u(a_u)\|$ **then**

toupdate $= \{R_x(a_u, a_{x1}) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\} \cup \{R_x(a_{x1}, a_u) \mid R_x(a_u, a_{x1}) \in \mathcal{A}_{\mathcal{OS}}\}$

Else if $upd = upd \,\|{+}R_u(a_{u1}, a_{u2})\|$ or $upd = upd \,\|{-}R_u(a_{u1}, a_{u2})\|$ **then**

toupdate $= \{R_u(a_{u1}, a_{u2})\}$

Else if $upd = upd \,\|{+}R_{u1} \sqsubseteq R_{u2}\|$ or $upd = upd \,\|{-}R_{u1} \sqsubseteq R_{u2}\|$ **then**

toupdate $= \{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\}$, where **dr** is defined as in Lemma 5.7

Else if $upd = upd \,\|{+}Trans(R_u)\|$ or $upd = upd \,\|{-}Trans(R_u)\|$ **then**

toupdate $= \{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\}$, where **dr** is defined as in Lemma 5.8

Else if $upd = upd \,\|{+}C_{u1} \sqsubseteq C_{u2}\|$ or $upd = upd \,\|{-}C_{u1} \sqsubseteq C_{u2}\|$ **then**

toupdate $= \{R_x(a_{x1}, a_{x2}) \mid R_x(a_{x1}, a_{x2}) \in \mathcal{A}_{\mathcal{OS}} \wedge R_x \in \mathbf{dr}\} \cup \{R_x(a_{x1}, a_{x2}) \mid \{C \mid C(a_{x1}) \in \mathcal{A}_{\mathcal{OS}} \vee C(a_{x2}) \in \mathcal{A}_{\mathcal{OS}}\} \cap \mathbf{dc} \neq \emptyset\}$, where **dr** and **dc** are defined as in Lemma 5.9.

Let $\beta^{aboxspl}_{\mathcal{OS}\uparrow upd} = \beta^{aboxspl}_{\mathcal{OS}}$

For $R(a_1, a_2) \in$ **toupdate**

let $\beta^{aboxspl}_{\mathcal{OS}\uparrow upd}(a_1, a_2) = |\{R(a_1, a_2) \in \mathcal{A}_{\mathcal{OS}\uparrow upd} \mid R(a_1, a_2)$ is not split-safe with respect to $updsds(\mathcal{OS} \uparrow upd)\}|$

Definition 5.36 (Updatable One-Step Node Map Structure)**:**
An *updatable one-step node map structure for an ontology state* $\mathcal{OS} = \langle \mathcal{O}, history \rangle$, denoted $\beta^{osnmap}_{\mathcal{OS}}$, is a function $\beta^{osnmap}_{\mathcal{OS}} : \mathbf{IN} \to \mathbf{OSN}$, defined inductively as follows:

- For the initial ontology state $\langle\langle \emptyset, \emptyset, \emptyset \rangle, \Box \rangle$, let $\beta^{osnmap}_{\langle\langle \emptyset,\emptyset,\emptyset \rangle,\Box \rangle}(a) = n.d.$ for each $a \in \mathbf{IN}$.
- Given an updatable one-step node map structure $\beta^{osnmap}_{\mathcal{OS}}$ and an syntactic ontology update upd, we let $\beta^{osnmap}_{\mathcal{OS}\uparrow upd}$ be the result of the algorithm in Figure 5.3.

Given the definition of one-step nodes in Definition 4.17, the algorithm in Figure 5.3 is self explaining. Whenever we have a syntactic ABox update, then we recompute all the affected one-step nodes. For syntactic ABox concept assertion updates, we compute the one-step nodes for the updated individual (the individual occurs in the concept assertion axiom) and all its neighbors. For syntactic ABox role assertion updates, we compute the one-step nodes for both updated individuals (the individuals occur in the role assertion axiom). We use two sets in the algorithm: **toupdate** is used to manage all individuals whose one-step nodes need to be recomputed, and **todelete** contains all individuals which need to be removed from the one-step node map, because the ABox does not contain ABox assertions for them anymore.

Figure 5.3 Updating one-step node maps

Input: Updatable one-step node map structure $\beta^{osnmap}_{\langle \mathcal{O}, history\rangle}$, syntactic ontology update upd

Output: Updatable one-step node map structure $\beta^{osnmap}_{\langle upd(\mathcal{O}), upd \circ history\rangle}$

Algorithm:

Let toupdate $=$ **todelete** $= \emptyset$
Let $\langle \mathcal{T}_2, \mathcal{R}_2, \mathcal{A}_2\rangle = upd(\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle)$
If $upd = upd \,\|{+}C_u(a_u)\|$ or $upd = upd \,\|{-}C_u(a_u)\|$ **then**
  toupdate $=$ **toupdate** $\cup \{a_u\} \cup \{a \mid a$ `is a neighbor of` a_u `in` $\mathcal{A}_2\}$
Else if $upd = upd \,\|{+}R_u(a_{u1}, a_{u2})\|$ or $upd = upd \,\|{-}R_u(a_{u1}, a_{u2})\|$ **then**
  toupdate $= \{a_{u1}, a_{u1}\}$
todelete $= Ind(\mathcal{A}) \setminus Ind(\mathcal{A}_2)$
Let

$$\beta^{osnmap}_{\langle upd(\mathcal{O}), upd\circ history\rangle}(a) = \begin{cases} osn^{a,\mathcal{A}_2} & \text{if } a \in \textbf{toupdate} \wedge a \notin \textbf{todelete}, \\ n.d. & \text{if } a \in \textbf{todelete}, \\ \beta^{osnmap}_{\langle \mathcal{O}, history\rangle}(a) & \text{otherwise} \end{cases}$$

In Theorem 5.3, we show that the updatable one-step node map structure from Definition 5.36 is a one-step node map structure for each ontology state.

Theorem 5.3 (Updatable One-Step Node Map Structure)**:**
For each ontology state $\mathcal{OS} = \langle\langle \mathcal{T}, \mathcal{R}, \mathcal{A}\rangle, history\rangle$, the updatable one-step node map structure $\beta^{osnmap}_{\mathcal{OS}}$ is a sound and complete one-step node map structure for $\mathcal{OS}$.

Proof of Theorem 5.3. We have to show that for each named individual $a \in NInd(\mathcal{A})$, $\beta^{osnmap}_{\mathcal{OS}}(a) = osn^{a,\mathcal{A}}$. By induction on the construction of ontology states:

- Induction base: For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\square\rangle$, we have $\mathcal{A} = \emptyset$ and $NInd(\mathcal{A}) = \emptyset$.
- Induction step: Let $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle = upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle)$. We assume that for each individual $a \in NInd(\mathcal{A})$, $\beta^{osnmap}_{\mathcal{OS}}(a) = osn^{a,\mathcal{A}}$ and have to show that for each individual $a \in NInd(\mathcal{A}_2)$, $\beta^{osnmap}_{\langle upd(\mathcal{O}),upd\circ history\rangle}(a) = osn^{a,\mathcal{A}_2}$.

  The one-step nodes for an individual a are based on its concept assertion axioms, the concept assertion axioms of the direct neighbors and the role assertions for a. For each syntactic ontology update, we determine all concerned individuals and their neighbors in the set **toupdate**. Furthermore, if an individual is removed from the ABox, we also remove it from the updatable one-step node map.

□

5.5.2 Updatable Island Map Structure

In the following, we define an updatable island map structure, motivated by individual island maps from Section 4.3. Please recall that individual island maps are used for sound and complete instance checking and instance retrieval.

Definition 5.37 (Updatable Island Map Structure)**:**
An *updatable island map structure for an ontology state* $\mathcal{OS} = \langle\mathcal{O}, history\rangle$, denoted $\beta^{islmap}_{\mathcal{OS}}$, is a function $\beta^{islmap}_{\mathcal{OS}} : \mathbf{IN} \to \wp(\mathbf{IN})$, defined inductively as follows:

- For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\square\rangle$, let $\beta^{islmap}_{\langle\langle\emptyset,\emptyset,\emptyset\rangle,\square\rangle}(a) = \emptyset$ for each $a \in \mathbf{IN}$.
- Given an updatable island map structure $\beta^{islmap}_{\langle\mathcal{O},history\rangle}$ and an syntactic ontology update upd, we let $\beta^{islmap}_{\langle upd(\mathcal{O}),upd\circ history\rangle}$ be the result of the algorithm in Figure 5.4.

The algorithm in Figure 5.4 is self-explaining, given the results about updatable split decision systems. In each syntactic ontology update, we compute the set of all individuals, denoted **indaff**, for which a role assertion was changed from split-safe to non-split-safe, and vice versa. Then, for each individual island, which contains an assertion about an individual in **indaff**, we recompute the individual island and update the updatable island map structure.

Theorem 5.4 (Updatable Island Map Structure)**:**
For each ontology state $\mathcal{OS} = \langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle, history\rangle$, the updatable island map structure $\beta^{islmap}_{\mathcal{OS}}$ can be used as a sound and complete island map structure for $\mathcal{OS}$.

Proof of Theorem 5.4. We have to show that for each named individual $a \in NInd(\mathcal{A})$, we have that $\langle\mathcal{T},\mathcal{R}, LoadAs(\beta^{islmap}_{\mathcal{OS}}(a)), a\rangle$ is an individual island for $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle$, where

$$LoadAs(\mathbf{inds}) = \{C(a) \in \mathcal{A} \mid a \in \mathbf{inds}\} \cup \{R(a_1,a_2) \in \mathcal{A} \mid \{a_1,a_2\} \subseteq \mathbf{inds}\}.$$

Figure 5.4 Updating island maps

Input: Updatable island map structure $\beta^{islmap}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}$, syntactic ontology update upd, $\beta^{aboxspl}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}$ and $\beta^{aboxspl}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle}$

Output: Updatable island map structure $\beta^{islmap}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle}$

Algorithm:

Let $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle = upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle)$
Let diff $= \downarrow^M_S (\beta^{aboxspl}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}) \ominus \downarrow^M_S (\beta^{aboxspl}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle})$
Let indaff $= \{a_1 \mid \exists a_2 \in \mathbf{IN}.(a_1,a_2) \in \mathbf{diff} \vee (a_2,a_1) \in \mathbf{diff}\}$
Let inds$_{toRecompute}$ = **indaff** $\cup \{a \mid (\beta^{islmap}_{\langle\mathcal{O},history\rangle}(a) \cap \mathbf{indaff}) \neq \emptyset\}$
Set $\beta^{islmap}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle} = \beta^{islmap}_{\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle}$
For each $a \in$ **inds**$_{toRecompute}$

- Use the algorithm from Figure 4.1 to compute the individual island $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}^{isl},a\rangle$ for individual a with respect to ontology $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle$
- **Set** $\beta^{islmap}_{\langle upd(\mathcal{O}),upd\circ history\rangle}(a) = Ind(\mathcal{A}^{isl})$

By induction on the construction of ontology states:

- Induction base: For the initial ontology state $\langle\langle\emptyset,\emptyset,\emptyset\rangle,\Box\rangle$, we have $\mathcal{A} = \emptyset$ and thus $Ind(\mathcal{A}) = \emptyset$.
- Induction step:

  Let $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle = upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle)$. We assume that for each $a \in NInd(\mathcal{A})$, we have that $\langle\mathcal{T},\mathcal{R},LoadAs(\beta^{islmap}_{\mathcal{OS}}(a)),a\rangle$ is an individual island for $\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle$, and we have to show that for each named individual $a \in NInd(\mathcal{A}_2)$, we have that the structure $\langle\mathcal{T}_2,\mathcal{R}_2,LoadAs(\beta^{islmap}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle}(a)),a\rangle$ is an individual island for $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle$.

  Since the updatable complete split structure $\beta^{aboxspl}_{\mathcal{OS}}$ is a complete split dependency structure for the ontology state $\langle\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle,history\rangle$ and $\beta^{aboxspl}_{\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle}$ is a complete split dependency structure for $\langle upd(\langle\mathcal{T},\mathcal{R},\mathcal{A}\rangle),upd\circ history\rangle$, we can conclude that the set **diff** in the algorithm in Figure 5.4 contains a superset of all individual pairs whose split-safety changed during the syntactic ontology update. Since all islands with role assertions between individuals from **diff** are recomputed, we have for each individual $a \in NInd(\mathcal{A}_2)$, that $\langle\mathcal{T}_2,\mathcal{R}_2,LoadAs(\beta^{islmap}_{\mathcal{OS}}(a)),a\rangle$ is an individual island for $\langle\mathcal{T}_2,\mathcal{R}_2,\mathcal{A}_2\rangle$.

□

5.6 Concluding Remarks

In Chapter 5, we have defined updatable algorithms for the computation of individual islands and one-step nodes. We use a class of syntactic ontology updates in order to formalize addition and retraction of assertions from TBoxes, ABoxes, and RBoxes of an ontology. Although these kind of updates are not full updates, in the sense of model-based updates, we think that our proposal can be a first step towards reasoning over dynamic ontologies.

We have defined an abstract split decision system, which can be used to determine $\mathcal{SHI}$-splittable role assertions under syntactic ontology updates. The abstract split decision system uses unsound plus incomplete data structures. This allows us to improve efficiency of implementations for our update algorithms. Furthermore, we provide a concrete updatable instantiation of an abstract split decision system.

Updates on the data structures can still be quite time-consuming. While updates on the assertional part of the ontology have only local effects, the impact of a TBox or RBox axiom can be immense. For instance, assume there is an update on a role description in the upper part of the role hierarchy, in the worst case a universal role description (subsuming all other role descriptions). In this case, all role assertions in the ABox have to be reevaluated for $\mathcal{SHI}$-splittability. However, this cannot be avoided as long as soundness and completeness of reasoning has to be ensured.

Chapter 6: System Description and Evaluation

In Chapter 6, we describe a prototypical implementation of the algorithms introduced in order to improve performance of query answering over description logic ontologies. In particular, we implement a system which is based on syntactic ontology updates from Chapter 5. Furthermore, we evaluate our modularization techniques with respect to test ontologies.

In Section 6.1, we introduce a client-server architecture which is used to perform efficient instance checking and instance retrieval over ontologies in practice. We describe the internal data management in the prototype and implementation issues. In addition, we describe how query answering is performed.

In Section 6.2, we evaluate research hypothesis with respect to test ontologies. We show that our modularization techniques can be used in order to release the main memory dependency from description logic reasoning systems. We investigate granularity of modularizations as well as scalability of our updatable data structures. We use two ontologies for evaluation. We introduce a real world ontology which is used to evaluate our techniques with respect to real world domain knowledge. A benchmark ontology is used to show further applicability results and provide first scalability statistics for our updatable data structures.

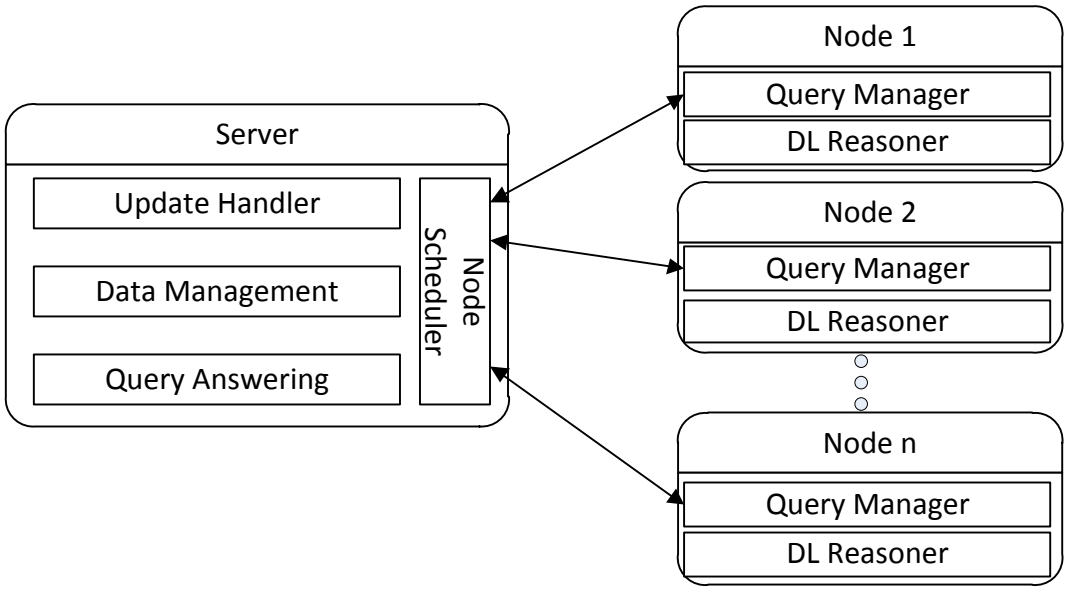
6.1 System Description

6.1.1 General Structure

In the following, we provide a general overview over the prototype. We have implemented the algorithms for updatable reasoning over $\mathcal{SHI}$-ontologies using the programming language Java. We used the description logic reasoner Racer [HMW04] for our evaluation.

The prototype consists of two parts: a server module and client modules (also called nodes). While the server module is used for updating the internal representation of the current ontology state, the client modules are used for reasoning. The structure of the modules is depicted in Figure 6.1. Modules are described in detail below.

Figure 6.1 Module structure of the system



6.1.2 Data Loading and Management

The update handler module contains the implementation for loading data sets and invokes all necessary updates on internal data structures. For both parts of the ontology, terminological part and assertional part, we enable different kinds of data sources as input. In the following, we always cover addition and retraction of facts if we mention *loading*.

For the terminological part, we provide an interface to load assertions from OWL files. The parsing of OWL files is performed with help of a Java API called OWLAPI, see [HB09] for details. Please note that the order of loading facts from an OWL file is determined by OWLAPI. Thus, if assertions should be loaded in a particular order, then these assertions have to be distributed over different OWL files. Furthermore, to the best of our knowledge, OWLAPI will always load the whole OWL file into main memory, even if OWLAPI is only used for parsing. This has to be kept in mind, when dealing with large ontologies. In addition, we have developed a SAX-based OWL parser, for OWL files which do not fit into main memory.

For the assertional part of an ontology, we allow loading data from three different data sources. First, loading from OWL files is supported as well. However, due to the constraints mentioned above, we allow two additional data sources.

We allow to load ABox assertions from text files in a comma separated value (CSV) format. An example for the format is given in Figure 6.2. Each line is preceded by an identifier CA or RA indicating, whether the assertion is a concept assertion axiom or a role assertion axiom. For concept assertion axioms, the second entry in a line is an

Figure 6.2 Example for a comma separated value input file

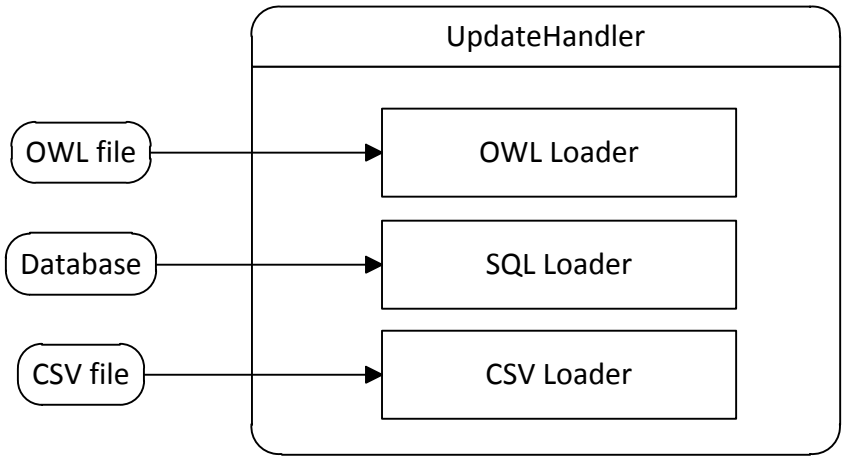
```
CA, ann , Professor
CA, eve , Professor
CA, c1 , UndergraduateCourse
CA, c2 , GraduateCourse
...
RA, teaches , ann , c1
RA, teaches , eve , c2
...
```

individual name and the third entry in a line is an atomic concept description. Negated concept names are denoted by `concept^-`. For role assertion axioms, the second entry is a role description, the third entry is the first individual name and the fourth entry is the second individual name of the role assertion axiom.

Furthermore, we implemented ABox assertion loading directly from a database. Concept assertion axioms and role assertion axioms can be loaded from tables respectively.

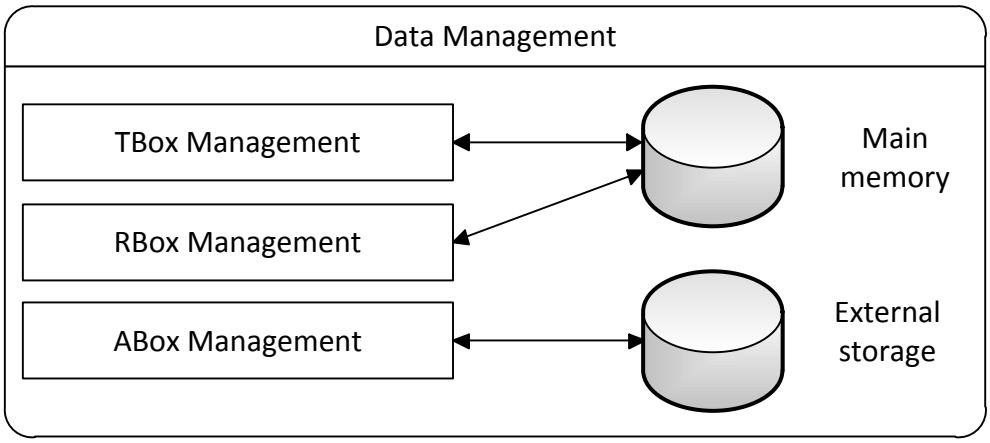
The interface of the update handler module is summarized in Figure 6.3. In order to extend our update handler module to different input formats, such as KRSS [PSS93] or KIF [GF92], one has to implement an interface for stepwise converting assertions into our in-memory representation.

Figure 6.3 Informal interface of the Update Handler module



The implementation of most of the updatable data structures from Chapter 5 is rather straightforward, given a library for multisets in Java that we have developed.

Figure 6.4 Structure of the Data Management module



In Figure 6.4, the general data management is depicted. While TBox and RBox are kept in main memory, the ABox is serialized to a database. In our prototypical implementation, we used the relational database management system MySQL, see [WA02]. Apart from the assertional data, we also serialize

- the identifiers of one-step nodes for each individual and
- information about splittability of role assertions.

For each serialized data structure, we have implemented caching algorithms, in order to avoid working on external memory directly for each update. During our experiments, a segmented least recently used cache, see for instance [KLW94], turned out to be most efficient. There exist two separate least recently used caches, one called *probationary* and the other one called *protected*.

Whenever an element is missed, i.e. the element cannot be found in the cache and has to be loaded from external memory, the element is put at the end of the probationary cache after loading. If there is a hit on an element, i.e. the element is already present in one of the two segments, then the element is moved to the end of the protected segment. Thus, elements which end up in the protected segment have been at least used twice. This seems like a good trade off between simple least recently used caching algorithms and (hard-to-implement) least often used caching algorithms. For details about the serialization of both caches see [KLW94].

6.1.3 Query Answering

The query answering process is performed in client modules. These clients (nodes) can run on distributed machines. All necessary assertions for reasoning on individuals are

distributed among these nodes. Each node determines local solutions, which are communicated back to the server. The server combines the pre-filtered results and decides about additional refine steps. The scenario is described in detail below.

6.1.3.1 Preparation Step

Before query answering can take place, we need to prepare some data structures and distribute them to client modules. In particular, we serialize all cached data, such as ABox cache and one-step node cache, to the disk. Furthermore, we determine the minimal set of distinct (with respect to similarity) one-step nodes.

This step is straightforward, since our implementation of one-step node maps in Java allows us to determine the range of the one-step node map function immediately. Thus, we have a set of n distinct once step nodes, which can be used for query answering over all named individuals.

Since reasoning over a one-step node is independent from other one-step nodes, the process can be easily parallelized. We use a set of m client modules, prepared in advance, such that each client module is capable of reasoning over one-step nodes. The server module sends $\frac{n}{m}$ one-step nodes to each client module. Each client module prepares an internal representation and sets up a local ontology with a representation of all local one-step nodes. In our experiments, we have used the description logic reasoning system Racer, in order to reason over one-step nodes with respect to terminological knowledge.

6.1.3.2 Instance Checking

Given an atomic concept description C and a named individual $a \in NInd(\mathcal{A})$, we have to find out, whether the current ontology state entails $C(a)$. First, we read off the one-step node for a, then we determine the responsible client module for the one-step node $osn^{a,\mathcal{A}}$ of a. The instance checking query for the one-step node is forwarded to the responsible client module.

The client module then determines, with help of a description logic reasoner, whether the corresponding one-step node entails C for the root individual, or the one-step node entails $\neg C$ for the root individual. In addition, the client module decides, whether the one-step node is splittable or not. Depending on the outcome, the client module performs the following:

- If $osn^{a,\mathcal{A}} \models_{\mathcal{T},\mathcal{R}} C(a)$, then the client module returns a message to the server indicating that a is an instance of concept description C.
- Else if $osn^{a,\mathcal{A}} \models_{\mathcal{T},\mathcal{R}} \neg C(a)$ or $osn^{a,\mathcal{A}} \not\models_{\mathcal{T},\mathcal{R}} C(a)$ and $osn^{a,\mathcal{A}}$ is splittable, then the client module returns a message to the server indicating that a is not an instance of concept description C.

- Otherwise, the client module loads the individual island of individual a from the server, performs instance checking on the individual island and then returns a message to the server about the outcome of the sound and complete instance check.

6.1.3.3 Instance Retrieval

Given an atomic concept description C, we have to find all named individuals which are an instance of C with respect to the current ontology state. The server module sends a request to all client modules to evaluate their one-step nodes and the associated individuals. Each client module determines locally three sets of individuals:

- $\mathbf{s}_{yes}$: Contains all named individuals $a \in NInd(\mathcal{A})$, such that $osn^{a,\mathcal{A}} \models_{\mathcal{T},\mathcal{R}} C(a)$.
- $\mathbf{s}_{no}$: Contains all named individuals $a \in NInd(\mathcal{A})$, such that $osn^{a,\mathcal{A}} \models_{\mathcal{T},\mathcal{R}} \neg C(a)$ or $osn^{a,\mathcal{A}} \not\models_{\mathcal{T},\mathcal{R}} C(a)$, such that $osn^{a,\mathcal{A}}$ is splittable.
- $\mathbf{s}_{open}$: $NInd(\mathcal{A}) \setminus (\mathbf{s}_{yes} \cup \mathbf{s}_{no})$.

The set $\mathbf{s}_{yes}$ contains obvious solutions, the set $\mathbf{s}_{no}$ contains obvious non-solutions, and all the individuals in $\mathbf{s}_{open}$ have to be checked further by individual instance checking in the client modules. Afterwards, the result, i.e a set of named individuals, is sent back to the server module. In the end, the server module computes the union of all answers from all client modules.

6.2 Evaluation

We have used two benchmark ontologies for evaluation of our modularization techniques: one synthetic benchmark introduced in [GPH05] and a real world multimedia annotation ontology used in the CASAM project and introduced in [GMN+09]. The results for both ontologies are outlined below.

6.2.1 LUBM

The Lehigh University Benchmark, short LUBM, is a synthetic ontology developed to benchmark knowledge base systems with respect to large OWL applications. The ontology is situated in the university domain. The background knowledge, i.e. the terminology, is described in a schema called Univ-Bench, see [GPH05] for an overview over the history, different versions and the predecessor Univ 1.0. The expressivity of the ontology is chosen to be in OWL Lite, which corresponds to the description logic $\mathcal{SHIF}$. However, the de facto expressivity is lower. For instance, the ontology does not introduce any cardinality/functionality expressions on roles.

The terminology defines 43 classes and 32 properties (including 25 object properties and 7 datatype properties). The datatype properties are ignored in our experiments. According to [TV03], this ontology can be categorized as a "description logic-style" ontology which has a moderate number of classes but several restrictions and properties per class. The terminology of LUBM is rather simple.

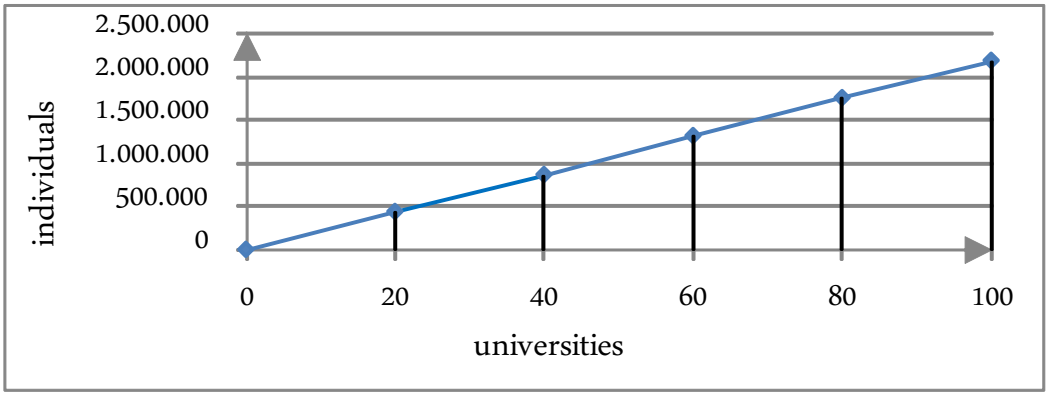
While the terminological part of LUBM is static, the assertional part is dynamic in size and can be generated as big as necessary/desired. There exists a small tool written in Java, called Univ-Bench Artificial Data Generator. Given a number n as input, the tool generates n different universities, containing information about individuals, e.g. students, professors, publications and courses. The basic unit of a University is a Department. The number of departments varies by university. To make data creation more random, one can manually set a seed number as input to the data generator.

The dataset we have used for our experiments was generated by the Univ-Bench artificial data generator with the following parameters:

```
java -classpath D:\research\ontologies\MYLUBM
    edu.lehigh.swat.bench.uba.Generator
    -univ 1000 -seed 0
    -onto file:/univ-bench.owl
```

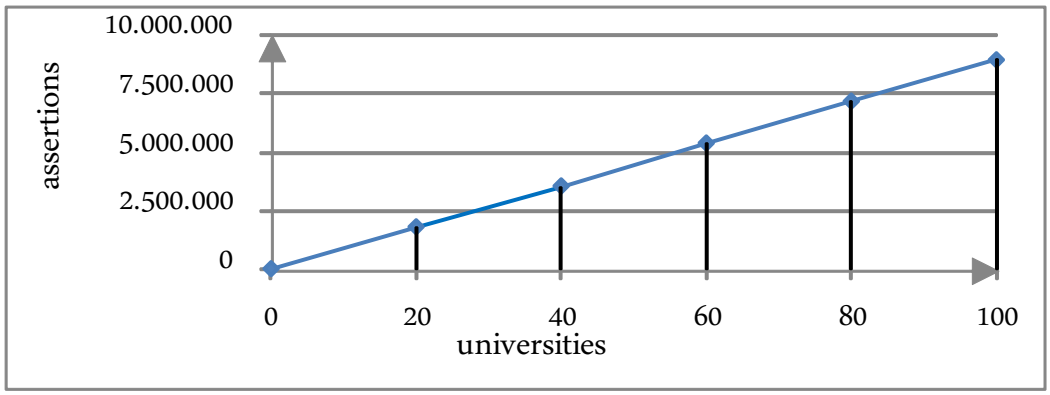
In Figure 6.5, we show the number of individuals in the dataset, for different numbers of universities. It can be seen that the number of individuals increases almost linearly with the number of universities. Around 30 percent of the individuals in the dataset are publications, another 30 percent are undergraduate students, 10 percent are graduate students, 10 percent are courses and graduate courses. The remaining 20 percent of the individuals are for instance professors, assistants and departments. For more details about the data distribution, see [GPH05].

Figure 6.5 Number of individuals in LUBM



In Figure 6.6, the number of ABox assertions is shown. Most of the role assertions in the ontology cover the enrollment into a course (around 45 percent of the role assertions), being a publication author (around 22 percent of the role assertions) or being a member of an organization (around 15 percent of the role assertions). The remaining role assertions cover facts like, for instance, teaching a course or having a master degree from a university.

Figure 6.6 Number of ABox assertions in LUBM

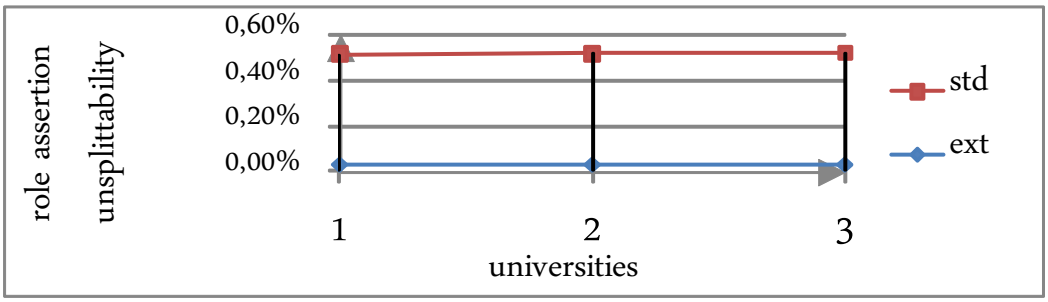


Since the number of individuals, as well as the number of ABox assertions is almost linear in the number of universities, LUBM seems like an adequate general, synthetic basis for instance checking and instance retrieval queries. LUBM is especially useful to evaluate scalability of our algorithms with a growing size of the ABox.

In the following, we investigate the efficiency of ABox modularization techniques from Chapter 3. The most important measure for efficiency seems to be the amount of $\mathcal{SHI}$-splittable role assertions, i.e. how many of the role assertions can be broken up. First of all, please note that component-based modularization of the assertional LUBM dataset yields one big module, i.e. each individual is connected to each other individual by a chain of role assertions. This is true for any number of universities. The connection between different universities is mainly because of degree-relationships between people and universities. Since only one ABox module is obtained, we do not provide any further statistics for component-based modularization.

The results for $\mathcal{SHI}$-splittability (from Definition 3.18) with respect to LUBM are shown in Figure 6.7 with the label *std*. The dataset for LUBM 1, i.e. only one university, contains 49336 role assertions, out of which 49082 are $\mathcal{SHI}$-splittable. This means that only 0.5 percent of the role assertions are $\mathcal{SHI}$-unsplittable. This ratio does not change with a growing number of universities. Almost all $\mathcal{SHI}$-unsplittable role assertions have transitive roles, e.g. the role $suborganizationOf$. In addition, role assertions with the role $headOf$ are also $\mathcal{SHI}$-unsplittable, since, for instance, the not obvious concept description $Chair$ can be propagated.

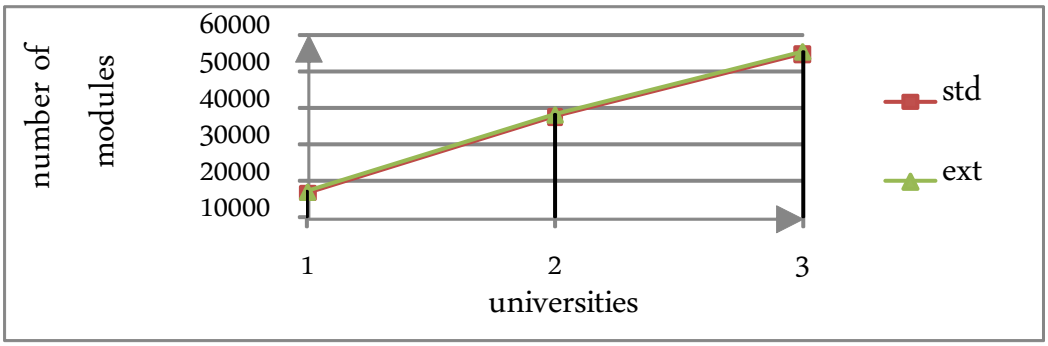
Figure 6.7 Percentage of unsplittable role assertions in LUBM



We have investigated an extended $\mathcal{SHI}$-splittability criteria, such that role assertions with transitive roles are splittable if all propagated concept descriptions are enforced by simple domain- or range-restrictions. In this case, without further proof, role assertions over transitive roles can be split up as well. The result for this extended splittability criterion are shown in Figure 6.7 with the label *ext*. For the extended criterion and one university, only 15 role assertions (out of 49336, 0.03 percent) turn out to be unsplittable. All these 15 role assertions contain the role $headOf$. For more universities, the ratio of unsplittable role assertions remains the same, since each department introduces exactly one head of the department.

Given the set of splittable role assertions, we can determine the number of ABox modules for different LUBM datasets. The results are shown in Figure 6.8.

Figure 6.8 Number of modules in LUBM

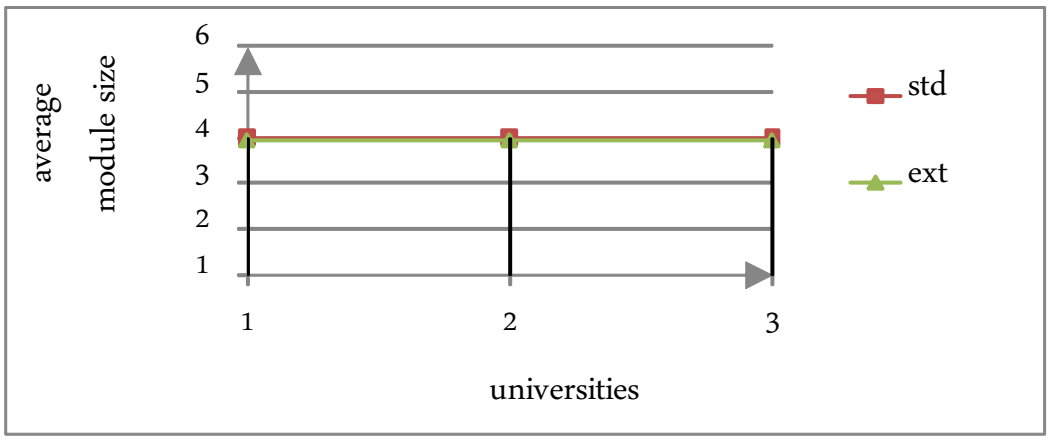


For component-based modularization, one big module is obtained, since each individual is related to each other individual by a chain of role assertions. With respect to $\mathcal{SHI}$-splittability, we obtain 16920 modules for one university and 37748 modules for two universities. With the extended criterion for splittability, i.e. improved handling of transitive roles, the number of modules can be further increased, as expected. For instance, for one university we obtain 17159 modules, instead of 16920. Please remember that the

number of individuals in one university is 17174. Each ABox module contains in average 1.01 individuals.

In order to further evaluate the quality of ABox modules, we show the average size (measured in number of ABox assertions) of the modules in Figure 6.9. For component-based modularization, the module is as big as the whole ABox (not shown). With respect to $\mathcal{SHI}$-splittability, the average size is between three and four ABox assertions per ABox module.

Figure 6.9 Average size of modules in LUBM



Next, we evaluated the number of distinct one-step nodes for different number of universities. The result is shown in Figure 6.10. Please recall that we compute a one-step node for each individual and then use a similarity relation among one-step nodes to group one-step nodes together. It can be seen that the number of distinct one-step nodes is relatively constant - compared to the linear number of individuals. Thus, in this case, one-step nodes can indeed be a candidate for a proxy reasoning structure, which exploits the local similarity to group individuals.

In Figure 6.10, we show the percentage of distinct complete one-step nodes as well. Around 99 percent of the one-step nodes are complete for each LUBM dataset. Thus, for 99 percent of the one-step nodes we can perform sound and complete reasoning over individuals directly.

Another evaluation measure is load time. The load time covers loading data from external memory (here: CSV files), applying the update algorithms introduced in Chapter 5 and serializing the data to a database representation. We process the terminological part first and afterwards the assertional part is loaded.

Figure 6.10 Number of distinct one-step nodes for LUBM

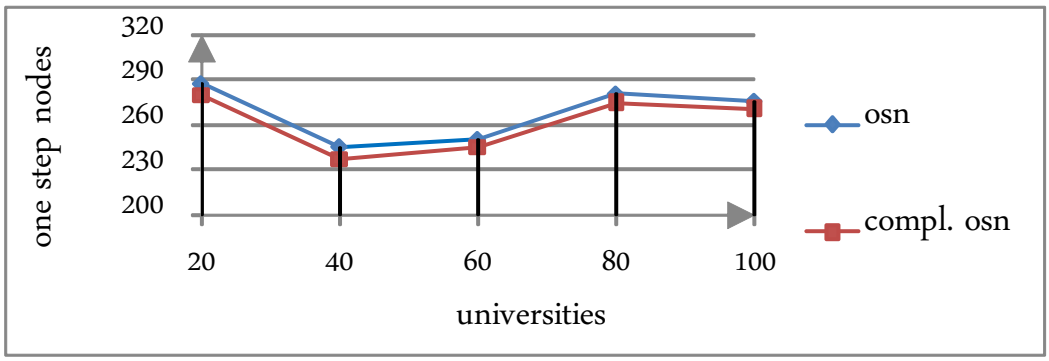
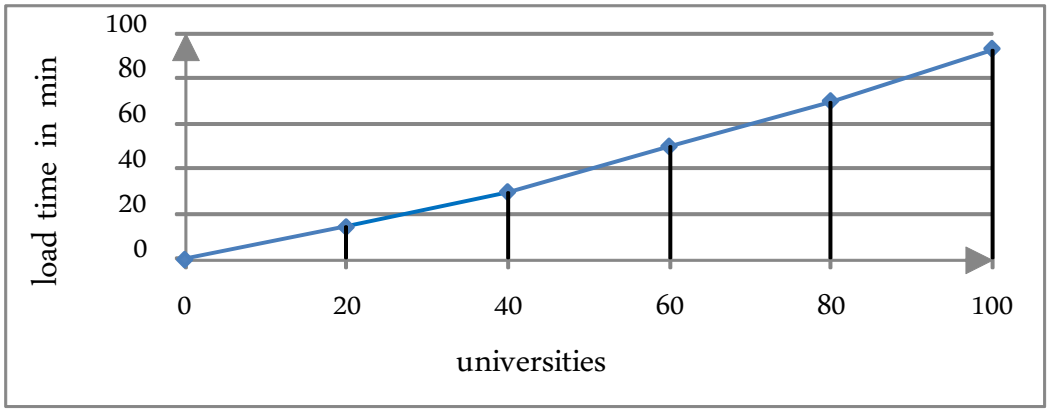


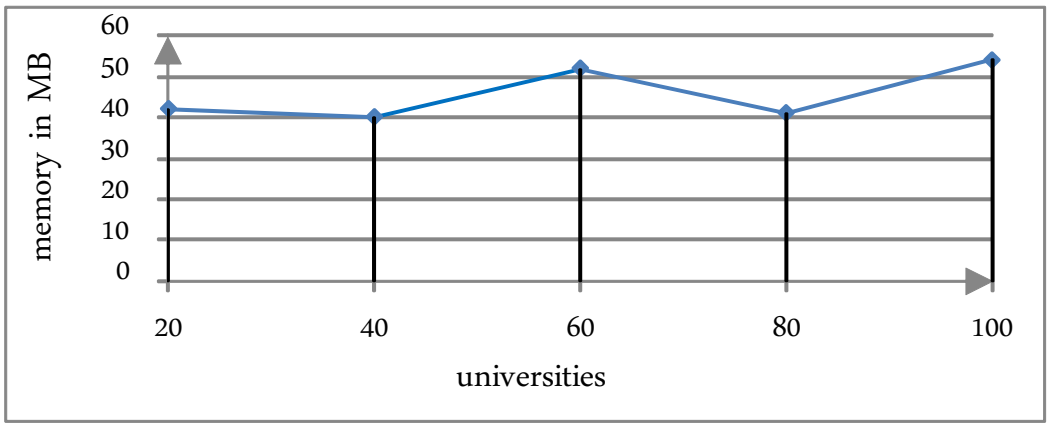
Figure 6.11 Load time for LUBM



The load time (using incremental updates) is shown for different numbers of universities in Figure 6.11. We add the ABox assertions from each university step by step. It can be seen that the curve is almost linear. We conjecture two main reasons for non-linearity:

1. The underlying database implementation cannot guarantee linear-time inserts. This is probably because the relational database system needs to update all indices for each update. While there exist optimization techniques, such as index update delay for bulk insertions, these techniques only perform well if the system does not read data during the bulk update phase. However, during syntactic ontology updates our algorithms work directly on the database representation (apart from caching).
2. For some assertional ABox updates, the system needs to reload all assertions for an individual from the database. Some role assertion sets for individuals, e.g. for all the *University*-individuals, grow linearly over time. Thus, single updates can already become more expensive with a growing number of universities.

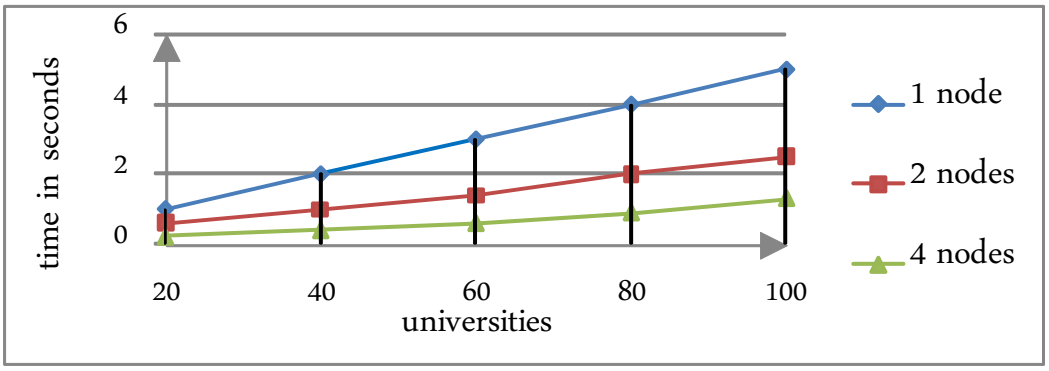
Figure 6.12 Main memory used for loading LUBM



The main memory used by the prototypical implementation during the loading phase is shown in Figure 6.12. Since we have implemented our prototype with the Java programming language, statistics with respect to main memory are easily misunderstood. Java uses the concept of garbage collection, where objects are not deallocated directly. There exists a separate thread in the background which keeps track of all object references. Whenever an object is not referenced anymore, the garbage collector might decide, once active, to free the memory used by the object.

The programmer (program) has almost no control over the garbage collection process. Thus, any memory snapshot obtained might not represent the total amount of used memory. The results in Figure 6.12 show that our implementation uses around 45 MB during the load process. This memory is used for caching ABox assertions, one-step nodes and the terminological part in main memory.

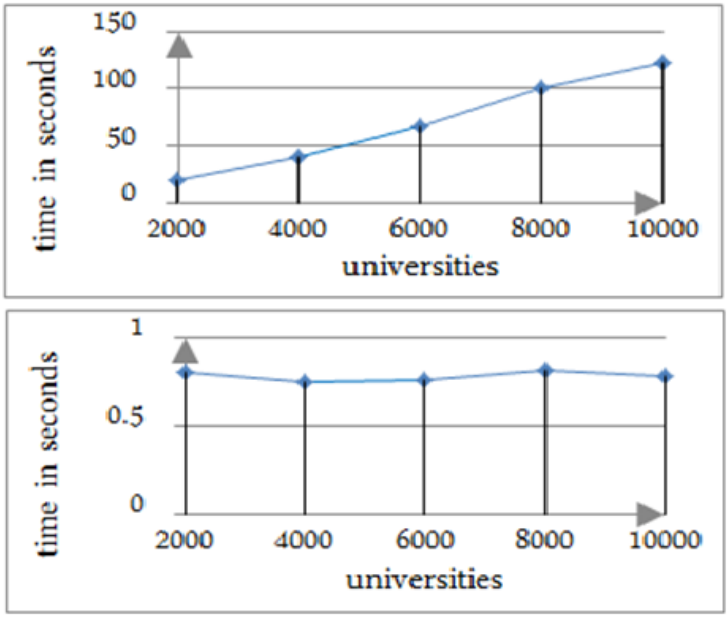
Figure 6.13 Time for instance retrieval for *Chair* and different number of nodes



In Figure 6.13, we show the instance retrieval time for the concept description *Chair* and different numbers of universities. It can be seen that for one client node, the instance retrieval time grows linearly with the number of universities. Once we increase the number of client nodes, the instance retrieval time is reduced. Since the initial filter step can be distributed easily - because of the independence of distinct one step nodes - the retrieval time for four nodes is almost $\frac{1}{4}$ compared to one node. Retrieval tests for other atomic concept descriptions yield similar results.

In Figure 6.14, we show instance retrieval times for the concept descriptions *Chair* (top) and *University* (bottom) for up to 10000 universities (one node only). For retrieving all the *Chair*-instances from the ontology, the time increases linearly with the number of universities (linear number of answers). The query answering time for retrieving all the *University*-instances is constant (constant number of answers). Please note that our experiments indicate that the underlying database system is dominating the instance retrieval time for the *Chair*-query. The actual determination of the solution islands takes less than one second even for 10000 universities.

Figure 6.14 Instance retrieval times for LUBM 10000



6.2.2 CASAM Multimedia Content Ontology

The CASAM project is focused on computer-aided semantic annotation of multimedia content. The novelty is the aggregation of human and machine knowledge. For a detailed discussion of the research objectives, see [GMN+10], [PTP10], and [CLHB10]. Within

Figure 6.15 Excerpt of the MCO concept classification

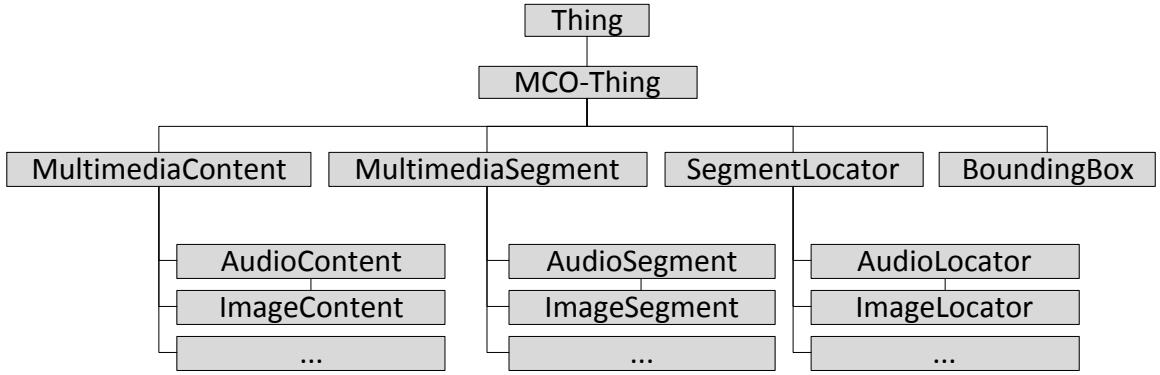
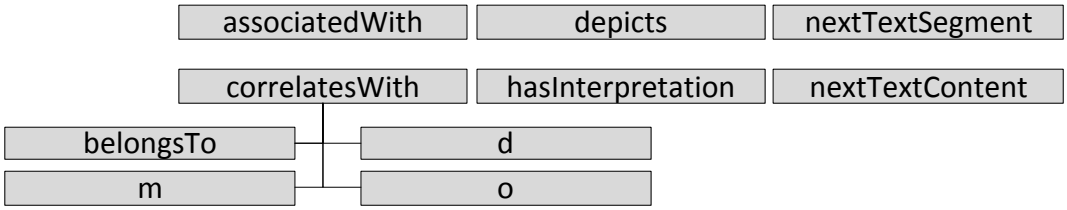


Figure 6.16 Excerpt of the MCO role classification

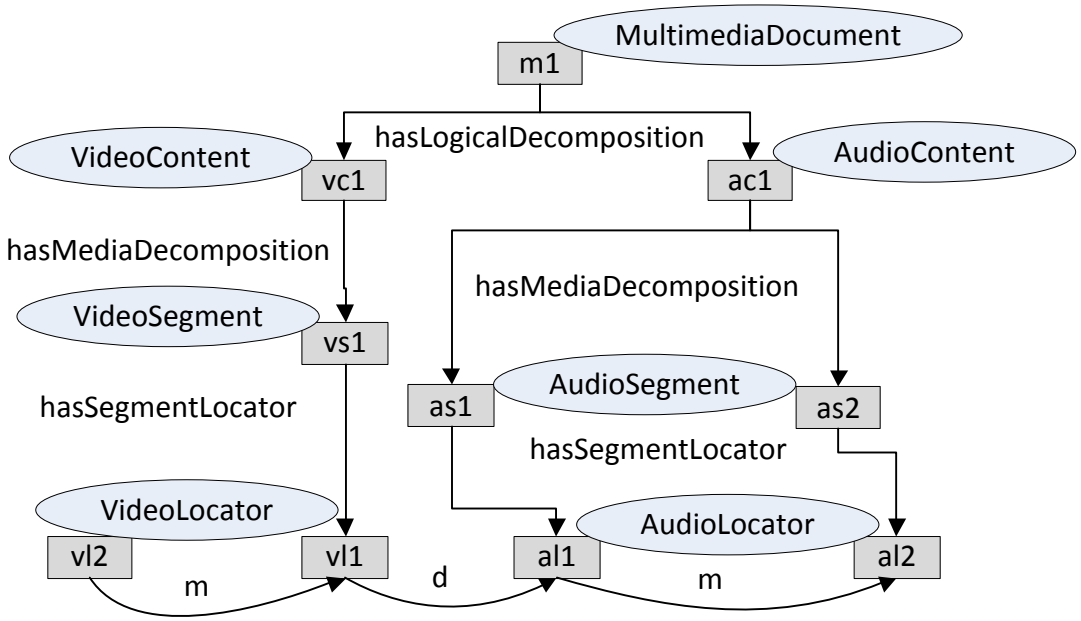


the CASAM project, there is a need to define an expressive annotation language which allows for typical-case reasoning systems. The proposed annotation language is defined by the so-called Multimedia Content Ontology, short MCO, introduced in [GMN+09]. Inspired by the MPEG-7 standard, see [IF02], strictly necessary elements describing the structure of multimedia documents are extracted. The intention is to exploit quantitative and qualitative time information in order to relate co-occurring observations about events in videos. Co-occurrences are detected either within the same or between different modalities, i.e. text, audio and speech, regarding the video shots.

In the following, we present small excerpts of MCO as far as relevant for understanding our evaluation results. A part of the concept classification is shown in Figure 6.15.

An excerpt of the role classification is shown in Figure 6.16. The role descriptions are used to relate multimedia objects with each other. Please note that role description *correlatesWith* and its subroles are used to represent quantitative information as qualitative information. The roles d, m, and o are derived from the Allen-relations [All83], and represent *disjoint*, *meets*, and *overlapping* relations, respectively. The role descriptions *depicts* and *hasInterpretation* map individuals of the MCO to observations/elements of an analysis module. Different interpretations are related for instance by the role description *associatedWith*. For more details about MCO please refer to [GMN+09].

Figure 6.17 MCO ABox example

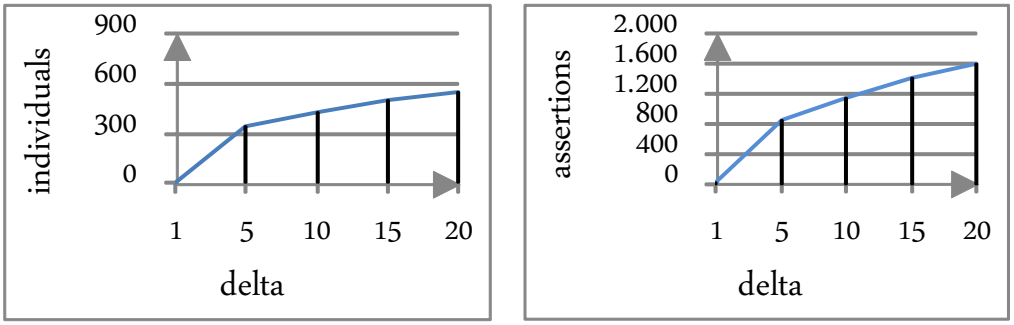


An excerpt of a multimedia document described with MCO is depicted in Figure 6.17. The ABox excerpt contains the description of a multimedia document $m1$, which has video and audio content. The video content, named $vc1$, has a video segment $vs1$. The audio content, named $ac1$, is decomposed into several audio segments, such as $as1$ and $as2$. Each segment is associated with a locator and the locators are related by qualitative spatial/temporal relations.

For our evaluation with respect to MCO, we have a number of multimedia documents from the CASAM project. The source ontologies can be found in [CAS10]. The set of test ontologies contains documents with identifiers ranging from 1 to 14. Each document is decomposed into several so-called *delta* files. Each delta represents additional information about the document of concern. We evaluated our modularization techniques with respect to all documents. Here we only show the results for Document 1, since for all the other documents we obtained very similar statistics.

In Figure 6.18, we show the number of individuals in the dataset, with an increasing delta. It can be seen that most individuals are introduced in the first delta files. The remaining delta files only introduce additional ABox assertions about already known individuals. The number of ABox assertions for different delta is also shown in Figure 6.18. Please note that the number of individuals, as well as the number of ABox assertions is not linear in the number of delta. Thus, a MCO document cannot be directly used for evaluation purposes. At least one would have to consider the number of individuals up to the delta to extract more clear scalability results.

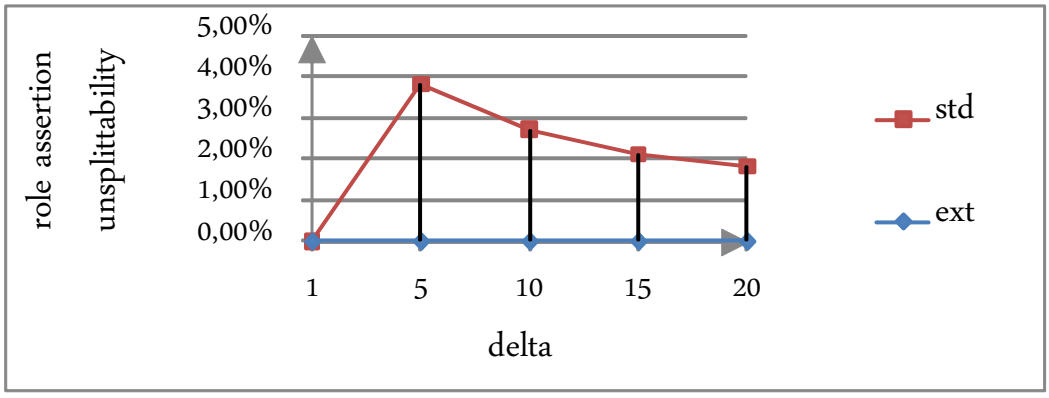
Figure 6.18 Number of individuals and ABox assertions in Document 1



In the following, we investigate the efficiency of ABox modularization techniques from Chapter 3. First of all, please note that component-based modularization of Document 1 yields one big module. This is true for all the other documents as well. Since only one ABox module is obtained, we do not provide any further statistics for component-based modularization.

The results for $\mathcal{SHI}$-splittability (from Definition 3.18) with respect to MCO Document 1 are shown in Figure 6.19 with the label *std*. The dataset for delta 1 contains 524 role assertions, out of which 504 are $\mathcal{SHI}$-splittable. This means that only 3 percent of the role assertions are $\mathcal{SHI}$-unsplittable. This ratio decreases with a growing number of deltas, because only $\mathcal{SHI}$-splittable role assertions are added. All $\mathcal{SHI}$-unsplittable role assertions have the transitive role $nextTextContent$. In addition, no other kinds of role assertions are $\mathcal{SHI}$-unsplittable.

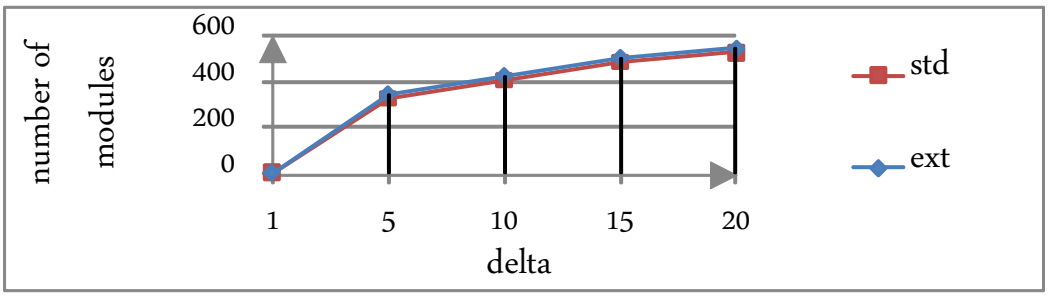
Figure 6.19 Percentage of unsplittable role assertions in Document 1



Again, we have investigated an extended $\mathcal{SHI}$-splittability criteria, such that role assertions with transitive roles are splittable if all propagated concept descriptions are enforced

by simple domain- or range-restrictions. The result for the extended splittability criterion are shown in Figure 6.19 with the label *ext*. For the extended criterion and any delta, no more role assertion is unsplittable, i.e. all role assertions in the ontology can be split up.

Figure 6.20 Number of modules in Document 1



Given the set of splittable role assertions, we can determine the number of ABox modules for different delta. The results are shown in Figure 6.20. For component-based modularization, one big module is obtained, since each individual is related to each other individual by a chain of role assertions. With respect to $\mathcal{SHI}$-splittability, we obtain 326 modules for five delta and 545 modules for 20 delta. With the extended criterion for splittability, i.e. improved handling of transitive roles, the number of modules can be further increased, as expected. For instance, for five delta we obtain 346 modules, instead of 326. Please remember that the number of individuals with five delta is 346. Each ABox module contains in average one individual.

In order to further evaluate the quality of ABox modules, we show the average size (measured in number of ABox assertions) of the modules in Figure 6.21. For component-based modularization, the module is as big as the whole ABox (not shown). With respect to $\mathcal{SHI}$-splittability, the average size is between two and three ABox assertions per ABox module.

Figure 6.21 Average size of modules in Document 1

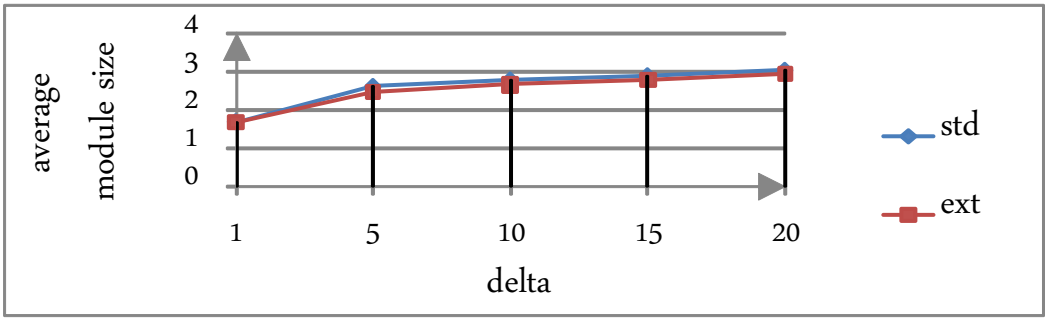
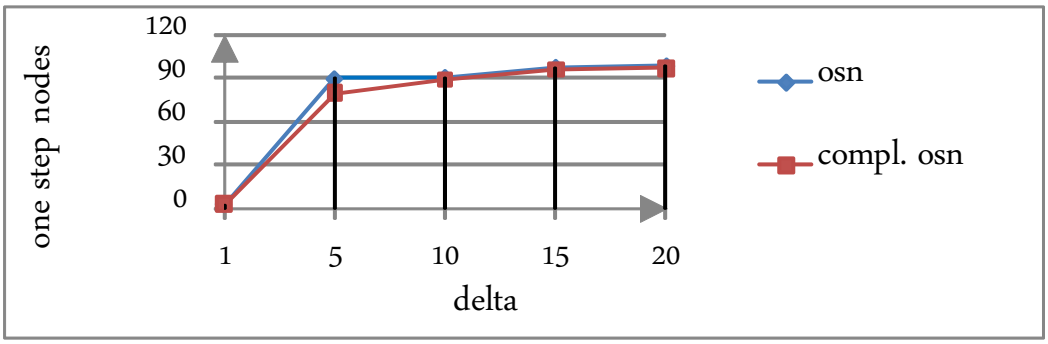


Figure 6.22 Number of distinct one-step nodes for Document 1



Next, we have evaluated the number of distinct one-step nodes for different number of delta. The result is shown in Figure 6.22. It can be seen that the number of distinct one-step nodes is relatively constant - after most individuals are introduced in the third and fourth delta. Thus, in this case, one-step nodes can indeed be a candidate for a proxy reasoning structure, which exploits the local similarity to group individuals. In Figure 6.22, we show the percentage of distinct complete one-step nodes as well. Around 98 percent of the one-step nodes are complete for each LUBM delta. Thus, for 98 percent of the one-step nodes we can perform sound and complete reasoning over individuals directly.

We do not provide any scalability statistics for the MCO ontology here, since the number of assertions is so small that one document is loaded within few seconds by our prototypical implementation.

Chapter 7: Conclusions

The main goal of this thesis was to release the main memory dependency from tableau-based description logic reasoning systems. We focused on the semi-expressive description logic $\mathcal{SHI}$, which can be seen as a first step towards more expressive description logics.

The modularization of the assertional part of ontologies was our main idea. We have derived a criterion, called $\mathcal{SHI}$-splittability, for the modularization of the ABox of an input ontology. The main technique used for modularization of ABoxes are ABox splits, which break up an role assertions in an ABox if particular conditions are satisfied. Role assertions can be broken up if, for instance, only obvious information is propagated. A graph component-based modularization can be used to extract a set of modules out of the ABox after breaking up all $\mathcal{SHI}$-splittable role assertions. Traditional description logic algorithms can then be used to reason over these ABox modules.

One might think that additional axioms in an ontology always makes reasoning more hard. An interesting side effect is that our modularization techniques show that additional axioms in the TBox can help to reduce the average size of ABox modules, and thus, can reduce instance checking and instance retrieval times.

Based on modularization techniques, we have introduced the notion of individual islands for individuals in ABoxes. These individual islands can be used for sound and complete instance checking. Our evaluation shows that these individual islands are usually quite small and fit into main memory.

In order to improve instance retrieval over description logic ontologies, we have introduced a similarity measure over individual islands. Similarity among individual islands can be used to significantly reduce the number of instance checks necessary for instance retrieval. Our evaluation showed that many individual islands are similar to each other - for a synthetic benchmark ontology as well as a real world multimedia ontology.

Please note that similarity of two individual (islands) is not only a means to optimize instance retrieval. It could also be used to create statistics about individuals in an ontology, i.e., create a kind of summarization. These summarizations can be even more important in stream-like data processing, such that differences between summarizations can give the user details about differences of ontology (ABox) snapshots.

We think that in the future updates over ontologies and stream-like processing of ontologies will become more important. Therefore, we have provided updatable data structures which enable incremental modularization of the assertional part of ontologies. Our evaluation showed that our prototypical implementation scales for the test ontologies. Last but not least, we have described our prototypical implementation in detail.

During the creation of the thesis, additional related work was published. We think that this work underlines the importance of our results and that our contribution is still valuable to the research community. We summarize some of the related work below.

In [LW10], the authors investigate inseparability of ontologies with respect to a given signature. This technique, for the lightweight description logic $\mathcal{EL}$, can possibly be used in order to extract modules from ontologies and also in order to define similarity of modules with respect to a signature. In [KLPW10], the authors apply similar techniques in order to define decompositions of ontologies. It is shown that the decomposition is tractable for the description logic $\mathcal{EL}$ and not more complex than concept subsumption for more expressive description logics. The main difference to our results is that we focused on ABox modularization directly for semi-expressive ontologies and use pure syntactical analysis in order to define modules (or decompositions). Furthermore, we have already considered updating modularizations.

In [TL10], the authors propose an index data structure for RDF data. The intention is to find similarities over instances in the RDF dataset by using bisimulations, i.e. something quite similar to our approach based on graph homomorphisms. The authors group bisimilar graph substructures, in order to reduce the complexity of query answering. The main difference to our modularization techniques is that we take the terminology into account for modularizing ABoxes. And then we try to find similarities among the (already small) modules. Furthermore, we focus on a semi-expressive description logic - not a plain graph structure. Updating the structure index is still open work in [TL10].

Another line of research leads to the approximation of reasoning over description logic ontologies. In [TGH10], it is discussed how approximate reasoning techniques can be used for more efficient instance retrieval. The main contribution of [TGH10] is the extension of efficient instance retrieval techniques for non-atomic concept assertions. They use so-called *approximate extensions* in order to reduce instance retrieval to atomic concepts. In [RPZ10], approximation of very expressive TBoxes is discussed. The authors convert input ontologies into $\mathcal{EL}^{++}$-ontologies and then use reasoning results over these $\mathcal{EL}^{++}$-ontologies for reasoning over the input ontology.

We think that these recent developments confirm the research goals stated at the beginning of this thesis. Due to the increased interest in the development of Semantic Web applications, efficient instance retrieval and query answering becomes more important every day. Our work can be seen as another step towards more efficient reasoning on the Semantic Web.

In the following, we would like to discuss interesting directions for future work. The effectiveness of our modularization techniques can be further improved. For instance, TBox modularization techniques can contribute to smaller ABox modularizations. If we are able to split up the TBox into different modules, we could create one ABox modularization for each TBox module. Since each TBox module only contains a subset of assertions from the original TBox, it is clear that additional role assertions become $\mathcal{SHI}$-splittable. However, it needs to be shown whether the overhead of several ABox modularizations in parallel, one for each TBox module, actually pays off. In addition, we think that further

optimizations of our modularization techniques are possible. So far, we focused on the entailment of atomic concept descriptions. The number of $\mathcal{SHI}$-splittable role assertions might increase if the vocabulary is known and restricted beforehand.

An extension from the semi-expressive description logic $\mathcal{SHI}$ to $\mathcal{SHIQ}$ should be possible. Although our proof techniques are not directly applicable, we think that a syntactical analysis of the TBox and RBox can be used to identify a set of $\mathcal{SHIQ}$-unsplittable role assertions. Our homomorphism-based similarity criteria for individuals cannot be directly applied in the presence of cardinality restrictions. Further extensions, for instance to $\mathcal{SHOIQ}$, might be possible, but will surely require a lot of work and sophisticated analysis techniques.

Another direction for future work is the focus on more expressive query languages. While we focus on instance checking and instance retrieval, the next natural step is conjunctive query answering [GHLS07]. We think that query answering with respect to the class of grounded conjunctive queries, i.e. solutions are only bound to named individuals in the ABox, is straightforward. One would have to combine the results from sound (and complete reasoning) in order to identify possible variable bindings. The extension to standard conjunctive queries is without doubt much harder.

Since rules over ontologies have become more important recently, it would be interesting to implement a rule-based query answering engine on top of our ABox modularizations. We already performed first tests. By syntactical analysis of rule bodies we decided which individual islands have to be extended/merged. The first results are quite encouraging.

Finally, more comprehensive experimental studies are required. Recently published work [SCH10] on new data generation algorithms for synthetic test ontologies might be a good place to start from. In general, we believe that our results carry over to other ontologies. However there exist scenarios, especially extensive use of transitive roles, which make it much harder to find fine-grained ABox modularizations.

References

[ACKZ09] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite Family and Relations. *J. Artif. Intell. Res. (JAIR)*, 36:1–69, 2009.

[AFWZ02] Alessandro Artale, Enrico Franconi, Frank Wolter, and Michael Zakharyaschev. A Temporal Description Logic for Reasoning over Conceptual Schemas and Queries. In *Proceedings of the European Conference on Logics in Artificial Intelligence*, JELIA '02, pages 98–110, London, UK, 2002. Springer-Verlag.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[Baa90] Franz Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 621–626. AAAI Press, 1990.

[Baa96] Franz Baader. Using Automata Theory for Characterizing the Semantics of Terminological Cycles. *Annals of Mathematics and Artificial Intelligence*, 18(2–4):175–219, 1996.

[Baa99] Franz Baader. Logic-Based Knowledge Representation. In *Artificial Intelligence Today*, pages 13–41. Springer-Verlag, 1999.

[Baa02] Franz Baader. Terminological Cycles in a Description Logic with Existential Restrictions. LTCS-Report LTCS-02-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2002. See http://lat.inf.tu-dresden.de/research/reports.html.

[BBH96] Franz Baader, Martin Buchheit, and Bernhard Hollander. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1-2):195–213, 1996.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[BBL08] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.

[BBM+92] Ronald J. Brachman, Alexander Borgida, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lori Alperin Resnick. The CLASSIC Knowledge Representation System or, KL-ONE: The Next Generation. In *FGCS*, pages 1036–1043, 1992.

[BCM+07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, New York, NY, USA, 2007.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). www.w3.org/TR/REC-rdf-syntax/, 2004.

[BH91] Franz Baader and Bernhard Hollunder. KRIS: Knowledge Representation and Inference System. *SIGART Bulletin*, 2(3):8–14, 1991.

[BHLW03] F. Baader, J. Hladik, C. Lutz, and F. Wolter. From tableaux to automata for description logics. In Moshe Vardi and Andrei Voronkov, editors, *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2003)*, volume 2850 of *Lecture Notes in Computer Science*, pages 1–32. Springer, 2003.

[BHS05] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics as Ontology Languages for the Semantic Web. In Dieter Hutter and Werner Stephan, editors, *Mechanizing Mathematical Reasoning*, volume 2605 of *Lecture Notes in Computer Science*, pages 228–248. Springer, 2005.

[BKM99] Franz Baader, Ralf Küsters, and Ralf Molitor. Computing least common subsumers in description logics with existential restrictions. In *IJCAI'99: Proceedings of the 16th International Joint Conference on Artifical Intelligence*, pages 96–101, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[BKvH03] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In Dieter Fensel, James A. Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web*, pages 197–222. MIT Press, 2003.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[Bli89] Wayne D. Blizard. Multiset Theory. *Notre Dame Journal of Formal Logic*, 30(1):36–66, 1989.

[BM07] Dan Brickley and Libby Miller. The Friend Of A Friend (FOAF) vocabulary specification. http://xmlns.com/foaf/spec/, November 2007.

[BS85] Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2):171–216, 1985.

[BS00] F. Baader and U. Sattler. Tableau Algorithms for Description Logics. In R. Dyckhoff, editor, *Proceedings of the International Conference on Automated Reasoning with Tableaux and Related Methods (Tableaux 2000)*, volume 1847 of *Lecture Notes in Artificial Intelligence*, pages 1–18, St Andrews, Scotland, UK, 2000. Springer-Verlag.

[BS01] Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, 69(1):5–40, 2001.

[BS03] Alexander Borgida and Luciano Serafini. Distributed Description Logics: Assimilating Information from Peer Sources. *J. Data Semantics*, 1:153–184, 2003.

[BW97] Alexander Borgida and Grant E. Weddell. Adding Uniqueness Constraints to Description Logics (Preliminary Report). In François Bry, Raghu Ramakrishnan, and Kotagiri Ramamohanarao, editors, *DOOD*, volume 1341 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 1997.

[CAS10] Test Documents CASAM. MCO test documents. http://http://www.sts.tu-harburg.de/~wandelt/casamtest.zip, 2010.

[CDGL+05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pages 602–607, 2005.

[CdK08] Ronald Cornet and Nicolette de Keizer. Forty years of SNOMED: A literature review. *BMC Med Inform Decis Mak*, 8 Suppl 1:S2, 2008.

[CLHB10] Chris Creed, Peter Lonsdale, Robert Hendley, and Russell Beale. Synergistic annotation of multimedia content. In *Proceedings of the 2010 Third International Conference on Advances in Computer-Human Interactions*, ACHI '10, pages 205–208, Washington, DC, USA, 2010. IEEE Computer Society.

[DFK+07] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Aaron Kershenbaum, Edith Schonberg, Kavitha Srinivas, and Li Ma. Scalable semantic retrieval through summarization and refinement. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pages 299–304. AAAI Press, 2007.

[DFK+09] Julian Dolby, Achille Fokoue, Aditya Kalyanpur, Edith Schonberg, and Kavitha Srinivas. Scalable highly expressive reasoner (SHER). *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(4):357 – 361, 2009. Semantic Web challenge 2008.

[DLN+98] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.

[DM00] Francesco M. Donini and Fabio Massacci. Exptime tableaux for ALC. *Artif. Intell.*, 124:87–138, November 2000.

[DS05] Andreas Doms and Michael Schroeder. GoPubMed: exploring PubMed with the Gene Ontology. *Nucleic Acids Research*, 33(Web-Server-Issue):783–786, 2005.

[FKM+06] Achille Fokoue, Aaron Kershenbaum, Li Ma, Edith Schonberg, and Kavitha Srinivas. The Summary Abox: Cutting Ontologies Down to Size. In Isabel Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Mike Uschold, and Lora Aroyo, editors, *The Semantic Web - ISWC 2006*, volume 4273 of *Lecture Notes in Computer Science*, pages 343–356. Springer Berlin / Heidelberg, 2006.

[Fra11] Franz Inc. Allegrograph. http://www.franz.com/agraph/, 2011.

[FS06] Ulrich Furbach and Natarajan Shankar, editors. *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer Science*. Springer, 2006.

[GF92] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford University, Stanford, CA, USA, 1992.

[GFW08] Marcos André Gonçalves, Edward A. Fox, and Layne T. Watson. Towards a digital library theory: a formal digital library ontology. *Int. J. on Digital Libraries*, 8(2):91–114, 2008.

[GHKS09] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Extracting Modules from Ontologies: A Logic-Based Approach. In Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors, *Modular Ontologies*, volume 5445 of *Lecture Notes in Computer Science*, pages 159–186. Springer, 2009.

[GHLS07] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive Query Answering in the Description Logic SHIQ. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 2007.

[GHS08] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of Conjunctive Queries in SHOQ. In *Proceedings of the 11th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 252–262. AAAI Press/The MIT Press, 2008.

[GMN+09] O. Gries, R. Möller, A. Nafissi, K. Sokolski, and M. Rosenfeld. CASAM Domain Ontology. Technical report, Hamburg University of Technology, 2009.

[GMN+10] Oliver Gries, Ralf Möller, Anahita Nafissi, Maurice Rosenfeld, Kamil Sokolski, and Michael Wessel. A Probabilistic Abduction Engine for Media Interpretation Based on Ontologies. In Pascal Hitzler and Thomas Lukasiewicz,

editors, *RR*, volume 6333 of *Lecture Notes in Computer Science*, pages 182–194. Springer, 2010.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[GPSK06] Bernardo Cuenca Grau, Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Modularity and web ontologies. In *Proceedings of KR-2006*, pages 198–209. AAAI Press, 2006.

[GR10] Birte Glimm and Sebastian Rudolph. Status QIO: Conjunctive query entailment is decidable. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference (KR 2010)*, 2010.

[Gru09] Tom Gruber. Ontology. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1963–1965. Springer US, 2009.

[Gua98] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy.* IOS Press, Amsterdam, The Netherlands, 1st edition, 1998.

[HB09] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for working with OWL 2 ontologies. In Rinke Hoekstra and Peter F. Patel-Schneider, editors, *OWLED*, volume 529 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[HH08] Zhisheng Huang and Frank Harmelen. Using Semantic Distances for Reasoning with Inconsistent Ontologies. In *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pages 178–194, Berlin, Heidelberg, 2008. Springer-Verlag.

[HKP+09] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, World Wide Web Consortium, October 2009.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The Even More Irresistible SROIQ. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), Lake District of the United Kingdom*, pages 57–67. AAAI Press, 2006.

[HM04] Volker Haarslev and Ralf Moeller. Optimization Techniques for Retrieving Resources Described in OWL/RDF Documents: First Results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning, KR 2004*, pages 2–5, 2004.

[HM08] Volker Haarslev and Ralf Möller. On the Scalability of Description Logic Instance Retrieval. *J. Autom. Reason.*, 41(2):99–142, 2008.

[HMS04] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reducing SHIQ-Description Logic to Disjunctive Datalog Programs. In Didier Dubois, Christopher A. Welty, and Mary-Anne Williams, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada*, pages 152–162. AAAI Press, 2004.

[HMW04] V. Haarslev, R. Möller, and M. Wessel. Querying the Semantic Web with Racer + nRQL. In *Proceedings of the KI-2004 International Workshop on Applications of Description Logics (ADL'04), Ulm, Germany, September 24*, 2004.

[Hor98] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, California, June 1998.

[HS99] Ian Horrocks and Ulrike Sattler. A Description Logic with Transitive and Inverse Roles and Role Hierarchies. *J. Log. Comput.*, 9(3):385–410, 1999.

[HS01] Ian Horrocks and Ulrike Sattler. Ontology Reasoning in the SHOQ(D) Description Logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001*, pages 199–204. Morgan Kaufmann, 2001.

[HS07] Ian Horrocks and Ulrike Sattler. A Tableau Decision Procedure for SHOIQ. *J. Autom. Reason.*, 39(3):249–276, 2007.

[HST00a] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Very Expressive Description Logics. *Logic Journal of the IGPL*, 8(3):239–264, 2000.

[HST00b] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Reasoning with Individuals for the Description Logic SHIQ. In *CADE-17: Proceedings of the 17th International Conference on Automated Deduction*, pages 482–496, London, UK, 2000. Springer-Verlag.

[HT73] John E. Hopcroft and Robert Endre Tarjan. Efficient algorithms for graph manipulation [H] (Algorithm 447). *Commun. ACM*, 16(6):372–378, 1973.

[HVHT05] Zhisheng Huang, Frank Van Harmelen, and Annette Ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the 19th International Joint Conference on Artificial intelligence*, pages 454–459, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[HWPS06] Christian Halaschek-Wiener, Bijan Parsia, and Evren Sirin. Description Logic Reasoning with Syntactic Updates. In Robert Meersman and Zahir Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 722–737. Springer, 2006.

[IF02] ISO/IEC15938-5FCD. Multimedia Content Description Interface (MPEG-7). http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm, 2002.

[Kir06] Atanas Kiryakov. OWLIM: Balancing between scalable repository and lightweight reasoner. In *Proc. of WWW2006*, Edinburgh, Scotland, 2006.

[KKS09] Sebastian Ryszard Kruk, Ewelina Kruk, and Katarzyna Stankiewicz. Evaluation of Semantic and Social Technologies for Digital Libraries. In Sebastian Ryszard Kruk and Bill McDaniel, editors, *Semantic Digital Libraries*, pages 203–214. Springer, 2009.

[KL04] Eldar Karabaev and Carsten Lutz. Mona as a DL Reasoner. In Volker Haarslev and Ralf Möller, editors, *Description Logics*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[KLPW10] Boris Konev, Carsten Lutz, Denis Ponomaryov, and Frank Wolter. Decomposing Description Logic Ontologies. In Fangzhen Lin and Ulrike Sattler, editors, *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR2010)*. AAAI Press, 2010.

[KLW94] Ramakrishna Karedla, J. Spencer Love, and Bradley G. Wherry. Caching strategies to improve disk system performance. *Computer*, 27(3):38–46, 1994.

[KN03] Yevgeny Kazakov and Hans De Nivelle. Subsumption of concepts in FL0 with respect to descriptive semantics is PSPACE-complete. In *In Proc. DL 03. http://CEUR-WS.org/Vol-81*, 2003.

[Knu81] Donald E. Knuth. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[Luk08] Thomas Lukasiewicz. Expressive probabilistic description logics. *Artif. Intell.*, 172(6-7):852–883, 2008.

[Lut03] C. Lutz. Description Logics with Concrete Domains—A Survey. In Philippe Balbiani, Nobu-Yuki Suzuki, Frank Wolter, and Michael Zakharyaschev, editors, *Advances in Modal Logics Volume 4*. King's College Publications, 2003.

[LW10] Carsten Lutz and Frank Wolter. Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. *Journal of Symbolic Computation*, 45(2):194–228, 2010.

[Min74] Marvin Minsky. A Framework for Representing Knowledge. Technical report, MIT-AI Laboratory, Cambridge, MA, USA, 1974.

[Mot08] Boris Motik. KAON2 - Scalable Reasoning over Ontologies with Large Data Sets. *ERCIM News*, 2008(72), 2008.

[MP06] Lutz Maicher and Jack Park, editors. *Charting the Topic Maps Research and Applications Landscape, First International Workshop on Topic Maps Research and Applications, TMRA 2005, Leipzig*, volume 3873 of *Lecture Notes in Computer Science*. Springer, 2006.

[MW88] David Maier and David Scott Warren. *Computing with Logic: Logic Programming with Prolog*. Benjamin/Cummings, 1988.

[Neb94] Bernhard Nebel. Base Revision Operations and Schemes: Semantics, Representation, and Complexity. In *Proceedings of the 11th European Conference on Artificial Intelligence*, pages 341–345. John Wiley and Sons, August 1994.

[PS98] Peter F. Patel-Schneider. DLP System Description. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Description Logics*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[PSH07] Peter F. Patel-Schneider and Ian Horrocks. A comparison of two modelling paradigms in the Semantic Web. *J. Web Sem.*, 5(4):240–250, 2007.

[PSS93] Peter F. Patel-Schneider and B. Swartout. Description-Logic Knowledge Representation System Specification. Technical report, KRSS Group of the ARPA Knowledge Sharing Effort, November 1993.

[PTP10] Katerina Papantoniou, George Tsatsaronis, and Georgios Paliouras. KDTA: Automated Knowledge-Driven Text Annotation. In José L. Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *ECML/PKDD (3)*, volume 6323 of *Lecture Notes in Computer Science*, pages 611–614. Springer, 2010.

[PTZ09] Jeff Z. Pan, Edward Thomas, and Yuting Zhao. Completeness Guaranteed Approximations for OWL-DL Query Answering. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[Qui68] Ross Quillian. Semantic memory. In *Semantic Information Processing*, pages 216–270. MIT Press, 1968.

[RDE+07] Kurt Rohloff, Mike Dean, Ian Emmons, Dorene Ryder, and John Sumner. An evaluation of triple-store technologies for large data stores. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems - Volume Part II*, OTM'07, pages 1105–1114, Berlin, Heidelberg, 2007. Springer-Verlag.

[Rei77] Raymond Reiter. On Closed World Data Bases. In *Logic and Data Bases*, pages 55–76, 1977.

[RPZ10] Yuan Ren, Jeff Z. Pan, and Yuting Zhao. Soundness Preserving Approximation for TBox Reasoning. In Maria Fox and David Poole, editors, *AAAI*. AAAI Press, 2010.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15:91–110, August 2002.

[SBK+07] Stefan Schlobach, E. Blaauw, M. El Kebir, Annette ten Teije, Frank van Harmelen, S. Bortoli, M. C. Hobbelman, K. Millian, Y. Ren, S. Stam, P. Thomassen, R. C. van het Schip, and W. van Willigem. Anytime Classification by Ontology Approximation. In Ruzica Piskac, Frank van Harmelen, and Ning Zhong, editors, *New Forms of Reasoning for the Semantic Web*, volume 291 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial intelligence - Volume 1*, pages 466–471, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[SCH10] Giorgos Stoilos, Bernardo Cuenca Grau, and Ian Horrocks. How Incomplete is your Semantic Web Reasoner? In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 10)*, pages 1431–1436. AAAI Publications, 2010.

[Smo88] Gert Smolka. A Feature Logic with Subsorts. *LILOG-Report*, 33:–1–1, 1988.

[Spa07] Bastian Spanneberg. Prototypische Implementierung eines Triple-Store-basierten ALC-Reasoners. Technical report, Hamburg University of Technology, 2007.

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[SS89] Manfred Schmidt-Schaubß. Subsumption in KL-ONE is undecidable. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 421–431, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[SSS91] Manfred Schmidt-Schauss and Gert Smolka. Attributive concept descriptions with complements. *Artif. Intell.*, 48:1–26, February 1991.

[Str05] Umberto Straccia. Towards a Fuzzy Description Logic for the Semantic Web (Preliminary Report). In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2005.

[TGH10] Tuvshintur Tserendorj, Stephan Grimm, and Pascal Hitzler. Approximate Instance Retrieval on Ontologies. In *Database and Expert Systems Applications DEXA 2010, Bilbao, Spain*, pages 503–511, 2010.

[TL10] Duc Thanh Tran and Guenter Ladwig. Structure Index for RDF Data. In *Proceedings of the Workshop on Semantic Data Management (SemData) at the 36th International Conference on Very Large Databases (VLDB2010)*. VLDB Endowment, September 2010.

[Tob01] Stephan Tobies. Complexity results and practical algorithms for logics in knowledge representation. *The Computing Research Repository*, 2001.

[TRKH08] Tuvshintur Tserendorj, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Approximate OWL-reasoning with Screech. In Diego Calvanese and Georg Lausen, editors, *RR*, volume 5341 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2008.

[TV03] Christoph Tempich and Raphael Volz. Towards a benchmark for Semantic Web reasoners - an analysis of the DAML ontology library. In York Sure and Óscar Corcho, editors, *EON*, volume 87 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[Vou08] M.A. Vouk. Cloud computing; Issues, research and implementations. In *Information Technology Interfaces, 2008. ITI 2008. 30th International Conference on Information Technology*, pages 31 –40, 2008.

[WA02] Michael Widenius and Davis Axmark. *MySQL Reference Manual*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.