UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

The DLE-Hyper Tableau Calculus: A Decision Procedure for $\mathcal{SHIQ}$

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Informatik

vorgelegt von
Dennis Faßbender

Erstgutachter: Prof. Dr. Ulrich Furbach
(Institut für Informatik, AG Künstliche Intelligenz)

Zweitgutachter: Dipl.-Inform. Claudia Schon
(Institut für Informatik, AG Künstliche Intelligenz)

Koblenz, im März 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☒	☐

. . . . . .
(Ort, Datum) (Unterschrift)

Acknowledgements

I would like to thank Claudia Schon and Björn Pelzer for their support during my work on this thesis.

Contents

Chapter 1

Introduction

In the last decade or so, expressive Description Logics (DLs) such as $\mathcal{SHIQ}$ have attracted considerable attention, due largely to their potential as *ontology languages* [2] for the *Semantic Web* [9]. The goal of the Semantic Web is to enrich websites with machine-readable semantic information about their contents in order to facilitate the search for information, whether by automated algorithms or by human users. Ontologies are needed to provide these agents with common definitions of concepts in order to ensure that the semantic information with which a website is annotated is interpreted the same way by all agents. Description Logics are well-suited as ontology languages because they offer both clearly defined semantics and sufficient expressive power. Until recently, though, Description Logic reasoners were not able to reason over large real-world ontologies such as GALEN[1].

In 2007, a major step forward was made when Motik et al. [6] presented HermiT[2], a highly efficient reasoner capable of deciding the satisfiability of $\mathcal{SHIQ}$ knowledge bases and of classifying previously unclassifiable ontologies. HermiT's underlying calculus is based on 'hyper tableaux' [6], a combination of clausal normal form tableaux and hyper-resolution [17] developed by Peter Baumgartner, Ulrich Furbach and Ilkka Niemelä at University of Koblenz in 1996. In the meantime, the original Hyper Tableau calculus was extended with efficient equality handling and the resulting calculus was named E-Hyper Tableau calculus. It is currently implemented in E-KRHyper [16]. Due to HermiT's success as well as the fact that, like E-KRHyper, it works with hyper tableaux, Ulrich Furbach, Claudia Schon and Björn Pelzer decided to extend the E-Hyper Tableau calculus with a new inference rule in order to enable it to decide $\mathcal{SHIQ}$ as well.

In this thesis, I will present the result of this work, the DLE-Hyper Tableau calculus, and prove that it is indeed a decision procedure for $\mathcal{SHIQ}$. First, chapter 2 provides a general introduction to Description Logics as well as a formal specification of the syntax and semantics of $\mathcal{SHIQ}$ and $\mathcal{ALCHIQ}$. (The latter DL will be relevant in the next chapter.) Chapter 3 then describes the transformation of a $\mathcal{SHIQ}$ knowledge base into a set of *DLE-clauses* that are compatible with the DLE-Hyper Tableau calculus. Next, the original E-Hyper Tableau calculus is covered in chapter 4 before chapter 5 presents the changes that were made in order to obtain the DLE-Hyper Tableau calculus. Finally, in chapter 6, I will prove that the DLE-Hyper Tableau calculus is sound, terminating and complete, which means that it is a decision procedure. Chapter 7 contains brief descriptions of related work as well as ideas for possible future extensions to the calculus, while chapter 8 concludes this thesis.

[1] `http://www.opengalen.org/`
[2] `http://hermit-reasoner.com/`

Chapter 2

The Description Logics $\mathcal{SHIQ}$ and $\mathcal{ALCHIQ}$

Before introducing the formal syntax and semantics of $\mathcal{SHIQ}$ and $\mathcal{ALCHIQ}$ in detail, the following section will provide a brief introduction to Description Logics (DLs) in general. While a comprehensive treatment of DLs would go beyond the scope of this thesis, more in-depth information can be found in [1] or [3] (available online[1]).

2.1 Introduction to Description Logics

Description Logics are knowledge representation languages with well-understood semantics. Their basic building blocks are *atomic concepts* (unary predicates), *atomic roles* (binary predicates) and *individuals* (constants). DL concepts are interpreted as sets of individuals, whereas roles represent binary relations on the set of all individuals. Atomic concepts and atomic roles can be used to describe more complex concepts and roles with the help of *concept constructors*.

For example, if one wanted to define the concept *HappyMother* as "a woman who has at least two but no more than four children, all of whom are happy," one could use the following definition:

$$\mathit{HappyMother} \equiv \mathit{Woman} \sqcap (\geq 2\ \mathit{hasChild}.\top) \sqcap (\leq 4\ \mathit{hasChild}.\top) \sqcap \forall \mathit{hasChild}.\mathit{HappyChild}$$

Woman and *Happy* are concepts, and *hasChild* denotes a role. The *conjunction* operator $\sqcap$ corresponds to the set intersection operator $\cap$. $\top$ stands for the universal concept which every individual belongs to. The complex concept $\geq 2\ \mathit{hasChild}.\top$ ($\geq$-*number restriction*) represents the set of those individuals that are in a *hasChild* relationship with at least two distinct individuals. $\leq 4\ \mathit{hasChild}.\top$ ($\leq$-*number restriction*) is interpreted analogously. The set corresponding to $\forall \mathit{hasChild}.\mathit{HappyChild}$ (*value restriction*) contains all those individuals that are in a *hasChild* relationship with happy children only (or no children at all).

In expressive Description Logics such as $\mathcal{SHIQ}$ it is also possible to define relationships between roles. For example, it would make sense to state that any two individuals that are in a *hasChild* relationship are also in a *relatedTo* relationship. This can be expressed as follows:

$$\mathit{hasChild} \sqsubseteq \mathit{relatedTo}$$

$\sqsubseteq$ is essentially equivalent to the subset operator $\subseteq$. (In fact, the $\equiv$ operator used to define *HappyMother* is unnecessary as every axiom of the form $C \equiv D$ can be replaced by the two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.) The above axiom states that the binary relation represented by the role *hasChild* must be

[1] `http://www.cs.ox.ac.uk/ian.horrocks/Publications/download/2007/BaHS07a.pdf`

contained in the relation assigned to *relatedTo*. Now suppose we wanted *relatedTo* to be interpreted as a symmetric relation. This can be done using the *inverse role* operator $-$:

$$relatedTo \sqsubseteq relatedTo^{-}$$

We have thus stated that the relation represented by *relatedTo* is a subset of its own inverse relation.

Finally, a DL knowledge base (KB) can also contain *assertions* about particular individuals. The following assertions express that Mary is a woman who is the mother of Tim, who is a happy child. She is also the mother of Timothy, but as indicated by the last assertion, Tim and Timothy are to be interpreted as the same person:

$$\begin{aligned}
&Woman(mary),\\
&HappyChild(tim),\\
&hasChild(mary, tim),\\
&hasChild(mary, timothy),\\
&tim \simeq timothy
\end{aligned}$$

While the above examples have hopefully conveyed a feel for the syntax and semantics of Description Logics, it is now time to formally define the expressive DLs $\mathcal{SHIQ}$ and $\mathcal{ALCHIQ}$.

2.2 Syntax and Semantics of $\mathcal{SHIQ}$ and $\mathcal{ALCHIQ}$

This section is essentially a summary of the relevant information found in [13] and [2], with the latter article containing more details on topics such as inference problems in DL knowledge bases. We will first have a look a the syntax of $\mathcal{SHIQ}$ before covering its semantics.

Given a set of *atomic roles* N_R, the set of *roles* is defined as $N_R \cup \{R^- \mid R \in N_R\}$, where R^- denotes the *inverse role* corresponding to the atomic role R. In order to avoid having to use unnecessarily complex expressions such as R^{----} (which is simply equivalent to R), let Inv be a function on the set of roles that computes the inverse of a role, with $Inv(R) = R^-$ and $Inv(R^-) = R$.

A *role inclusion axiom* is an expression of the form $R \sqsubseteq S$, where R and S are atomic or inverse roles, respectively. A *transitivity axiom* is of the form $Trans(S)$ for S an atomic or inverse role. An RBox $\mathcal{R}$ is a finite set of role inclusion axioms and transitivity axioms.

$\sqsubseteq^*$ denotes the reflexive transitive closure of $\sqsubseteq$ over $\{R \sqsubseteq S, Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is *transitive* in $\mathcal{R}$ if there exists a role S such that $S \sqsubseteq^* R$, $R \sqsubseteq S^*$, and either $Trans(S) \in \mathcal{R}$ or $Trans(Inv(S)) \in \mathcal{R}$. If no transitive role S with $S \sqsubseteq^* R$ exists, R is called *simple*.

Let N_C be the set of *atomic concepts*. The set of *concepts* is then defined as the smallest set containing $\top$, $\bot$, A, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\geq n\ S.C$, and $\leq n\ S.C$ for $A \in N_C$, C and D concepts, R a role, S a simple role, and n a non-negative integer.

A *general concept inclusion* (abbreviated as *GCI*) is of the form $C \sqsubseteq D$, and a TBox $\mathcal{T}$ is a finite set of GCIs.

Given a set N_I of *individuals*, an ABox $\mathcal{A}$ is a finite set of *concept assertions* $C(a)$, *role assertions* $R(a, b)$, and *equality* as well as *inequality assertions* $a \simeq b$ and $a \not\simeq b$, where C is a concept, R is a role, and $a, b \in N_I$ are individuals.

A $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$ is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$. When dealing with several different knowledge bases, the notations $\mathcal{K}_\mathcal{R}$, $\mathcal{K}_\mathcal{T}$ and $\mathcal{K}_\mathcal{A}$ will be used to denote the RBox, TBox and ABox, respectively, of the KB $\mathcal{K}$. The tuple $I = (\cdot^I, \Delta^I)$ is an *interpretation* for $\mathcal{K}$ iff Δ^I is a nonempty set and $\cdot^I$ assigns an

element $a^I \in \Delta^I$ to each individual a, a set $A^I \subseteq \Delta^I$ to each atomic concept A, and a relation $R^I \subseteq \Delta^I \times \Delta^I$ to each atomic role R. Based on these building blocks, $\cdot^I$ assigns values to more complex concepts and roles as described in Table 2.1. I is a *model* of $\mathcal{K}$ ($I \models \mathcal{K}$) if it satisfies all axioms and assertions in $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$ as shown in Table 2.1.

A concept C is called *satisfiable w.r.t.* $\mathcal{R}$ and $\mathcal{T}$ iff there exists a model I of $\mathcal{R}$ and $\mathcal{T}$ with $C^I \neq \emptyset$. The concept D *subsumes* C w.r.t. $\mathcal{R}$ and $\mathcal{T}$ ($C \sqsubseteq_{(\mathcal{R},\mathcal{T})} D$) iff $C^I \subseteq D^I$ holds in every model I of $\mathcal{R}$ and $\mathcal{T}$. C and D are called *equivalent* w.r.t. $\mathcal{R}$ and $\mathcal{T}$ ($C \equiv_{(\mathcal{R},\mathcal{T})} D$) iff they subsume each other. As a result, checking whether two concepts C and D are equivalent can be reduced to checking whether the concepts subsume each other. Checking subsumption can then be reduced to checking satisfiability: $C \sqsubseteq_{(\mathcal{R},\mathcal{T})} D$ holds iff $C \sqcap \neg D$ is unsatisfiable w.r.t. $\mathcal{R}$ and $\mathcal{T}$.

Concepts and Roles

$\top^I = \Delta^I$	(universal concept)
$\bot^I = \emptyset$	(empty concept)
$(\neg C)^I = \Delta^I \backslash C^I$	(negation)
$(C \sqcup D)^I = C^I \cup D^I$	(disjunction)
$(C \sqcap D)^I = C^I \cap D^I$	(conjunction)
$(R^-)^I = \{(y, x) \mid (x, y) \in R^I\}$	(inverse role)
$(\forall R.C)^I = \{x \mid \forall y : (x, y) \in R^I \Rightarrow y \in C^I\}$	(value restriction)
$(\exists R.C)^I = \{x \mid \exists y : (x, y) \in R^I \wedge y \in C^I\}$	(exists restriction)
$(\geq n\ R.C)^I = \{x \mid \sharp\{y \mid (x, y) \in R^I \wedge y \in C^I\} \geq n\}$	($\geq$-number restriction)
$(\leq n\ R.C)^I = \{x \mid \sharp\{y \mid (x, y) \in R^I \wedge y \in C^I\} \leq n\}$	($\leq$-number restriction)

TBox axioms

$C \sqsubseteq D \Rightarrow C^I \subseteq D^I$	(concept inclusion axiom)

RBox axioms

$R \sqsubseteq S \Rightarrow R^I \subseteq S^I$	(role inclusion axiom)
$Trans(R) \Rightarrow (R^I)^+ \subseteq R^I$	(transitivity axiom)

ABox assertions

$C(a) \Rightarrow a^I \in C^I$	(concept assertion)
$R(a, b) \Rightarrow (a^I, b^I) \in R^I$	(role assertion)
$a \simeq b \Rightarrow a^I = b^I$	(equality assertion)
$a \not\simeq b \Rightarrow a^I \neq b^I$	(inequality assertion)

Table 2.1: Model-theoretic semantics of $\mathcal{SHIQ}$. $\sharp N$ denotes the number of elements in the set N; R^+ is the transitive closure of R.

The *negation normal form* $nnf(C)$ of a concept C is the concept that is equivalent to C but contains negations only in front of atomic concepts. For example, given two atomic concepts A and B and

an atomic role R, the following example demonstrates how the negation normal form of the complex concept C is computed:

$$\begin{aligned} C &= \neg((A \sqcap B) \sqcup \exists R.A) \\ C' &= \neg(A \sqcap B) \sqcap \neg(\exists R.A) \\ C'' &= (\neg A \sqcup \neg B) \sqcap \forall R.\neg A = nnf(C) \end{aligned}$$

$\dot{\neg}C$ is used as an abbreviation for $nnf(\neg C)$.

The Description Logic $\mathcal{ALCHIQ}$ is equivalent to $\mathcal{SHIQ}$ except that it does not have transitive roles.

2.3 Example

In order to get a feel for the semantics of $\mathcal{SHIQ}$, consider the knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, whose components contain the axioms and assertions introduced in the previous section:

$$\begin{aligned} \mathcal{R} = \{&hasChild \sqsubseteq relatedTo, \\ &relatedTo \sqsubseteq relatedTo^{-}\} \end{aligned}$$

$$\mathcal{T} = \{HappyMother \equiv Woman \ \sqcap \ (\geq 2 \ hasChild.\top) \ \sqcap \ (\leq 4 \ hasChild.\top) \ \sqcap \forall hasChild.HappyChild\}$$

$$\begin{aligned} \mathcal{A} = \{&Woman(mary), \\ &HappyChild(tim), \\ &hasChild(mary, tim), \\ &hasChild(mary, timothy), \\ &tim \simeq timothy\} \end{aligned}$$

We will now construct a model I of $\mathcal{K}$. Let $\Delta^I = \{mary, tim\}$, with $mary^I = mary$, $tim^I = tim$ and $timothy^I = tim$, as the equality assertion in the ABox requires *tim* and *timothy* to have identical interpretations in any model of $\mathcal{K}$. For the remaining ABox assertions to be true in I, we must have $mary^I \in Woman^I$, $tim^I \in HappyChild$ and $(mary^I, tim^I) \in hasChild^I$, so let $Woman = \{mary\}$, $HappyChild^I = \{tim\}$ and $hasChild^I = \{(mary, tim)\}$.

Now consider the RBox axioms. Since $hasChild \sqsubseteq relatedTo$ implies $hasChild^I \subseteq relatedTo^I$, we set $relatedTo^I = \{(mary, tim)\}$ for now. However, this would entail $(relatedTo^{-})^I = \{(tim, mary)\}$, which would mean that the second RBox axiom is false in I because $relatedTo^I$ is not a subset of $(relatedTo^{-})^I$. Hence, we need to add $(tim, mary)$ to $relatedTo^I$, which gives us the symmetric relations $relatedTo^I = (relatedTo^{-})^I = \{(mary, tim), (tim, mary)\}$.

As the right-hand side of the TBox axiom is empty in I (while Mary is a woman whose children are all happy, she does not have enough *distinct* children to be considered a happy mother in our world), we can simply set $HappyMother^I = \emptyset$. Since every axiom and assertion in $\mathcal{K}$ is now true in I, I is a model of $\mathcal{K}$.

As an example of a less intuitive interpretation that is also a model of $\mathcal{K}$, consider the following definition of I:

$$\begin{aligned}
\Delta^I &= \{tim, mary\} \\
tim^I &= tim \\
timothy^I &= tim \\
mary^I &= mary \\
HappyMother^I &= \{mary\} \\
Woman^I &= \{mary\} \\
HappyChild^I &= \{mary, tim\} \\
hasChild^I &= \{(mary, tim), (mary, mary)\} \\
relatedTo^I &= \{(mary, tim), (tim, mary), (mary, mary)\}
\end{aligned}$$

In this interpretation, Mary is a happy mother because she has two distinct, happy children. The fact that she is her own child may seem strange, but it is possible in the world described by this knowledge base.

Chapter 3

Transforming $\mathcal{SHIQ}$ into DLE-Clauses

In order to decide the satisfiability of a $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$ using the DLE-Hyper Tableau calculus, it must first be transformed into a set of *DLE-clauses*, a variant of the *DL-clauses* described in [13]. The transformation is a five-step process: first, $\mathcal{K}$ is transformed into an equisatisfiable $\mathcal{ALCHIQ}$ knowledge base $\Omega(\mathcal{K})$ (section 3.1). $\Omega(\mathcal{K})$ is then brought into its equisatisfiable *normalized* form $\Delta(\Omega(\mathcal{K}))$ (section 3.2). Next, the RBox and TBox of $\Delta(\Omega(\mathcal{K}))$ are turned into a set of DL-clauses $\Xi(\Delta(\Omega(\mathcal{K})))$ such that $\Delta(\Omega(\mathcal{K}))$ and $\Xi(\Delta(\Omega(\mathcal{K})))\cup\Delta(\Omega(\mathcal{K}))_{\mathcal{A}}$ are equisatisfiable (section 3.5). The transformations up to this point were all described by Motik et al. in [13]. Then, $\Xi(\Delta(\Omega(\mathcal{K})))\cup\Delta(\Omega(\mathcal{K}))_{\mathcal{A}}$ is converted into an equisatisfiable set of *DL-clauses* $DL(\Delta(\Omega(\mathcal{K})))$, which is finally transformed into an equisatisfiable set of *DLE-clauses* $DLE(\Delta(\Omega(\mathcal{K})))$ (section 3.5) that can be processed by the DLE-Hyper Tableau calculus (to be introduced in chapter 5).

3.1 From $\mathcal{SHIQ}$ to $\mathcal{ALCHIQ}$

Motik et al. [13] transform a $\mathcal{SHIQ}$ KB $\mathcal{K}$ into an equisatisfiable $\mathcal{ALCHIQ}$ KB $\Omega(\mathcal{K})$ by removing all transitivity axioms using a procedure presented in [12]. The following two definitions describe the procedure and are almost verbatim copies of Definitions 5.1.1 and 5.1.2 in [12]. Note that C and D denote concepts, while R and S are roles.

Definition 3.1.1 (**Concept Closure [12]).** For a $\mathcal{SHIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$, let $clos(\mathcal{K})$ denote the *concept closure* of $\mathcal{K}$, defined as the smallest set of concepts satisfying the following conditions:

- If $C \sqsubseteq D \in \mathcal{T}$, then $nnf(\neg C \sqcup D) \in clos(\mathcal{K})$;
- If $C \equiv D \in \mathcal{T}$, then $nnf(\neg C \sqcup D) \in clos(\mathcal{K})$ and $nnf(\neg D \sqcup C) \in clos(\mathcal{K})$;
- If $C(a) \in \mathcal{A}$, then $nnf(C) \in clos(\mathcal{K})$;
- If $C \in clos(\mathcal{K})$ and D is a subconcept of C, then $D \in clos(\mathcal{K})$;
- If $\leq n\ R.C \in clos(\mathcal{K})$, then $nnf(\neg C) \in clos(\mathcal{K})$;
- If $\forall R.C \in clos(\mathcal{K})$, $S \sqsubseteq^* R$, and $Trans(S) \in \mathcal{R}$, then $\forall S.C \in clos(\mathcal{K})$.

Definition 3.1.2 (**[12]).** For a $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$, $\Omega(\mathcal{K})$ is an $\mathcal{ALCHIQ}$ knowledge base constructed as follows:

- $\Omega(\mathcal{K})_{\mathcal{R}}$ is obtained from $\mathcal{K}_{\mathcal{R}}$ by removing all axioms $Trans(R)$;

- $\Omega(\mathcal{K})_{\mathcal{T}}$ is obtained by adding to $\mathcal{K}_{\mathcal{T}}$ the axiom $\forall R.C \sqsubseteq \forall S.(\forall S.C)$, for each concept $\forall R.C \in clos(\mathcal{K})$ and role S such that $S \sqsubseteq^* R$ and $Trans(S) \in \mathcal{K}_{\mathcal{R}}$;
- $\Omega(\mathcal{K})_{\mathcal{A}} = \mathcal{K}_{\mathcal{A}}$

Lemma 3.1.1. $\mathcal{K}$ is satisfiable if and only if $\Omega(\mathcal{K})$ is satisfiable.

Proof. This Lemma is identical to Theorem 5.2.3 in [12], which contains the proof. □

3.2 Normalizing an $\mathcal{ALCHIQ}$ Knowledge Base

Before an $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$ can be converted into a set of DLE-clauses, it must first be transformed into a *normalized* $\mathcal{ALCHIQ}$ knowledge base $\Delta(\mathcal{K})$. The following section lays out the theory behind the normalization process, while section 3.2.2 presents a practical example.

3.2.1 Theory

The following definition by Motik et al. [13] formally specifies the conditions under which a knowledge base is considered normalized. The actual normalization procedure is described in Table 3.1.

Definition 3.2.1 (**Normalization [13]**). For A an atomic concept, the concepts A, $\neg A$, $\top$, and $\bot$ are called literal concepts. A GCI is *normalized* if it is of the form $\top \sqsubseteq \bigsqcup_{i=1}^{n} C_i$, where each C_i is of the form B, $\forall R.B$, $\geq n\ R.B$, or $\leq n\ R.B$, and B is a literal concept. A TBox $\mathcal{T}$ is *normalized* if all GCIs in it are normalized. An ABox $\mathcal{A}$ is *normalized* if (i) each concept assertion in $\mathcal{A}$ is of the form $B(s)$ or $\geq n\ R.B$ for B a literal concept, (ii) each role assertion in $\mathcal{A}$ contains only atomic roles, and (iii) $\mathcal{A}$ contains at least one assertion. A knowledge base $\mathcal{K}$ is *normalized* iff $\mathcal{T}$ and $\mathcal{A}$ are normalized.

Note that the RBox is not affected by the transformation.

Lemma 3.2.1. An $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$ is satisfiable if and only if $\Delta(\mathcal{K})$ is satisfiable; $\Delta(\mathcal{K})$ can be computed in polynomial time; and $\Delta(\mathcal{K})$ is normalized.

Proof. This Lemma is identical to Lemma 1 in [13], which contains the proof. □

3.2.2 Example

In this section, we will transform an $\mathcal{ALCHIQ}$ KB into a normalized $\mathcal{ALCHIQ}$ KB. Since the $\mathcal{SHIQ}$ KB $\mathcal{K}$ that was introduced in section 2.3 does not contain transitive roles, it is also an $\mathcal{ALCHIQ}$ KB, so we can use is as input to the normalization procedure. We then compute $\mathcal{K}$'s normalized form $\Delta(\mathcal{K})$. Recall that the RBox $\mathcal{R}$ was defined as follows:

$$\mathcal{R} = \{hasChild \sqsubseteq relatedTo,\\ relatedTo \sqsubseteq relatedTo^{-}\}$$

Since the axioms in $\mathcal{R}$ are not affected by the transformation, we have $\Delta(\mathcal{K})_{\mathcal{R}} = \mathcal{R}$. Next, consider the TBox $\mathcal{T}$:

$$\mathcal{T} = \{HappyMother \equiv Woman\ \sqcap\ (\geq 2\ hasChild.\top)\ \sqcap\ (\leq 4\ hasChild.\top)\ \sqcap \forall hasChild.HappyChild\}$$

First of all, we expand the $\equiv$ operator, which is simply an abbreviation for $\sqsubseteq$ and $\sqsupseteq$:

$$HappyMother \sqsubseteq Woman\ \sqcap\ (\geq 2\ hasChild.\top)\ \sqcap\ (\leq 4\ hasChild.\top)\ \sqcap \forall hasChild.HappyChild \quad (1)$$
$$Woman\ \sqcap\ (\geq 2\ hasChild.\top)\ \sqcap\ (\leq 4\ hasChild.\top)\ \sqcap \forall hasChild.HappyChild \sqsubseteq HappyMother \quad (2)$$

$$
\begin{aligned}
\Delta(\mathcal{K}) &= \{\top(\iota)\} \cup \textstyle\bigcup_{\alpha \in \mathcal{R} \cup \mathcal{A}} \Delta(\alpha) \cup \bigcup_{C_1 \sqsubseteq C_2 \in \mathcal{T}} \Delta(\top \sqsubseteq nnf(\neg C_1 \sqcup C_2)) \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup C') &= \Delta(\top \sqsubseteq \mathbf{C} \sqcup \alpha_{C'}) \cup \textstyle\bigcup_{i=1}^{n} \Delta(\alpha_{C'} \sqsubseteq C_i) \text{ for } C' = \bigsqcap_{i=1}^{n} C_i \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup \forall R.D) &= \Delta(\top \sqsubseteq \mathbf{C} \sqcup \forall R.\alpha_D) \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup \geq n\ R.D) &= \Delta(\top \sqsubseteq \mathbf{C} \sqcup \geq n\ R.\alpha_D) \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(\top \sqsubseteq \mathbf{C} \sqcup \leq n\ R.D) &= \Delta(\top \sqsubseteq \mathbf{C} \sqcup \leq n\ R.\dot{\neg}\alpha_{D'}) \cup \Delta(\alpha_{D'} \sqsubseteq D') \text{ for } D' = \dot{\neg} D \\
\Delta(D(a)) &= \{\alpha_D(a)\} \cup \Delta(\alpha_D \sqsubseteq D) \\
\Delta(R^{-}(a,b)) &= \{R(b,a)\} \\
\Delta(\beta) &= \{\beta\} \text{ for any other axiom } \beta
\end{aligned}
$$

$$
\alpha_C = \begin{cases} Q_C & \text{if } pos(C) = true \\ \neg Q_C & \text{if } pos(C) = false \end{cases} \quad \Big| \quad \begin{array}{l} \text{where } Q_C \text{ denotes a fresh atomic} \\ \text{concept unique for } C \end{array}
$$

$$
\begin{aligned}
pos(\top) &= false & pos(C_1 \sqcap C_2) &= pos(C_1) \vee pos(C_2) \\
pos(\bot) &= false & pos(C_1 \sqcup C_2) &= pos(C_1) \vee pos(C_2) \\
pos(A) &= true & pos(\forall R.C_1) &= pos(C_1) \\
pos(\neg A) &= false & pos(\geq n\ R.C_1) &= true
\end{aligned}
$$

$$
pos(\leq n\ R.C_1) = \begin{cases} pos(\dot{\neg} C_1) & \text{if } n = 0 \\ true & \text{otherwise} \end{cases}
$$

Table 3.1: Computing the normalized form $\Delta(\mathcal{K})$ of an $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$: A is an atomic concept, C_i are arbitrary concepts, $\mathbf{C}$ is a possibly empty disjunction of arbitrary concepts, D is not a literal concept, and ι is a fresh individual.
Source: [13] (Table 2)

By the first line in Table 3.1, TBox axioms of the form $C_1 \sqsubseteq C_2$, for C_1 and C_2 arbitrary concepts, are first rewritten as $\top \sqsubseteq nnf(\neg C_1 \sqcup C_2)$. In this example, $\neg C_1 \sqcup C_2$ corresponds to the following expressions for axioms (1) and (2), respectively:

$$\neg HappyMother \sqcup (Woman \sqcap (\geq 2\ hasChild.\top) \sqcap (\leq 4\ hasChild.\top) \sqcap \forall hasChild.HappyChild)$$
$$\neg(Woman \sqcap (\geq 2\ hasChild.\top) \sqcap (\leq 4\ hasChild.\top) \sqcap \forall hasChild.HappyChild) \sqcup HappyMother$$

Next, the negation normal forms are computed (the first expression is in negation formal form already) and '$\top \sqsubseteq$' is added:

$$\top \sqsubseteq \neg HappyMother \sqcup (Woman \sqcap (\geq 2\ hasChild.\top) \sqcap (\leq 4\ hasChild.\top) \sqcap \forall hasChild.HappyChild)$$
$$\top \sqsubseteq \neg Woman \sqcup (\leq 1\ hasChild.\top) \sqcup (\geq 5\ hasChild.\top) \sqcup (\geq 1\ hasChild.\neg HappyChild) \sqcup HappyMother$$

Note that the $\forall$-operator in front of *hasChild.HappyChild* was turned into $\exists$, which is written here as '≥ 1' since the two constructors are equivalent. The second axiom is already normalized, but in the

case of the first one, a fresh concept C needs to be introduced to get rid of the conjunction. This yields the following set of axioms:

$$\begin{aligned}
&\top \sqsubseteq \neg \mathit{HappyMother} \sqcup C \\
&C \sqsubseteq \mathit{Woman} \\
&C \sqsubseteq (\geq 2\ \mathit{hasChild}.\top) \\
&C \sqsubseteq (\leq 4\ \mathit{hasChild}.\top) \\
&C \sqsubseteq \forall \mathit{hasChild}.\mathit{HappyChild} \\
&\top \sqsubseteq \neg \mathit{Woman} \ \sqcup\ (\leq 1\ \mathit{hasChild}.\top) \ \sqcup\ (\geq 5\ \mathit{hasChild}.\top) \sqcup \\
&\qquad (\geq 1\ \mathit{hasChild}.\neg \mathit{HappyChild}) \sqcup \mathit{HappyMother}
\end{aligned}$$

After normalizing the four newly introduced axioms, we finally obtain

$$\begin{aligned}
\Delta(\mathcal{K})_{\mathcal{T}} = \{&\top \sqsubseteq \neg \mathit{HappyMother} \sqcup C, \\
&\top \sqsubseteq \neg C \sqcup \mathit{Woman}, \\
&\top \sqsubseteq \neg C \sqcup (\geq 2\ \mathit{hasChild}.\top), \\
&\top \sqsubseteq \neg C \sqcup (\leq 4\ \mathit{hasChild}.\top), \\
&\top \sqsubseteq \neg C \sqcup \forall \mathit{hasChild}.\mathit{HappyChild}, \\
&\top \sqsubseteq \neg \mathit{Woman} \ \sqcup\ (\leq 1\ \mathit{hasChild}.\top) \ \sqcup\ (\geq 5\ \mathit{hasChild}.\top) \sqcup \\
&\qquad (\geq 1\ \mathit{hasChild}.\neg \mathit{HappyChild}) \sqcup \mathit{HappyMother}\}
\end{aligned}$$

As for the ABox, we only need to add the assertion $\top(\iota)$. Apart from that, nothing needs to be done since $\mathcal{A}$ only contains atomic concepts and roles. Hence, the normalized ABox $\Delta(\mathcal{K})_{\mathcal{A}}$ looks as follows:

$$\begin{aligned}
\Delta(\mathcal{K})_{\mathcal{A}} = \{&\top(\iota), \\
&\mathit{Woman}(\mathit{mary}), \\
&\mathit{HappyChild}(\mathit{tim}), \\
&\mathit{hasChild}(\mathit{mary}, \mathit{tim}), \\
&\mathit{hasChild}(\mathit{mary}, \mathit{timothy}), \\
&\mathit{tim} \simeq \mathit{timothy}\}
\end{aligned}$$

3.3 Syntax and Semantics of DL-Clauses and DLE-Clauses

DL-clauses were defined by Motik et al. in [13]. The following definition is basically a verbatim copy of Definition 3 in [13] except for some minor changes.

Let N_V be a set of variables disjoint from the set of $\mathcal{SHIQ}$ individuals N_I. A *DL-atom* is an expression of the form $C(s)$, $R(s,t)$, or $s \simeq t$ for s and t variables or individuals, C a concept and R a role. A *DL-clause* is an expression of the form

$$V_1 \vee \cdots \vee V_n \leftarrow U_1 \wedge \cdots \wedge U_m$$

where U_i and V_j are atoms, $m \geq 0$ and $n \geq 0$. The conjunction $U_1 \wedge \cdots \wedge U_m$ is called the *body* and the disjunction $V_1 \vee \cdots \vee V_n$ is called the *head* of the clause. Correspondingly, the atoms V_j are also called *head atoms*, while U_i are called *body atoms*.

Given an interpretation $I = (\Delta^I, \cdot^I)$ and a variable mapping $\mu : N_V \to \Delta^I$, let $a^{I,\mu} = a^I$ for an individual a and $x^{I,\mu} = \mu(x)$ for a variable x. For s and t variables or individuals, C a concept, R a role, U_i

and V_j atoms, and S a set of DL-clauses, satisfaction in I and μ is defined in [13] as follows:

$I, \mu \models C(s)$	if $s^{I,\mu} \in C^I$
$I, \mu \models R(s,t)$	if $(s^{I,\mu}, t^{I,\mu}) \in R^I$
$I, \mu \models s \simeq t$	if $s^{I,\mu} = t^{I,\mu}$
$I, \mu \models \bigvee_{j=1}^{n} V_j \leftarrow \bigwedge_{i=1}^{m} U_i$	if $I, \mu \models U_i$ for all $1 \leq i \leq m$ implies $I, \mu \models V_j$ for some $1 \leq j \leq n$
$I \models \bigvee_{j=1}^{n} V_j \leftarrow \bigwedge_{i=1}^{m} U_i$	if $I, \mu \models \bigvee_{j=1}^{n} V_n \leftarrow \bigwedge_{i=1}^{m} U_i$ for all variable mappings μ
$I \models S$	if $I \models C$ for all $C \in S$

DLE-clauses are defined almost identically, with one minor difference: DL-atoms of the form $C(s)$ and $R(s,t)$ are represented by the *DLE-atoms* $C(s) \simeq \mathbf{t}$ and $R(s,t) \simeq \mathbf{t}$, respectively. $\mathbf{t}$ represents a special constant that is not contained in the set of variables or individuals. This means that all DLE-atoms are equations. The semantics of DL-clauses and DLE-clauses are essentially identical, with satisfaction of $C(s) \simeq \mathbf{t}$ and $R(s,t) \simeq \mathbf{t}$ in I and μ defined as follows:

$$I, \mu \models C(s) \simeq \mathbf{t} \quad \text{if } s^{I,\mu} \in C^I$$
$$I, \mu \models R(s,t) \simeq \mathbf{t} \quad \text{if } (s^{I,\mu}, t^{I,\mu}) \in R^I$$

Apart from that, the semantics are identical to DL-clause semantics. The above definition only differs from the corresponding definition for DL-clauses due to the slight change in syntax. It is easy to see that the DLE-atoms $C(s) \simeq \mathbf{t}$ and $R(s,t) \simeq \mathbf{t}$ and the DL-atoms $C(s)$ and $R(s,t)$ are actually equivalent. In fact, the '$\simeq \mathbf{t}$' will usually be omitted and the $C(s) \simeq \mathbf{t}$ and $R(s,t) \simeq \mathbf{t}$ atoms in DLE-clauses will simply be written as their DL-counterparts $C(s)$ and $R(s,t)$ since this makes the clauses more readable. This simplified representation is also used in the following definition.

Definition 3.3.1 (Clause Names). Let a and b be individuals, C an atomic concept or a $\geq$-number restriction, A an atomic concept, and R an atomic role. Then, a *role clause* is a unit clause of the form $(R(a,b) \leftarrow)$. Clauses of the form $(C(a) \leftarrow)$ are called *concept clauses*. If C is a $\geq$-number restriction, the clause is also called an *at-least clause*. The negative unit clause $(\leftarrow A(a))$ is called a *negative concept clause*. An *equality clause* is a unit clause of the form $(a \simeq b \leftarrow)$, whereas an *inequality clause* is a unit clause of the form $(\leftarrow a \simeq b)$.

3.4 From Normalized $\mathcal{ALCHIQ}$ to DL-Clauses

Now that the notion of DL-clauses has been defined, we will cover Motik et al.'s [13] method of translating the RBox and TBox of a normalized $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$ into a set of DL-clauses.

3.4.1 Theory

The procedure is described in Table 3.2, with $\Xi(\mathcal{K})$ denoting the resulting set of DL-clauses.

Lemma 3.4.1. Let $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ be a normalized $\mathcal{ALCHIQ}$ knowledge base. Then, $I \models \mathcal{K}$ if and only if $I \models \Xi(\mathcal{K})$ and $I \models \mathcal{A}$.

Proof. This Lemma is identical to Lemma 2 in [13], which contains the proof. □

The transformation described so far is exactly the one used by Motik et al. [13]. However, while their calculus works with ABoxes, the DLE-Hyper Tableau calculus can only process sets of clauses, which means we need to transform the ABox $\mathcal{A}$ into a set of DL-clauses as well.

$$\Xi(\mathcal{K}) = \{[\bigwedge_{i=1}^{n} head(C_i)] \leftarrow [\bigvee_{i=1}^{n} body(C_i)] \mid \text{ for each } \top \sqsubseteq \bigsqcup_{i=1}^{n} C_i \text{ in } \mathcal{T}\} \cup$$
$$\{ar(S, x, y) \leftarrow ar(R, x, y) \mid \text{ for each } R \sqsubseteq S \text{ in } \mathcal{R}\}$$

$$ar(R, s, t) = \begin{cases} R(s, t) & \text{if } R \text{ is an atomic role} \\ S(t, s) & \text{if } R \text{ is an inverse role and } R = S^- \end{cases}$$

head(*C*) and *body*(*C*) are defined as follows for a concept *C*:

C	head(C)	body(C)
A	$A(x)$	
$\neg A$		$A(x)$
$\geq n\ R.A$	$\geq n\ R.A(x)$	
$\geq n\ R.\neg A$	$\geq n\ R.\neg A(x)$	
$\forall R.A$	$A(y_C)$	$ar(R, x, y_C)$
$\forall R.\neg A$		$ar(R, x, y_C) \wedge A(y_C)$
$\leq n\ R.A$	$\bigvee_{i=1}^{n+1} \bigvee_{j=i+1}^{n+1} y_C^i \approx y_C^j$	$\bigwedge_{i=1}^{n+1} [ar(R, x, y_C^i) \wedge A(y_C^i)]$
$\leq n\ R.\neg A$	$\bigvee_{i=1}^{n+1} [A(y_C^i) \vee \bigvee_{j=i+1}^{n+1} y_C^i \approx y_C^j]$	$\bigwedge_{i=1}^{n+1} ar(R, x, y_C^i)$

Table 3.2: Translation of the RBox $\mathcal{R}$ and TBox $\mathcal{T}$ of a normalized $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$ into a set $\Xi(\mathcal{K})$ of DL-clauses. The variables $y_C^{(i)}$ are unique for the concept C (and i) and different from x.
Source: [13] (Table 3, slightly modified)

Definition 3.4.1. Let $\Pi(\mathcal{A})$ denote the set of DL-clauses obtained from $\mathcal{A}$ by applying the conversion rules shown in Table 3.3. The set of DL-clauses $DL(\mathcal{K})$ obtained from a normalized $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ is then defined as $DL(\mathcal{K}) = \Xi(\mathcal{K}) \cup \Pi(\mathcal{A})$.

Lemma 3.4.2. *$DL(\mathcal{K})$ is satisfiable if and only if $\Xi(\mathcal{K}) \cup \mathcal{A}$ is satisfiable.*

Proof. It is easy too see that the conversion rules in Table 3.3 transform each assertion in $\mathcal{A}$ into an equisatisfiable DL-clause, so $\Pi(\mathcal{A})$ and $\mathcal{A}$ are equisatisfiable. As a result, $DL(\mathcal{K}) = \Xi(\mathcal{K}) \cup \Pi(\mathcal{A})$ and $\Xi(\mathcal{K}) \cup \mathcal{A}$ are equisatisfiable as well. □

ABox assertion	DL-clause
$C(a)$	$C(a) \leftarrow$
$\neg A(a)$	$\leftarrow A(a)$
$R(a, b)$	$R(a, b) \leftarrow$
$a \simeq b$	$a \simeq b \leftarrow$
$a \not\simeq b$	$\leftarrow a \simeq b$

Table 3.3: Translating the ABox assertions of a normalized $\mathcal{ALCHIQ}$ KB into DL-clauses. C denotes an atomic concept or a $\geq$-number restriction, A an atomic concept, R an atomic role, and a and b are individuals.

3.4.2 Example

As an illustration of the procedure described in the previous section, we will now translate the normalized $\mathcal{ALCHIQ}$ KB $\Delta(\mathcal{K})$ from section 3.2.2 into a set of DL-clauses $DL(\Delta(\mathcal{K}))$. The RBox and TBox were defined as follows:

$$\begin{aligned}\Delta(\mathcal{K})_{\mathcal{R}} = \{&\mathit{hasChild} \sqsubseteq \mathit{relatedTo},\\ &\mathit{relatedTo} \sqsubseteq \mathit{relatedTo}^{-}\}\end{aligned}$$

$$\begin{aligned}\Delta(\mathcal{K})_{\mathcal{T}} = \{&\top \sqsubseteq \neg \mathit{HappyMother} \sqcup C,\\ &\top \sqsubseteq \neg C \sqcup \mathit{Woman},\\ &\top \sqsubseteq \neg C \sqcup (\geq 2\ \mathit{hasChild}.\top),\\ &\top \sqsubseteq \neg C \sqcup (\leq 4\ \mathit{hasChild}.\top),\\ &\top \sqsubseteq \neg C \sqcup \forall \mathit{hasChild}.\mathit{HappyChild},\\ &\top \sqsubseteq \neg \mathit{Woman} \sqcup (\leq 1\ \mathit{hasChild}.\top) \sqcup (\geq 5\ \mathit{hasChild}.\top) \sqcup\\ &\quad (\geq 1\ \mathit{hasChild}.\neg \mathit{HappyChild}) \sqcup \mathit{HappyMother}\}\end{aligned}$$

The resulting DL-clauses are shown below:

$$\begin{aligned}\Xi(\Delta(\mathcal{K})) = \{&\mathit{relatedTo}(x,y) \leftarrow \mathit{hasChild}(x,y),\\ &\mathit{relatedTo}(y,x) \leftarrow \mathit{relatedTo}(x,y),\\ &C(x) \leftarrow \mathit{HappyMother}(x),\\ &\mathit{Woman}(x) \leftarrow C(x),\\ &(\geq 2\ \mathit{hasChild}.\top) \leftarrow C(x),\\ &(\leq 4\ \mathit{hasChild}.\top) \leftarrow C(x),\\ &\forall \mathit{hasChild}.\mathit{HappyChild} \leftarrow C(x),\\ &y_1 \simeq y_2 \vee (\geq 5\ \mathit{hasChild}.\top(x))\\ &\vee(\geq 1\ \mathit{hasChild}.\neg \mathit{HappyChild}(x))\\ &\vee \mathit{HappyMother}(x) \leftarrow \mathit{Woman}(x) \wedge \mathit{hasChild}(x,y_1) \wedge \mathit{hasChild}(x,y_2)\}\end{aligned}$$

Most of the translations are straight-forward. The left-hand sides of the RBox axioms were moved to the bodies, whereas the right-hand sides constitute the head atoms. As for the TBox axioms, the atomic concepts and the $\geq$-number restrictions were moved to the heads of the clauses, whereas the negated atomic concepts appear in the bodies. However, the number restriction $(\leq 1\ \mathit{hasChild}.\top)$ caused atoms to be added to both the head ($y_1 \simeq y_2$) and the body ($\mathit{hasChild}(x, y_1)$ and $\mathit{hasChild}(x, y_2)$).

Now consider the ABox:

$$\begin{aligned}\Delta(\mathcal{K})_{\mathcal{A}} = \{&\top(\iota),\\ &\mathit{Woman}(\mathit{mary}),\\ &\mathit{HappyChild}(\mathit{tim}),\\ &\mathit{hasChild}(\mathit{mary}, \mathit{tim}),\\ &\mathit{hasChild}(\mathit{mary}, \mathit{timothy}),\\ &\mathit{tim} \simeq \mathit{timothy}\}\end{aligned}$$

Using the rules in Table 3.3, $\Delta(\mathcal{K})_{\mathcal{A}}$ is translated into the following set of DL-clauses:

$$\begin{aligned}\Pi(\Delta(\mathcal{K})_{\mathcal{A}}) = \{&\top(\iota) \leftarrow,\\ &\mathit{Woman}(\mathit{mary}) \leftarrow,\\ &\mathit{HappyChild}(\mathit{tim}) \leftarrow,\\ &\mathit{hasChild}(\mathit{mary}, \mathit{tim}) \leftarrow,\\ &\mathit{hasChild}(\mathit{mary}, \mathit{timothy}) \leftarrow,\\ &\mathit{tim} \simeq \mathit{timothy} \leftarrow\}\end{aligned}$$

The complete set of DL-clauses for $\Delta(\mathcal{K})$ is then obtained by $DL(\Delta(\mathcal{K})) = \Xi(\Delta(\mathcal{K})) \cup \Pi(\Delta(\mathcal{K})_{\mathcal{A}})$.

3.5 From DL-Clauses to DLE-Clauses

The translation from DL-clauses to DLE-clauses uses the *range restriction* method described in [5]. First, it introduces a fresh predicate *dom*. In the context of DLE-clauses, *dom* will be interpreted as the universal concept, so we always have $\mathit{dom}^I = \Delta^I$. Next, given a clause C, the body atoms $\mathit{dom}(x_1), \ldots, \mathit{dom}(x_n)$, where $x_1, \ldots, x_n$ are the variables that occur in C's head but not in its body, are added to C. In a clause set S that only contains function symbols of arity zero (constants or, as we will call them in the context of DLE-clauses, individuals), all that remains to be done is to add the unit clause $(\mathit{dom}(a) \leftarrow)$ to S for every constant a in S. When a fresh constant b is introduced to a DLE-clause set (as will be done by the DLE-Hyper Tableau calculus's *at-least* rule [section 5.2]), the clause $(\mathit{dom}(b) \leftarrow)$ will be added to the set as well. As *dom* is always interpreted as the universal concept, these clauses will be trivially true.

Note that DL-clauses do not contain function symbols of non-zero arity. If they did, the range restriction process would be slightly more complex, as described in [5].

As seen in Table 3.2, x is the only variable in a DL-clause set that may appear in a head atom without also appearing in a body atom, so we need to add at most one *dom* atom to the body of a clause.

Definition 3.5.1. Given a set of DL-clauses $DL(\mathcal{K})$ for a normalized $\mathcal{ALCHIQ}$ KB $\mathcal{K}$, the *non-restricted* set of DLE-clauses $DLE'(\mathcal{K})$ is obtained by replacing all atoms of the form $C(s)$ and $R(s,t)$ in $DL(\mathcal{K})$ by $C(s) \simeq \mathbf{t}$ and $R(s,t) \simeq \mathbf{t}$, respectively. Let *dom* denote a fresh concept not occurring in $DLE'(\mathcal{K})$. The final, *range-restricted* set of DLE-clauses $DLE(\mathcal{K})$ is then obtained from $DLE'(\mathcal{K})$ by (1) adding the body atom $\mathit{dom}(x)$ to every clause whose head contains the variable x and whose body does not contain x, and (2) adding the unit clause $(\mathit{dom}(a) \leftarrow)$ for every individual a in $DLE'(\mathcal{K})$.

Lemma 3.5.1. *$DLE(\mathcal{K})$ is satisfiable if and only if $DL(\mathcal{K})$ is satisfiable.*

Proof. The semantics of DL-clauses and DLE-clauses are essentially equivalent, so $DLE'(\mathcal{K})$ and $DL(\mathcal{K})$ are clearly equisatisfiable. The range restriction method used to obtain $DLE(\mathcal{K})$ was shown to be sound and complete in [11]. While the DLE-Hyper Tableau calculus will introduce new constants to the clause set, this is not a problem because it will always add the corresponding *dom* unit clauses and interpret *dom* as the universal concept to preserve the soundness and completeness of range restriction. As a result, $DLE(\mathcal{K})$ and $DLE'(\mathcal{K})$ must be equisatisfiable, which implies that $DLE(\mathcal{K})$ and $DL(\mathcal{K})$ are equisatisfiable as well. □

Theorem 3.5.1. *A $\mathcal{SHIQ}$ KB $\mathcal{K}$ is satisfiable if and only if the DLE-clause set $DLE(\Delta(\Omega(\mathcal{K})))$ is satisfiable.*

Proof. Lemma 3.1.1 states that $\mathcal{K}$ and $\Omega(\mathcal{K})$ are equisatisfiable. By Lemma 3.2.1, $\Omega(\mathcal{K})$ and $\Delta(\Omega(\mathcal{K}))$ are equisatisfiable as well. Lemma 3.4.1 then proves that $\Delta(\Omega(\mathcal{K}))$ and $\Xi(\Delta(\Omega(\mathcal{K}))) \cup \mathcal{A}$, where $\mathcal{A}$

denotes the ABox of $\Delta(\Omega(\mathcal{K}))$, are equisatisfiable. $\Xi(\Delta(\Omega(\mathcal{K}))) \cup \mathcal{A}$ and the corresponding set of DL-clauses $DL(\Delta(\Omega(\mathcal{K})))$ are shown to be equisatisfiable in Lemma 3.4.2. Finally, by Lemma 3.5.1, $DL(\Delta(\Omega(\mathcal{K})))$ and $DLE(\Delta(\Omega(\mathcal{K})))$ are also equisatisfiable. Since the input and the output of each step in the transformation of $\mathcal{K}$ into $DLE(\Delta(\Omega(\mathcal{K})))$ are equisatisfiable, the Lemma's statement must hold. □

Chapter 4

The E-Hyper Tableau Calculus

The DLE-Hyper Tableau calculus is an extension of the E-Hyper Tableau calculus [7], a hypertableau-based proof procedure for first-order logic that was developed by Peter Baumgartner, Ulrich Furbach and Björn Pelzer at National ICT Australia and University of Koblenz. Unlike the standard Hyper Tableau calculus [6] on which it was based, the E-Hyper Tableau calculus offers an efficient, superposition-based handling of equality inspired by earlier work in saturation-based theorem proving [4]. The calculus is currently implemented in E-KRHyper [16]. This thesis, however, focuses on its theoretical aspects as they will be vital to understanding the next chapter, which will finally introduce the DLE-Hyper Tableau calculus.

Note that this chapter is based almost entirely on [7] and [15], with most of the definitions in the next section being almost verbatim copies of the corresponding definitions in [15]. In other words, most of this is *not* original work, apart from the practical examples that I added to illustrate some important concepts.

4.1 Preliminaries

4.1.1 Terms and Substitutions

Let $\Sigma = (\mathcal{F}, \mathcal{P})$ denote a first-order signature consisting of a set of *function symbols* $\mathcal{F}$ and a set of *predicate symbols* $\mathcal{P}$, with $\mathcal{F}$ and $\mathcal{P}$ being infinite and disjoint. Each symbol has a fixed arity. A *constant* is a function symbol with arity zero. Furthermore, let $\mathcal{X}$ be an infinite set of variables that is disjoint from $\mathcal{F}$ and $\mathcal{P}$. The set of Σ-*terms* is then defined inductively as follows:

1. Every constant $c \in \mathcal{F}$ and every variable $x \in \mathcal{X}$ is a term.

2. If $t_1, \ldots, t_n$ are terms and $f \in \mathcal{F}$ is a function symbol with arity n, then $f(t_1, \ldots, t_n)$ is also a term.

Given a term $t = f(t_1, \ldots, t_n)$, s is a *subterm* of t iff

- $s = t$, or
- s is a subterm of t_i for some i with $1 \leq i \leq n$

$vars(t)$ denotes the set of variables occurring in the term t. t is *ground* iff $vars(t) = \emptyset$.

A *position* p is a sequence of natural numbers. Given a term t, $t|_p$ denotes the subterm of t at position p. Let $t = f(t_1, \ldots, t_n)$. If ϵ is an empty sequence, then $t|_\epsilon = t$. For $1 \leq i \leq n$, $t|_{i.p}$ is defined as $t_i|_p$. The example below uses the term t to illustrate which subterms certain positions refer to:

$$\begin{aligned} t &= f(a, g(a, b, h(c)), h(d)) \\ t|_3 &= h(d) \\ t|_{2.2} &= b \\ t|_{2.3.1} &= c \end{aligned}$$

The notation $t[s]_p$ means that t contains the subterm s at position p, i.e., $t|_p = s$. $t[p/s']$ stands for the term obtained from t by replacing the subterm $t|_p$ by s' at position p. In the above example we have $t[2/b] = f(a, b, h(d))$. If the position is obvious from the context, $t[s]$ states that t contains a subterm s at a certain position, while $t[s']$ then denotes the term obtained from t by replacing s by s' at that position.

A *substitution* σ maps variables in $\mathcal{X}$ to terms in $\mathcal{T}$. Its *domain* is defined as $dom(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}$ while $ran(\sigma) = \{x\sigma \in \mathcal{X} \mid x\sigma \neq x\}$ denotes its *range* (both sets are finite). If $vars(ran(\sigma)) = \emptyset$, then σ is called a *ground substitution*. If σ is a bijection of $\mathcal{X}$ onto itself, it is also called a *renaming*. Furthermore, σ is a *unifier* for two terms s and t if $s\sigma = t\sigma$. It is the *most general unifier (mgu)* if every other unifier θ of s and t is an *instance* of σ, meaning that for each unifier θ there exists a substitution δ such that $\sigma\delta = \theta$. In the context of terms, s is an *instance* of t if $s\sigma = t$ for some substitution σ. If there exists a renaming ρ with $s\rho = t$, then s is called a *variant* of t.

4.1.2 Term Rewriting

This section covers the basics of term rewrite systems. Unlike the rest of this chapter, it is based on [4] (section 5.1.1).

A *rewrite rule* $l \Rightarrow r$ is an ordered pair of terms. A *rewrite relation* $\Rightarrow$ is a binary relation that is

- closed under substitution ($s \Rightarrow t$ implies $s\sigma \Rightarrow t\sigma$ for any substitution σ), and
- closed under context application ($s \Rightarrow t$ implies $u[s] \Rightarrow u[t]$ for any term u)

Given a set of rewrite rules R, the *induced* rewrite relation $\Rightarrow_R$ denotes the smallest rewrite relation that includes R. The converse relation is denoted by $\Leftarrow_R$, while $\Leftrightarrow_R$ and $\Leftrightarrow_R^*$ denote the symmetric closure and the equivalence closure, respectively. R *rewrites* a term u to another term v if there exists a rule $s \Rightarrow t \in R$ such that $u = u[s\sigma]$ and $v = u[t\sigma]$ for some substitution σ. If a term cannot be rewritten by R, it is called *irreducible w.r.t.* R. If the term t' is irreducible w.r.t. R and $t \Rightarrow_R^* t'$ for some term t, then t' is called a *normal form* of t.

A *reduction relation* is a well-founded rewrite relation, which means that every term has a normal form and there are no infinite sequences of rewrite steps. A rewrite system is called *terminating* if it induces a well-founded rewrite relation. It is said to be *confluent* if $s \Rightarrow^* t$ and $s \Rightarrow^* u$ implies $t \Rightarrow^* v$ and $u \Rightarrow^* v$ for some v. This means that whenever a term s can be rewritten to two different terms t and u, it must be possible to rewrite both t and u to the same term v (the equation $t \approx u$ *converges*).

A rewrite system that is both terminating and confluent is called *convergent*. In a convergent rewrite system R, two terms s and t are considered equivalent ($s \Leftrightarrow_R^* t$) iff they have the same normal form, i.e., iff $s \approx t$ converges.

Finally, a rewrite system R is *lhs-irreducible* if no left-hand side of any rule in it can be rewritten by another rule. More formally, $s \Rightarrow t \in R$ implies that there is no rule $u[s] \Rightarrow v \in R$ for any terms u and v with $u \neq s$.

4.1.3 Atoms, Literals and Clauses

An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ for p an n-ary predicate symbol and $t_1, \ldots, t_n$ terms. A *literal* is an atom or a negated atom. Given a literal L, $\overline{L}$ denotes the complement of L. A literal is ground iff all terms occurring in it are ground.

The E-Hyper Tableau calculus assumes that the only predicate symbol in Σ is the equality symbol $\simeq$. An arbitrary first-order atom A that is not an equation can simply be turned into the equation $A \simeq \mathbf{t}$, where $\mathbf{t}$ denotes a special constant that does not appear anywhere else. (However, for the sake of simplicity, atoms such as $P(x) \simeq \mathbf{t}$ will still be written as $P(x)$.) This means that from now on, an atom will always be assumed to be an equation of the form $s \simeq t$, for s and t Σ-terms, whereas a literal may also be a negative equation, in which case it is written as $s \not\simeq t$. Note that the atom $s \simeq t$ will also be used to denote its symmetric variant $t \simeq s$. In other words, $s \simeq t$ always stands for '$s \simeq t$ or $t \simeq s$'.

A *clause* is a finite multiset of literals. In this thesis, clauses will usually be written as implications of the form $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$, where $A_1, \ldots, A_m$ ($m \geq 0$) denote the *head atoms* or *head literals* and $B_1, \ldots, B_n$ ($n \geq 0$) are the *body atoms* or *body literals*. The *head* of a clause is the multiset of its head literals, while the *body* is the multiset of the body literals. Hence, a clause may be written as $\mathcal{A} \leftarrow \mathcal{B}$, where $\mathcal{A}$ and $\mathcal{B}$ denote the head and the body, respectively. The notation $A, \mathcal{A} \leftarrow B, \mathcal{B}$ is used to denote the clause with head $\mathcal{A} \cup \{A\}$ and body $\mathcal{B} \cup \{B\}$. All variables in a clause are implicitly universally quantified.

A *unit clause* contains exactly one literal, with *positive unit clauses* containing a head literal and *negative unit clauses* containing a body literal. $\square$ denotes the *empty clause*, which does not contain any literals. It can also be written as $\leftarrow$. A clause is ground if all of its literals are ground. It is *pure* if no variable occurs in more than one distinct head literal. The clause $A_1, \ldots, A_m \leftarrow B_1, \ldots, B_n$ would be pure iff $vars(A_i) \cap vars(A_j) = \emptyset$ for all $i, j \in \{0, \ldots, m\}$ with $i \neq j$. A substitution π is a *purifying substitution* for a clause C iff $C\pi$ is pure.

4.1.4 Orderings

The E-Hyper Tableau calculus requires a *reduction ordering* $\prec$ that is total on the set of ground Σ-terms. A reduction ordering $\prec$ on a set of terms $\mathcal{T}$ is defined as a strict partial ordering that is

1. well-founded (i.e., every subset of $\mathcal{T}$ has a minimal element),
2. stable under context (i.e., $s \prec s'$ implies $t[p/s] \prec t[p/s']$ for any terms s, s' and t and any position p in t), and
3. stable under substitution (i.e., $s \prec t$ implies $s\sigma \prec t\sigma$ for all terms s and t any substitution σ)

$\preceq$ denotes the non-strict ordering induced by $\prec$, with $\succ$ and $\succeq$ denoting the converse of $\prec$ and $\preceq$, respectively. In order to compare atoms and clauses, the *multiset extension* $\prec^{mul}$ of $\prec$ is used. It is defined as follows for M and N multisets of terms:

$$M \prec^{mul} N \text{ iff } M \neq N \text{ and for all } m \in M \backslash N \text{ there exists an } n \in N \backslash M \text{ with } n \succ m$$

Each head atom $s \simeq t$ in a clause is assigned the multiset $\{s, t\}$ whereas each body atom $u \simeq v$ is assigned the multiset $\{u, u, v, v\}$. As we will see in a moment, this ensures that body atoms are bigger than identical head atoms. Two ground atoms A and B, each consisting of two terms (the two sides of the equation), are then compared using $\prec^{mul}$ (from now on, we will simply use $\prec$ to denote $\prec^{mul}$ as well) as follows: first, the bigger term in A is compared with the bigger term in B, with the atom containing the biggest term being bigger in $\prec$; if the terms are identical, the signs of the atoms are compared, with body atoms being bigger than head atoms; if the signs are identical as well, then the smaller terms in A and B are compared to determine the bigger of the two atoms.

The two-fold multiset extension of $\prec$ (which will also be denoted by $\prec$) is used to compare clauses. This means that comparing two clauses C and D is similar to comparing atoms: first, the biggest atoms in C and D are compared; if they are identical and occur the same number of times in both C and D, the next smallest atoms in the two clauses are considered and so on.

4.1.5 Interpretations

A *Herbrand interpretation* I is a set of ground Σ-equations, namely those equations that are considered true in I. Let F denote a ground Σ-literal, a Σ-clause or a set of Σ-clauses; then, $I \models F$ denotes that I satisfies F.

An *E-interpretation* is an interpretation that is also a congruence relation on the set of Σ-terms. I^E denotes the E-interpretation induced by I, i.e., the smallest congruence relation on Σ-terms that includes I. $I \models_E F$ denotes the fact that I *E-satisfies* F, which means $I^E \models F$. $F \models_E F'$ holds iff every E-interpretation that satisfies F also satisfies F', in which case we say that F *E-entails* F'.

4.1.6 Redundant Clauses

If D is a ground clause and S is a set of clauses, then S_D denotes the set of ground clauses or ground instances of clauses in S that are smaller than D. A ground clause D is redundant w.r.t. a set of clauses S iff $S_D \models_E D$ (more intuitively: iff D follows from smaller clauses in S). A non-ground clause D is redundant w.r.t. S iff $S_{D'} \models_E D'$ for every ground instance D' of D.

For example, given the clause $D = (A(b) \leftarrow)$, the clause set $S = \{(A(a) \leftarrow), (a \simeq b \leftarrow)\}$ and a reduction ordering $\prec$ such that $a \prec b$, we have $S_D = S$. Since S clearly E-entails $A(b)$, D is redundant w.r.t. S. If b was smaller than a in $\prec$, D would still follow from S, but it would be bigger than $(A(a) \leftarrow)$. As a result, we would have $S_D = \{(a \simeq b \leftarrow)\}$, which clearly does not E-entail D, so D would *not* be redundant w.r.t. S.

As another example, consider the case where $D = (b \simeq c \leftarrow)$, $S = \{(a \simeq b \leftarrow), (a \simeq c \leftarrow)\}$, and $a \prec b \prec c$. Here we once again have $S_D = S$ and $S \models_E D$, so D is redundant w.r.t. S.

4.2 Inference Rules

The E-Hyper Tableau calculus has four inference rules, with the first three presented here being based on the superposition calculus in [4]. Again, it is important to remember that the atom $s \simeq t$ also stands for $t \simeq s$.

The first inference rule is *sup-left* (*superposition-left*), which applies superposition to a body literal.

$$\text{sup-left}(\sigma)\ \frac{\mathcal{A} \leftarrow s[l'] \simeq t, \mathcal{B} \quad l \simeq r \leftarrow}{(\mathcal{A} \leftarrow s[r] \simeq t, \mathcal{B})\sigma}\ \text{if}\ \begin{cases} l' \text{ is not a variable,} \\ \sigma \text{ is a mgu of } l \text{ and } l', \\ l\sigma \not\preceq r\sigma, \text{ and} \\ s\sigma \not\preceq t\sigma \end{cases}$$

The upper left clause is called the rule's *left premise*, with the upper right clause denoting the *right premise*. The resulting clause is called the *conclusion*. A *sup-left* inference with left premise C, right premise D, substitution σ, and conclusion E is represented by $C, D \Rightarrow_{\text{sup-left}(\sigma)} E$. Examples of *sup-left* inferences are shown below. Note that, in contrast to the rest of this thesis, all atoms were written as equations this time.

$$(P(x) \simeq \mathbf{t} \leftarrow Q(x, f(y)) \simeq \mathbf{t}), (Q(a, f(b)) \simeq \mathbf{t} \leftarrow) \Rightarrow_{\text{sup-left}(\{x/a,\, y/b\})} (P(a) \simeq \mathbf{t} \leftarrow \mathbf{t} \simeq \mathbf{t})$$
$$(\leftarrow Q(x) \simeq \mathbf{t}), (Q(y) \simeq \mathbf{t} \leftarrow) \Rightarrow_{\text{sup-left}(\{x/y\})} (\leftarrow \mathbf{t} \simeq \mathbf{t})$$

Next up is the *unit-sup-right* rule (*unit superposition right*). Unlike *sup-left*, it applies superposition to a positive unit clause.

$$\text{unit-sup-right}(\sigma)\ \frac{s[l'] \simeq t \leftarrow \quad l \simeq r \leftarrow}{(s[r] \simeq t \leftarrow)\sigma}\ \text{if}\ \begin{cases} l' \text{ is not a variable,} \\ \sigma \text{ is a mgu of } l \text{ and } l', \\ (s[l'] \simeq t)\sigma \not\leq (l \simeq r)\sigma, \\ l\sigma \not\leq r\sigma, \text{ and} \\ s\sigma \not\leq t\sigma \end{cases}$$

Similar to *sup-left* inferences, *unit-sup-right* inferences are represented by $C, D \Rightarrow_{\text{unit-sup-right}(\sigma)} E$. Below are some examples, with the second one assuming $a \prec b \prec c$.

$$(P(f(x)) \simeq \mathbf{t} \leftarrow), (f(a) \simeq a \leftarrow) \Rightarrow_{\text{unit-sup-right}(\{x/a\})} (P(a) \simeq \mathbf{t} \leftarrow)$$
$$(b \simeq c \leftarrow), (a \simeq c \leftarrow) \Rightarrow_{\text{unit-sup-right}(\epsilon)} (a \simeq b \leftarrow)$$

Note that, in the second inference, the left premise and the right premise could not have been exchanged because $a \prec b \prec c$ implies that the right premise is smaller than the left premise. Also, the left premise needs to be interpreted as its symmetric variant $(c \simeq b \leftarrow)$ for the side conditions of the *unit-sup-right* rule to hold.

The *ref* rule (*reflexivity*) removes a body literal if its two sides can be unified.

$$\text{ref}(\sigma)\ \frac{A \leftarrow s \simeq t, B}{(A \leftarrow B)\sigma}\ \text{if } \sigma \text{ is a mgu of } s \text{ and } t$$

In the case of *ref*, there is no right premise, so the inferences are written as $C \Rightarrow_{\text{ref}(\sigma)} E$. Some examples are

$$(\leftarrow g(f(x), f(b)) \simeq g(f(a), y)) \Rightarrow_{\text{ref}(\{x/a,\, y/f(b)\})} (\leftarrow)$$
$$(\leftarrow \mathbf{t} \simeq \mathbf{t}) \Rightarrow_{\text{ref}(\epsilon)} (\leftarrow)$$

The *split* rule is applied to non-unit clauses with empty bodies. First, it applies a purifying substitution π to ensure that no variable is shared among head literals. The resulting instantiated head atoms are then returned as positive unit clauses.

$$\text{split}(\pi)\ \frac{A_1, ..., A_m \leftarrow}{A_1\pi \leftarrow \ ... \ A_m\pi \leftarrow}\ \text{if}\ \begin{cases} m \geq 2 \text{ and} \\ \pi \text{ is a purifying substitution for } A_1, ..., A_m \leftarrow \end{cases}$$

split inferences are denoted by $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$, where each $A_i \leftarrow$ represents one conclusion. Note that, in the following example, only x needs to be instantiated because y only occurs in one head literal.

$$(P(x), Q(x, y) \leftarrow) \Rightarrow_{\text{split}(\{x/a\})} (P(a) \leftarrow), (Q(a, y) \leftarrow)$$

4.3 Redundant Inferences and Saturation

An inference is ground iff all of its premises and all of its conclusions are ground. The substitution σ (or π, in the case of *split*) in a ground inference may be assumed to be the empty substitution ϵ. Given an inference $C, D \Rightarrow_{\text{sup-left}(\sigma)} E$ and a substitution γ such that $C\sigma\gamma, D\sigma\gamma \Rightarrow_{\text{sup-left}(\epsilon)} E\gamma$ is a ground inference, we say that the latter inference is a *ground instance* of the former one. For *unit-sup-right* and *ref* inferences, the definitions are analogous, except that the right premises D and $D\sigma\gamma$ are missing in the case of *ref*. As for *split*, if $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is a *split* inference and

$C\pi\gamma \Rightarrow_{\text{split}(\epsilon)} A_1\gamma \leftarrow, \ldots, A_m\gamma \leftarrow$ is a ground inference, then it is a ground instance of the former inference.

Given a clause set S, a ground inference $C, D \Rightarrow_{\text{sup-left}(\epsilon)} E$ or $C, D \Rightarrow_{\text{unit-sup-right}(\epsilon)} E$, or $C \Rightarrow_{\text{ref}(\epsilon)} E$ is redundant w.r.t. S iff E follows from $S_C \cup \{D\}$ (formally, $S_C \cup \{D\} \models_E E$), where $\{D\}$ is the empty set in the case of *ref*. A ground inference $C \Rightarrow_{\text{split}(\epsilon)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. S iff there is some i with $1 \leq i \leq m$ such that $(A_i \leftarrow)$ follows from S_C ($S_C \models_E (A_i \leftarrow)$). A non-ground inference is redundant w.r.t. S iff each of its ground instances is redundant w.r.t. S.

As mentioned in [7], all inferences work in a strictly order-decreasing way. As a result, when the conclusion of an inference applied to clauses from S is added to S, the inference will be redundant w.r.t. S afterwards. For example, consider the clause set $S = \{(P(b) \leftarrow), (a \simeq b \leftarrow)\}$ with $a \prec b$. Using *unit-sup-right* with left premise $C = (P(b) \leftarrow)$ and right premise $D = (a \simeq b \leftarrow)$, the clause $E = (P(a) \leftarrow)$ will be inferred and added to S. $E \in S$ as well as the fact that E is smaller than C due to $a \prec b$ imply $E \in S_C$. Hence, we now have $S_C \cup \{D\} \models_E E$, so the above *unit-sup-right* inference will be redundant after being applied once.

The following definition is of crucial importance in proving the completeness of the E-Hyper Tableau calculus (see [7] for the proof). When the DLE-Hyper Tableau calculus is introduced in chapter 5, it will be extended.

Definition 4.3.1 (**Saturation up to Redundancy [7]**). A clause set S is *saturated up to redundancy* iff for all clauses $C \in S$, such that C is not redundant w.r.t. S, all of the following hold:

- Every inference $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, ..., A_m \leftarrow$, such that $C\pi$ is not redundant w.r.t. S, is redundant w.r.t. S.
- Every inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{$ sup-left, sup-unit-right $\}$ and D is a fresh variant of a positive unit clause from S, such that neither $C\sigma$ nor $D\sigma$ is redundant w.r.t. S, is redundant w.r.t. S.
- Every inference $C \Rightarrow_{\text{ref}(\sigma)} E$, such that $C\sigma$ is not redundant w.r.t. S, is redundant w.r.t. S.

The fact that some clause set S is saturated up to redundancy basically means that it would not make sense to apply any more inferences because all relevant clauses have been derived. If S does not contain the empty clause, it will not be derived by further inferences either.

4.4 E-Hyper Tableaux

As defined in [7], a tableau $\mathbf{T}$ over a signature Σ is a labeled tree over the set of Σ-clauses. More precisely, $\mathbf{T} = (\mathbf{t}, \lambda)$, where $\mathbf{t}$ denotes a finite, ordered tree and λ denotes a labeling function that maps each node of $\mathbf{t}$ to some Σ-clause.

A branch $\mathbf{B}$ of length n in a tableau $\mathbf{T}$ is a sequence of nodes $(\mathbf{N}_1, \ldots, \mathbf{N}_n)$ (for $n \geq 0$), where $\mathbf{N}_1$ is the root and $\mathbf{N}_n$ is the leaf of $\mathbf{B}$. The clauses $\lambda(\mathbf{N}_i)$ (for $1 \leq i \leq n$) are called *clauses of* $\mathbf{B}$. The set of $\mathbf{B}$'s clauses, $\lambda(\mathbf{B})$, is then defined as $\lambda(\mathbf{B}) = \bigcup_{1 \leq i \leq n} \lambda(\mathbf{N}_i)$. For the sake of simplicity, we will also interpret $\mathbf{B}$ as the set of the clauses it contains and use $C \in \mathbf{B}$ rather than the more cumbersome $C \in \lambda(\mathbf{B})$ to refer to a clause C with which some node in $\mathbf{B}$ is labeled.

Given a tableau $\mathbf{T}$ containing a branch $\mathbf{B}$, the tableau obtained by adding an edge from the old leaf node of $\mathbf{B}$ to a new node labeled with a clause C is simply written as $\mathbf{B} \cdot C$. If $\mathbf{B}'$ is a sequence of nodes, we will write the tableau obtained by adding an edge from the leaf of $\mathbf{B}$ to the root of $\mathbf{B}'$ as $\mathbf{B} \cdot \mathbf{B}'$. The same notation will be used if $\mathbf{B}'$ denotes a set of n clauses, in which case it will be interpreted as a sequence of n nodes, each of which is labeled with a distinct clause from the clause set $\mathbf{B}'$. The order of the nodes in the sequence is irrelevant.

A branch is *closed* iff it contains the empty clause $\square$, and it is *open* iff it is not closed. Similarly, A tableau is closed iff each of its branches is closed, and it is open iff it is not closed.

4.5 Extension Rules

The E-Hyper Tableau calculus uses two derivation rules to extend branches in a tableau. The *Split* rule essentially encapsulates the *split* inference rule. The label d indicates that the resulting clauses are 'decision clauses'. In the next section, we will see why these labels are necessary.

$$\text{Split } \frac{\mathbf{B}}{\mathbf{B}\cdot A_1 \leftarrow^d \ \ldots \ \mathbf{B}\cdot A_m \leftarrow^d} \text{ if } \begin{cases} \text{there is a clause } C \in \mathbf{B} \text{ and} \\ \text{a substitution } \pi \text{ such that} \\ C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, ..., A_m \leftarrow \text{ and} \\ \mathbf{B} \text{ contains no variant of } A_i \leftarrow \\ \text{for any } 1 \leq i \leq m \end{cases}$$

The *Equality* rule is used for applying *sup-left*, *unit-sup-right* or *ref* to clauses in a branch.

$$\text{Equality } \frac{\mathbf{B}}{\mathbf{B}\cdot E} \text{ if } \begin{cases} \text{there is a clause } C \in \mathbf{B}, \\ \text{a fresh variant } D \text{ of a positive unit clause in } \mathbf{B}, \text{ and} \\ \text{a substitution } \sigma \text{ such that} \\ C, D \Rightarrow_{R(\sigma)} E \text{ with } R \in \{ \text{ sup-left, unit-sup-right } \} \text{ or} \\ C \Rightarrow_{\text{ref}(\sigma)} E, \text{ and} \\ \mathbf{B} \text{ contains no variant of } E \end{cases}$$

Note that an application of an inference rule to clauses from a branch $\mathbf{B}$ is *redundant* iff its conclusion (or, in the case of *split*, at least one of its conclusions) is redundant w.r.t. $\mathbf{B}$.

4.6 Deletion and Simplification Rules

The regular Hyper Tableau calculus [6] was *non-destructive*, meaning that the set of clauses in a branch could only be extended. In practice, however, it may be necessary to delete redundant clauses or replace them by smaller, simpler clauses, which is why the E-Hyper Tableau calculus has two rules for doing just that. The first rule, *Del* (*deletion*), replaces a clause by the trivial clause ($\mathbf{t} \simeq \mathbf{t} \leftarrow$) if it is redundant or *non-properly subsumed* by another clause in the same branch. Note that a clause C non-properly subsumes a clause D if there exists a substitution σ such that $C\sigma = D$.

$$\text{Del } \frac{\mathbf{B}\cdot C^{(d)}\cdot \mathbf{B}_1\cdot \mathbf{B}_2}{\mathbf{B}\cdot \mathbf{t} \simeq \mathbf{t} \leftarrow^{(d)}\cdot \mathbf{B}_1\cdot \mathbf{B}_2} \text{ if } \begin{cases} (1)\ C \text{ is redundant w.r.t. } \mathbf{B}\cdot \mathbf{B}_1, \text{ or some} \\ \text{clause in } \mathbf{B}\cdot \mathbf{B}_1 \text{ non-properly subsumes } C, \text{ and} \\ (2)\ \mathbf{B}_1 \text{ does not contain a decision clause} \end{cases}$$

The superscript (d) indicates that if the clause is labeled as a decision clause, the label will be preserved even though the actual clause is replaced. Note that the second constraint in the above definition is necessary to obtain a complete calculus. Without it, it would be possible to remove clauses that are shared by several branches but do not necessarily follow from clauses in *all* of these branches. Hence, removing such a clause may, in the worst case, prevent a contradiction from being uncovered in another branch.

The *Simp* rule (*simplification*) replaces a clause by another, smaller clause such that the removed clause is redundant w.r.t. the resulting branch.

$$\text{Simp}\ \frac{\mathbf{B}\cdot C^{(d)}\cdot \mathbf{B}_1\cdot \mathbf{B}_2}{\mathbf{B}\cdot D^{(d)}\cdot \mathbf{B}_1\cdot \mathbf{B}_2}\ \text{if}\ \begin{cases}\mathbf{B}\cdot C\cdot \mathbf{B}_1 \models_E D,\\ C \text{ is redundant w.r.t. } \mathbf{B}\cdot D\cdot \mathbf{B}_1, \text{ and}\\ \mathbf{B}_1 \text{ does not contain a decision clause}\end{cases}$$

Once again, it necessary to check whether $\mathbf{B}_1$ contains a decision clause. In this case, both the soundness and the completeness of the calculus would be affected otherwise.

4.7 Derivations

Before the notion of a derivation is defined, note that we will refer to *Split*, *Equality*, *Del*, and *Simp* as the *derivation rules* of the E-Hyper Tableau calculus. The *inference rules* introduced in section 4.2 are *not* considered derivation rules but will always be called inference rules.

An E-Hyper Tableau derivation of a set $\{C_1, \ldots, C_n\}$ of Σ-clauses is a possibly infinite sequence of tableaux $\mathbf{D} = (\mathbf{T}_i)_{0\leq i\leq \kappa}$ such that

1. $\mathbf{T}_0 = (\mathbf{t}_0, \lambda_0)$ is the clausal tableau over Σ that consists of a single branch $(\mathbf{N}_0, \ldots, \mathbf{N}_n)$ with $\lambda_0(\mathbf{N}_j) = C_j$ for all $1 \leq j \leq n$
2. for all $i > 0$, $\mathbf{T}_i$ is obtained from $\mathbf{T}_{i-1}$ by applying one of the E-Hyper Tableau calculus's derivation rules to some open branch of $\mathbf{T}_{i-1}$

As defined in section 4.4, a tableau $\mathbf{T}$ is a tuple $(\mathbf{t}, \lambda)$, where the tree $\mathbf{t}$ denotes a tuple $(\mathbf{N}, \mathbf{E})$ consisting of a set of nodes $\mathbf{N}$ and a set of edges $\mathbf{E}$. The *limit tree* of a derivation $\mathbf{D} = ((\mathbf{N}_i, \mathbf{E}_i), \lambda_i)_{i<\kappa}$ is then defined as $(\bigcup_{i<\kappa} \mathbf{N}_i, \bigcup_{i<\kappa} \mathbf{E}_i)$.

Now let $\mathbf{B} = (\mathbf{N}_i)_{i<\kappa}$ denote a possibly infinite branch with κ nodes in the limit tree $\mathbf{t}$ of some derivation. Also, let $\mathbf{B}_i = (\mathbf{N}_1, \ldots, \mathbf{N}_i)$ denote the sequence of the first i nodes in $\mathbf{B}$ for all $i < \kappa$. The set of *persistent clauses* $\mathbf{B}_\infty$ of $\mathbf{B}$ is then defined as follows:

$$\mathbf{B}_\infty = \bigcup_{i<\kappa} \bigcap_{i\leq j<\kappa} \lambda_j(\mathbf{B}_j)$$

Basically, $\mathbf{B}_\infty$ is the set of all those clauses in $\mathbf{B}$ that were not replaced by *Del* or *Simp*. We can then define the notion of an *exhausted branch*. Informally, a branch is called *exhausted* if applying further derivation rules would not help in finding a contradiction because all relevant clauses have been derived.

Definition 4.7.1 (Exhausted Branch [7]). Let $\mathbf{t}$ be a limit tree, and let $\mathbf{B} = (\mathbf{N}_i)_{i<\kappa}$ be a branch in $\mathbf{t}$ with κ nodes. The branch $\mathbf{B}$ is *exhausted* iff it does not contain the empty clause, and for every clause $C \in \mathbf{B}_\infty$ and every fresh variant D of every positive unit clause in $\mathbf{B}_\infty$ such that neither C nor D is redundant w.r.t. $\mathbf{B}_\infty$ all of the following hold, for all $i < \kappa$ such that $C \in \mathbf{B}_i$ and D is a variant of a clause in $\mathbf{B}_i$:

- if *Split* is applicable to $\mathbf{B}_i$ with underlying inference $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ and $C\pi$ is not redundant w.r.t. $\mathbf{B}_i$, then there is a $j < \kappa$ such that the inference $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, \ldots, A_m \leftarrow$ is redundant w.r.t. $\mathbf{B}_j$.
- if *Equality* is applicable to $\mathbf{B}_i$ with underlying inference $C, D \Rightarrow_{\text{R}(\sigma)} E$, for some R $\in$ { sup-left, unit-sup-right }, and neither $C\sigma$ nor $D\sigma$ is redundant w.r.t. $\mathbf{B}_i$, then there is a $j < \kappa$ such that the inference $C, D \Rightarrow_{\text{R}(\sigma)} E$ is redundant w.r.t. $\mathbf{B}_j$.
- if *Equality* is applicable to $\mathbf{B}_i$ with underlying inference $C \Rightarrow_{\text{ref}(\sigma)} E$ and $C\sigma$ is not redundant w.r.t. $\mathbf{B}_i$, then there is a $j < \kappa$ such that the inference $C \Rightarrow_{\text{ref}(\sigma)} E$ is redundant w.r.t. $\mathbf{B}_j$.

A *refutation* of a clause set S is a finite derivation of S that ends in a closed tableau. A derivation is *fair* iff it is a refutation or iff its limit tree has an exhausted branch.

Figure 4.1 contains an example E-Hyper Tableau.

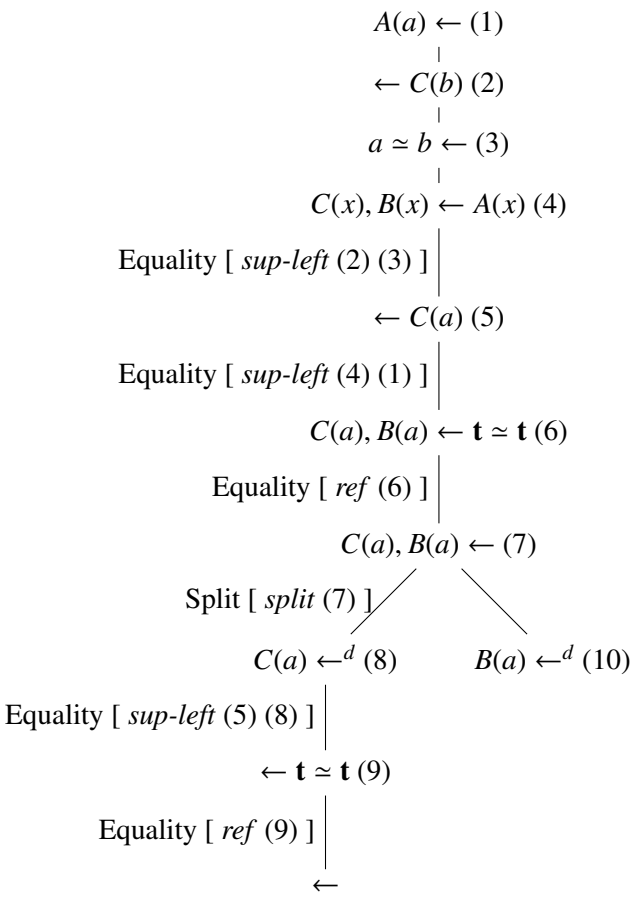

Figure 4.1: Example of an E-Hyper Tableau. The initial clause set consisted of clauses (1)-(4), with $a \prec b$. The labels next to the edges indicate which derivation and inference rules were applied to which clauses to derive the clause of the next node. The left branch is closed whereas the right one is open and exhausted. Note that clause (2) is redundant w.r.t. (3) and (5), so *Del* could replace it by the trivially true clause ($\mathbf{t} \simeq \mathbf{t} \leftarrow$).

4.8 Soundness and Completeness

The E-Hyper Tableau calculus was proven to be sound and complete in [7]. The statements of the most important theorems and propositions are repeated here so they can be referred to later on. Note that some of the identifiers were changed.

Theorem 4.8.1 (**Static Completeness [7]).** Let S be a clause set that is saturated up to redundancy. If $\square \notin S$, then S is satisfiable.

Theorem 4.8.2 (**Correctness of E-Hyper Tableaux [7]).** Let S be a clause set that has a refutation. Then S is E-unsatisfiable.

Proposition 4.8.1 (**Exhausted branches are saturated up to redundancy [7]).** If $\mathbf{B}$ is an exhausted branch of a limit tree of some fair derivation, then $\mathbf{B}_\infty$ is saturated up to redundancy.

Theorem 4.8.3 (**Completeness of E-Hyper Tableaux [7]).** Let S be a clause set and $\mathbf{t}$ be the limit tree of a fair derivation $\mathbf{D}$ of S. If $\mathbf{D}$ is not a refutation, then S is satisfiable.

4.9 Model Construction

While this section will not cover the details of the completeness proof in [7], the model construction method used there is helpful in proving the completeness of the DLE-Hyper Tableau calculus, so it will briefly be introduced here.

Let S denote a possibly infinite clause set. By induction on the term ordering $\prec$, sets of rewrite rules ε_C and R_C are defined for each positive Σ-clause C. Suppose that ε_D has been defined for all ground Σ-clauses D with $D \prec C$. Let $R_C = \bigcup_{D \prec C} \varepsilon_D$. ε_C is then defined as follows:

$$\varepsilon_C = \begin{cases} \{l \Rightarrow r\} & \text{if } C = (l \simeq r \leftarrow) \text{ is a ground instance of a positive unit clause in } S\text{, } l \succ r, \\ & \text{and } l \text{ is irreducible w.r.t. } R_C \\ \emptyset & \text{otherwise} \end{cases}$$

The rewrite system induced by S is defined as $R_S = \bigcup_C \varepsilon_C$, where C ranges over all ground Σ-clauses. As shown in [7], R_S must be a convergent rewrite system. An atom $s \simeq t$ is true in R_S (written as $R_S \models_E s \simeq t$) iff s and t reduce to the same normal form in R_S. Given a clause $C = (\mathcal{A} \leftarrow \mathcal{B})$, $R_S \models_E C$ iff $R_S \models_E A$ for some $A \in \mathcal{A}$ or $R_S \not\models_E B$ for some $B \in \mathcal{B}$. Finally, $R_S \models_E S'$ for some clause set S' iff $R_S \models_E C$ for every clause $C \in S'$. As shown in the completeness proof in [7], the rewrite system induced by the set of persistent clauses $\mathbf{B}_\infty$ of an exhausted branch $\mathbf{B}$ is a model of $\mathbf{B}$'s clause set (i.e., $R_{\mathbf{B}_\infty} \models_E \mathbf{B}$).

Chapter 5

The DLE-Hyper Tableau Calculus

As the regular E-Hyper Tableau calculus has no means of correctly handling the ≥-number restrictions occurring in DLE-clauses, it was equipped with a new inference rule (*at-least*) and a corresponding extension rule (*At-Least*) to remedy this shortcoming. The resulting calculus is called the DLE-Hyper Tableau calculus. This chapter describes the changes that had to be made to the E-Hyper Tableau calculus in order to obtain the new calculus, which is capable of processing DLE-clauses derived from $\mathcal{SHIQ}$ knowledge bases. In the next chapter, we will then prove that the DLE-Hyper Tableau calculus is in fact a decision procedure for $\mathcal{SHIQ}$.

Unlike the previous chapter, the contents of this chapter are entirely original work. Note, however, that Björn Pelzer, who already implemented the E-Hyper Tableau calculus in E-KRHyper [16], took care of the actual implementation of the theoretical concepts presented here.

5.1 Preliminaries

All of the inference and derivation rules of the E-Hyper Tableau calculus are still defined the same way. The definitions of branches, tableaux and derivations also remain unchanged except for the fact that DLE-Hyper Tableau branches and derivations are finite since the calculus terminates (as will be shown in section 6.2). Furthermore, since DLE-clauses are equational clauses, they can be handled by the E-Hyper Tableau calculus's inference rules without a problem, although the additional rules of the DLE-Hyper Tableau calculus are needed due to the ≥-number restrictions. The concepts and individuals in DLE-clauses will be represented by first-order predicates and constants, respectively. (Note that, technically, $\simeq$ is still the only true predicate symbol.) Since this thesis focuses on the theoretical aspects of the DLE-Hyper Tableau calculus, ≥-number restrictions will simply be written in the form $\geq n\ R.C$, as in the chapters about Description Logics. In practice, they may be represented by predicates such *atleast_n_R_C*.

Given a DLE-clause set S, we assume that each ≥-number restriction is assigned a distinct label l before the derivation begins. When an inference rule is applied to a clause containing a ≥-number restriction, the label is preserved (e.g., $(\geq 1\ R.C_l(a) \leftarrow), (a \simeq b \leftarrow) \Rightarrow_{\text{unit-sup-right}(\epsilon)} (\geq 1\ R.C_l(b) \leftarrow)$, where the subscript l indicates the label). As we shall see in a moment, the labels are necessary for the new *at-least* rule to work correctly.

Let N_S denote the set of individuals occurring in S. The set of all individuals N_X is then defined inductively:

1. $N_I \subseteq N_X$

2. if $a \in N_X$ and $\geq n\ R.C_l$ is a ≥-number restriction occurring in S, then $a.l.1, \ldots, a.l.n \in N_X$

If the name of any individual added in the second step clashes with the name of an individual, concept or role in S, then the problematic identifier in S must be renamed appropriately before the DLE-Hyper Tableau calculus can be applied. Since the DLE-Hyper Tableau calculus, like the E-Hyper Tableau calculus, uses a signature Σ with a set of function symbols $\mathcal{F}$, and since individuals in DLE-clauses are represented by first-order constants, we assume that N_X is a subset of $\mathcal{F}$. In fact, we even assume $\mathcal{F} = N_X$, as DLE-clauses do not contain function symbols of non-zero arity.

5.2 The *at-least* Inference Rule

The DLE-Hyper Tableau calculus handles $\geq$-number restrictions with the help of a new inference rule called *at-least*. In order to be able to treat atomic roles and inverse roles R the same way when describing the rule, we will use the *ar* function as defined in Table 3.2:

$$ar(R, s, t) = \begin{cases} R(s,t) & \text{if } R \text{ is an atomic role} \\ S(t,s) & \text{if } R \text{ is an inverse role and } R = S^- \end{cases}$$

Also, to simplify things, we will define a function *cc* ('concept clause') that expects a literal concept B and an individual a as arguments and generates a positive or negative concept clause. Let A denote an atomic concept; then

$$cc(B, a) = \begin{cases} A(a) \leftarrow & \text{if } B = A \\ \leftarrow A(a) & \text{if } B = \neg A \end{cases}$$

The *at-least* inference rule is defined as follows, for a and $a.l.i$ (with $1 \leq i \leq n$) individuals, R an atomic or inverse role and B a literal concept:

$$\text{at-least} \ \frac{(\geq n\ R.B_l(a) \leftarrow)}{\begin{array}{l} \bigcup_{1 \leq i \leq n} \{ ar(R, a, a.l.i) \leftarrow, \\ \qquad cc(B, a.l.i), \\ \qquad dom(a.l.i) \leftarrow \} \cup \\ \bigcup_{1 \leq i < j \leq n} \{ \leftarrow a.l.i \simeq a.l.j \} \end{array}}$$

at-least inferences with premise C and conclusion E, where E denotes a set of clauses, are represented by $C \Rightarrow_{\text{at-least}} E$. The rule basically 'expands' the number restriction by generating n fresh, pairwise distinct R-successors of a that must satisfy the concept B, as implied by the clauses in the conclusion. The *dom* clauses are needed because, as described in section 3.5, a range-restricted clause set S must contain the clause $(dom(a) \leftarrow)$ for every constant a in S in order for range restriction to be sound and complete.

The reasoning behind the use of the labels should become clear now: without them, *at-least* would generate the same successors $a.1, \ldots, a.n$ for both $(\geq n\ R.A(a) \leftarrow)$ and $(\geq n\ R.\neg A(a) \leftarrow)$, for example, which would lead to a contradiction even though the two clauses are not inherently contradictory. In the rest of this thesis, the labels both in *at-least* clauses and in the names of successors will sometimes be omitted since the full notation can become cumbersome at times. For example, when dealing with a case where *at-least* is applied to $(\geq 1\ R.B(a) \leftarrow)$ (implicitly labeled with l), the resulting successor may simply be denoted by a' even though its actual name would be $a.l.1$.

An example of an application of *at-least* is shown below, where R is an atomic role, A an atomic concept, and the premise is labeled with l.

$$\begin{aligned}\geq 3\ R.A_l(a) \Rightarrow_{\text{at-least}} \{&R(a, a.l.1) \leftarrow,\\ &R(a, a.l.2) \leftarrow,\\ &R(a, a.l.3) \leftarrow,\\ &A(a.l.1) \leftarrow,\\ &A(a.l.2) \leftarrow,\\ &A(a.l.3) \leftarrow,\\ &dom(a.l.1) \leftarrow,\\ &dom(a.l.2) \leftarrow,\\ &dom(a.l.3) \leftarrow,\\ &\leftarrow a.l.1 \simeq a.l.2,\\ &\leftarrow a.l.1 \simeq a.l.3,\\ &\leftarrow a.l.2 \simeq a.l.3\}\end{aligned}$$

The following example illustrates the effects of applying *at-least* to an at-least clause containing an inverse role as well as a negated atomic concept:

$$\begin{aligned}\geq 2\ R^-.\neg A_l(a) \Rightarrow_{\text{at-least}} \{&R(a.l.1, a) \leftarrow,\\ &R(a.l.2, a) \leftarrow,\\ &\leftarrow A(a.l.1),\\ &\leftarrow A(a.l.2),\\ &dom(a.l.1) \leftarrow,\\ &dom(a.l.2) \leftarrow,\\ &\leftarrow a.l.1 \simeq a.l.2\end{aligned}$$

This time, only two successors were generated, so only one inequality clause had to be added. Note that when one successor is generated, no inequality clauses are added. In general, n pairwise distinct successors require $\binom{n}{2}$ inequality clauses.

5.3 The *At-Least* Extension Rule

Now that the *at-least* inference rule has been introduced, we will take a look at the *At-Least* extension rule, which encapsulates the *at-least* rule and works on actual branches in a tableau. This is analogous to the way *Split* and *Equality* encapsulate their respective inference rules, as described in section 4.5. The rule's definition is straight-forward:

$$\text{At-Least}\ \frac{\mathbf{B}}{\mathbf{B}\cdot E}\ \text{if}\ \begin{cases}\text{there is an at-least clause } C \in \mathbf{B} \text{ such that}\\ C \Rightarrow_{\text{at-least}} E \text{ is not redundant}\end{cases}$$

Unlike *Equality* and *Split*, *At-Least* does not append a single node but an entire sequence of nodes to $\mathbf{B}$.

An *at-least* inference $C \Rightarrow_{\text{at-least}} E$ is redundant w.r.t. a clause set S iff all clauses in E follow from S_C or iff the individual in C is *blocked* in S. (Blocking will be covered in the next section.) Note that the

clauses in E can only follow from S_C if *at-least* has already been applied to C because the individuals in E cannot have been introduced to S in any other way (remember that their names are based on the unique label of the number restriction in C). Also, for this check to work correctly, all $\geq$-number restrictions must be bigger in $\prec$ than the biggest term that will be generated by applying *at-least* to them. Otherwise, S_C would not contain all clauses in E, in which case some clauses in E would not follow from S_C. Hence, the inference would not be redundant and could be applied again.

5.4 Pairwise Anywhere Blocking

In order to obtain a terminating calculus, it is sometimes necessary to *block* an application of the *at-least* rule, i.e., to prevent it from generating successors for a given individual. Consider the $\mathcal{SHIQ}$ KB $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with

$$\begin{aligned}\mathcal{R} &= \emptyset,\\ \mathcal{T} &= \{A \sqsubseteq \geq 1\ R.A\},\\ \mathcal{A} &= \{A(a)\}\end{aligned}$$

Clearly, $\mathcal{K}$ is satisfiable by setting $\Delta^I = \{a\}$, $a^I = a$, $A^I = \{a\}$, and $R^I = \{(a, a)\}$. However, translating $\mathcal{K}$ into DLE-clauses would yield the following clause set:

$$\begin{aligned}\{&\top(\iota) \leftarrow,\\ &dom(\iota) \leftarrow,\\ &\geq 1\ R.A_l(x) \leftarrow A(x),\\ &A(a) \leftarrow,\\ &dom(a) \leftarrow\}\end{aligned}$$

(Note that the $\geq$-number restriction in the first clause was labeled with l.) Without blocking, the DLE-Hyper Tableau calculus would not terminate when applied to the above set because *at-least* would simply keep on generating successors, as shown in Figure 5.1.

As a remedy to this problem, we use a slightly modified version of the *pairwise anywhere blocking* technique defined in [13]. Before presenting it, however, a couple of definitions are required.

Definition 5.4.1 (**Named and Unnamed individuals).** Given a branch $\mathbf{B}$ in a DLE-Hyper Tableau derivation from $DLE(\mathcal{K})$, those individuals in $\mathbf{B}$ that already appeared in $DLE(\mathcal{K})$ are called *named individuals*. In contrast, the individuals in $\mathbf{B}$ that were introduced by *at-least* are called *unnamed individuals*. $N_{\mathbf{B}}$ denotes the set of all (named and unnamed) individuals in $\mathbf{B}$.

Note that *at-least* adds new individuals to the set $N_{\mathbf{B}}$. Next, we formally define the notions of *successor* and *predecessor*.

Definition 5.4.2 (**Successors and Predecessors).** An individual b is called a *successor* of a, with a being the *predecessor* of b, if and only if b was generated by applying the *at-least* rule to an at-least clause containing a. The relations *descendant* and *ancestor* are the transitive closures of the *successor* and *predecessor* relations, respectively.

Every unnamed individual has exactly one predecessor, whereas named individuals do not have any predecessor since they are not generated by *at-least*. Note that even if two named individuals a and b occur in a role clause of the form $(R(a, b) \leftarrow)$, b is *not* considered a successor of a. An individual (named or unnamed) may have multiple successors. As mentioned in chapter 4, the E-Hyper Tableau calculus uses a reduction ordering $\prec$ that is total on ground terms. Apart from $\prec$ being a reduction ordering, the

DLE-Hyper Tableau calculus also requires that for all individuals $a, a' \in N_{\mathbf{B}}$ in a branch $\mathbf{B}$, $a' \prec a$ if a' is the predecessor of a. (As a result, $a' \prec a$ also holds if a' is an ancestor of a.) Furthermore, all named individuals must be smaller in $\prec$ than any unnamed individual.

Definition 5.4.3 (**Equality Sequence).** An *equality sequence* is a sequence of equality clauses of the form $((a_1 \simeq a_2 \leftarrow), (a_2 \simeq a_3 \leftarrow), \ldots, (a_{m-2} \simeq a_{m-1} \leftarrow), (a_{m-1} \simeq a_m \leftarrow))$, for $a_1, \ldots, a_m$ individuals and $m \geq 2$.

In the above case, we say that the sequence *connects* a_1 to a_m and that the equality clauses in the sequence are its *links*. Furthermore, we say that a DLE-clause set S *contains* an equality sequence ES if ES consists entirely of (symmetric variants of) equality clauses in S. Note that if S contains an equality sequence ES connecting a to b, it also contains a sequence connecting b to a since one could simply reverse the order of the links in ES and replace them by their symmetric variants to obtain that sequence. Also, if one equality sequence in S connects a to b and another connects a to c, for a, b and c pairwise distinct individuals, then there obviously exists a sequence connecting a to c. This is important in the following definition.

Given a DLE-Hyper Tableau branch $\mathbf{B}$, with $N_{\mathbf{B}}$ denoting the set of individuals in $\mathbf{B}$, we define an equivalence relation $EQ_{\mathbf{B}}$ on $N_{\mathbf{B}}$ as follows:

$$EQ_{\mathbf{B}} = \{(a, b) \in N_{\mathbf{B}} \times N_{\mathbf{B}} \mid \ a = b \text{ or } \mathbf{B} \text{ contains an equality sequence connecting } a \text{ to } b\}$$

Note that, due to the properties of equality sequences mentioned above, $EQ_{\mathbf{B}}$ is indeed a reflexive, transitive relation.

Definition 5.4.4 (**Equivalence Class).** The *equivalence class* $[a]_{\mathbf{B}}$ of an individual a in a branch $\mathbf{B}$ is the set

$$[a]_{\mathbf{B}} = \{b \mid (a, b) \in EQ_{\mathbf{B}}\}$$

When it is clear which branch is being referred to, the subscript will usually be omitted and $[a]$ will be used to denote the equivalence class of the individual a in the branch in question. For example, when the statement of a Lemma refers to a branch $\mathbf{B}$ and no other branch is mentioned, $[a]$ stands for $[a]_{\mathbf{B}}$.

For an individual a and a set of individuals N, we write $a \preceq N$ if $a \preceq b$ for all $b \in N$. The *minimum* of N is then defined as $min\ N = a$ if $a \in N$ and $a \preceq N$. Note that the statement $a = min[a]_{\mathbf{B}}$ for an individual $a \in N_{\mathbf{B}}$ is equivalent to $a \preceq [a]_{\mathbf{B}}$. In this case, we say that a is the minimum of its equivalence class or that it is *minimal* in its class.

Definition 5.4.5 (**Blocking-relevant Concepts [13]).** A *blocking-relevant concept* is a concept of the form A, $\geq n\ R.A$, or $\geq n\ R.\neg A$ for A an atomic concept and R an atomic or inverse role.

Now we can finally introduce the notion of *pairwise anywhere blocking*. While the labeling method presented in the following definition is slightly different from the one used by Motik et al. [13], the rest of the definition was essentially copied from [13].

Definition 5.4.6 (**Pairwise Anywhere Blocking).** The label of an individual a and a pair of individuals (a, b) in a branch $\mathbf{B}$ is computed as follows:

$$\mathcal{L}_{\mathbf{B}}(a) = \{C \mid (C(a') \leftarrow) \in \mathbf{B} \text{ for some } a' \in [a]_{\mathbf{B}}, \text{ where } C \text{ is a blocking-relevant concept}\}$$
$$\mathcal{L}_{\mathbf{B}}(a, b) = \{R \mid (R(a', b') \leftarrow) \in \mathbf{B} \text{ for some } a' \in [a]_{\mathbf{B}} \text{ and } b' \in [b]_{\mathbf{B}}, \text{ where } R \text{ is an atomic role}\}$$

By induction on the term-ordering $\prec$, each individual a in $\mathbf{B}$ is then assigned a status as follows:

- a is *directly blocked* by an individual b iff both a and b are unnamed, b is not blocked, $b \prec a$, $\mathcal{L}_{\mathbf{B}}(a) = \mathcal{L}_{\mathbf{B}}(b)$, $\mathcal{L}_{\mathbf{B}}(a_{pre}) = \mathcal{L}_{\mathbf{B}}(b_{pre})$, $\mathcal{L}_{\mathbf{B}}(a_{pre}, a) = \mathcal{L}_{\mathbf{B}}(b_{pre}, b)$, and $\mathcal{L}_{\mathbf{B}}(a, a_{pre}) = \mathcal{L}_{\mathbf{B}}(b, b_{pre})$, where a_{pre} and b_{pre} are the predecessors of a and b, respectively.

- a is *indirectly blocked* iff its predecessor is blocked

- a is *blocked* iff it is either directly or indirectly blocked

The intuition behind this definition is that the pair of individuals a_{pre}, a is basically equivalent to the pair b_{pre}, b. If no contradiction was discovered after generating the successors of a, then generating successors of b will not result in a contradiction being found either.

In the example in Figure 5.1, the equivalence class of each individual only contains the individual itself. As a result, the following labels would have been computed (the branch identifier was omitted here):

$$\begin{aligned}
\mathcal{L}(a) &= \{\geq 1\ R.A\} \\
\mathcal{L}(a.l.1) &= \{\geq 1\ R.A\} \\
\mathcal{L}(a.l.1.l.1) &= \{\geq 1\ R.A\} \\
\mathcal{L}(a, a.l.1) &= \{R\} \\
\mathcal{L}(a.l.1, a) &= \emptyset \\
\mathcal{L}(a.l.1, a.l.1.l.1) &= \{R\} \\
\mathcal{L}(a.l.1.l.1, a.l.1) &= \emptyset
\end{aligned}$$

Note that we have $\mathcal{L}(a) = \mathcal{L}(a.l.1)$, $\mathcal{L}(a.l.1) = \mathcal{L}(a.l.1.l.1)$ (i.e., the predecessors of $a.l.1$ and $a.l.1.l.1$ have identical labels) as well as $\mathcal{L}(a, a.l.1) = \mathcal{L}(a.l.1, a.l.1.l.1)$ and $\mathcal{L}(a.l.1, a) = \mathcal{L}(a.l.1.l.1, a.l.1)$. As a result, $a.l.1.l.1$ would have been directly blocked by $a.l.1$, which means that *at-least* would not have been applied to $(\geq 1\ R.A_l(a.l.1.l.1) \leftarrow)$ and the calculus would have terminated after deriving that clause.

5.5 Redundancy and Exhausted Branches

The following is an extended version of Definition 4.3.1, which introduced the notion of *saturation up to redundancy* for regular E-Hyper Tableaux. Due to the new inference rule, a minor change was necessary to adapt it to DLE-Hyper Tableaux.

Definition 5.5.1 (**Saturation up to Redundancy).** A DLE-clause set S is *saturated up to redundancy* iff for all clauses $C \in S$, such that C is not redundant w.r.t. S, all of the following hold:

- Every inference $C \Rightarrow_{\text{split}(\pi)} A_1 \leftarrow, ..., A_m \leftarrow$, such that $C\pi$ is not redundant w.r.t. S, is redundant w.r.t. S.

- Every inference $C, D \Rightarrow_{R(\sigma)} E$, where $R \in \{$ sup-left, sup-unit-right $\}$ and D is a fresh variant of a positive unit clause from S, such that neither $C\sigma$ nor $D\sigma$ is redundant w.r.t. S, is redundant w.r.t. S.

- Every inference $C \Rightarrow_{\text{ref}(\sigma)} E$, such that $C\sigma$ is not redundant w.r.t. S, is redundant w.r.t. S.

- Every inference $C \Rightarrow_{\text{at-least}} E$, such that C is not redundant w.r.t. S, is redundant w.r.t. S.

The use of blocking leads to the following (theoretical) problem: given a clause set S that is saturated up to redundancy, suppose that the conclusion of the inference $C, D \Rightarrow_{\text{unit-sup-right}(\sigma)} E$, where E is a unit clause not containing a $\geq$-number restriction, follows from $S_C \cup \{D\}$, which means that the inference is redundant. Furthermore, suppose that $S_C \cup \{D\}$ contains an at-least clause $F = (\geq n\ R.B(a) \leftarrow)$ that was not expanded by *at-least* because a is blocked. We will assume that F actually contributes to rendering E redundant, i.e., $S_C \cup \{D\} \models_E E$ but $S_C \cup \{D\} \backslash \{F\} \not\models_E E$. (Note that, since the number of constants in the signature Σ is infinite, checking whether $S_C \cup \{D\} \models_E E$ holds is undecidable because every subset of n constants may be used as the set of a's successors.) This means that E is not added to S because it follows from the fact that a has n R-successors that satisfy B; however, this is not indicated by the corresponding role, concept and inequality clauses that would have been added by *at-least* because a is blocked. As a result, no unit clause in S indicates that E holds, which is why it is impossible to construct a model of S from the unit clauses contained in it.

As a result, the calculus needs to treat $\geq$-number restrictions as regular predicates without special semantics when performing redundancy checks. As mentioned above, this restriction is insignificant in practice because the redundancy checks it affects are undecidable anyway. We will refer to this simplified treatment of $\geq$-number restrictions as *naive semantics*. For example, $(\geq n\ R.C_l(a) \leftarrow)$ and $(a \simeq b \leftarrow)$ E-entail $(\geq n\ R.C_l(b) \leftarrow)$ under naive semantics, but the former at-least clause does not imply that a has n distinct R-successors because it is interpreted as just another predicate with no special meaning. The fact that these successors exist will only be E-entailed by the clause set once *at-least* has generated the corresponding unit clauses. If a is blocked in the saturated set, it will never be E-entailed under naive semantics. It is easy to see that any clause or inference that is redundant under naive semantics is also redundant under proper semantics.

In order to fully understand the implications of this restriction, let S denote a clause set that is saturated up to redundancy and does not contain the empty clause. In particular, this implies that the conclusion of every *sup-left*, *unit-sup-right*, *ref*, or *split* inference with non-redundant premises is E-entailed by smaller clauses in S under naive semantics. Since the original E-Hyper Tableau calculus, which only has these four inference rules, is complete, this means that it is possible to construct a rewrite system I_S from S that is a model of S under naive semantics using the model construction technique from section 4.9. Note that I_S will be naive in the sense that at-least clauses of the form $(\geq n\ R.C(a) \leftarrow)$ will simply be interpreted as true, as would a unit clause containing a regular predicate such as $(P(a) \leftarrow)$. If *at-least* was applied to the clause in question, then this would be correct in so far as the unit clauses generated by *at-least* would ensure that a actually has n distinct R-successors that satisfy C in I_S. However, if a is blocked, then I_S is not a proper model of S because the required successors of a do not exist in I_S. Hence, what remains to be shown in order to prove that the DLE-Hyper Tableau calculus is complete is that a saturated clause set that has a naive model also a proper model. We may, however, assume that the calculus will discover any contradiction in a given clause set that would have been found by the (complete) E-Hyper Tableau calculus. After all, the use of naive semantics when performing redundancy checks ensures that blocked at-least clauses cannot contribute to rendering applicable E-Hyper Tableau inferences redundant in DLE-Hyper Tableaux.

Since the DLE-Hyper Tableau calculus terminates (as will be shown in section 6.2), a DLE-Hyper Tableau branch can only consist of finitely many nodes. As a result, Definition 4.7.1 (Exhausted Branch) can simply be adapted as follows.

Definition 5.5.2 (Exhausted Branch). A DLE-Hyper Tableau branch **B** is exhausted iff it does not contain the empty clause and the set of its clauses is saturated up to redundancy.

It is obvious that Proposition 4.8.1 (Exhausted branches are saturated up to redundancy) carries over to DLE-Hyper Tableau branches.

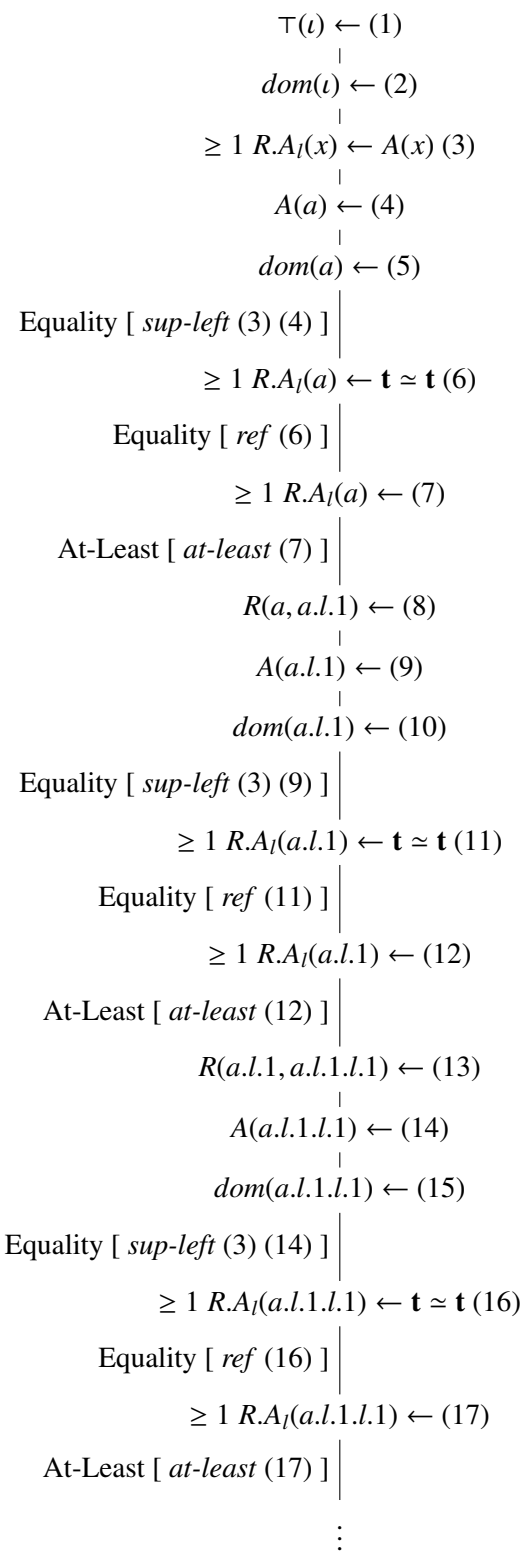

Figure 5.1: Example of how applying *at-least* without blocking leads to infinite derivations.

Chapter 6

Proofs

Now that the DLE-Hyper Tableau calculus has been introduced, we will show that it is a decision procedure for $\mathcal{SHIQ}$ by proving that the calculus is sound, terminating and complete.

6.1 Soundness

Theorem 6.1.1 (Soundness). The DLE-Hyper Tableau is sound.

Proof. The E-Hyper Tableau calculus was already shown to be sound in [7]. Since the DLE-Hyper Tableau calculus is identical to the E-Hyper Tableau calculus except for the newly introduced *at-least* inference rule and the corresponding *At-Least* extension rule, it is sufficient to show that the *at-least* rule is sound in order to prove that the entire DLE-Hyper Tableau calculus is sound.

Suppose there is a branch $\mathbf{B}$ and a DLE-interpretation I such that $I \models \mathbf{B}$. Let $\mathbf{B}'$ denote the branch obtained by applying *At-Least* to $\mathbf{B}$ with underlying inference $C \Rightarrow_{\text{at-least}} E$, where C is of the form $(\geq n\ R.B_l(a) \leftarrow)$, for R a role, B a literal concept and a an individual. This means that E consists of the clauses $(ar(R, a, a.l.i) \leftarrow)$, $cc(B, a.l.i)$ and $(dom(a.l.i) \leftarrow)$ for all $1 \leq i \leq n$ as well as $(\leftarrow a.l.i \simeq a.l.j)$ for all $1 \leq i < j \leq n$, where $a.l.1, \ldots, a.l.n$ are fresh individuals that do not occur in $\mathbf{B}$. $I \models \mathbf{B}$ implies that there exist elements $t_1, \ldots, t_n \in \Delta^I$ such that $(a^I, t_i) \in R^I$ and $t_i \in B^I$ for all $1 \leq i \leq n$ as well as $t_i \neq t_j$ for all $1 \leq i < j \leq n$. Now let I' be the interpretation obtained from I by setting $a.l.i^{I'} = t_i$ for all $1 \leq i \leq n$. *dom* is always interpreted as the universal concept, so those clauses are trivially true in both I and I'. Clearly, I' is a model of $\mathbf{B}' = \mathbf{B} \cdot E$. Hence, the *at-least* rule preserves satisfiability, which implies that the DLE-Hyper Tableau calculus is sound. □

6.2 Termination

Theorem 6.2.1 (Termination). The DLE-Hyper Tableau calculus terminates.

Proof. (Note that due to the lack of function symbols of non-zero arity in DLE-clauses, the DLE-Hyper Tableau calculus does not need to deal with inherently problematic clauses such as $P(f(x)) \leftarrow P(x)$)).

As shown in [7], adding the conclusion of a *Split* or *Equality* inference to a branch renders that inference redundant. The same holds true for *At-Least*, as described in section 5.3. *Simp* replaces a clause C in a branch $\mathbf{B}$ by a clause D such that C is redundant w.r.t. $\mathbf{B} \backslash \{C\} \cup \{D\}$. However, if C contributed to rendering an inference redundant, that inference will still be redundant after applying *Simp* because D is even smaller than C, so any inference that was redundant w.r.t. $\mathbf{B}$ will also be redundant w.r.t. $\mathbf{B} \backslash \{C\} \cup \{D\}$. Consequently, the calculus cannot get stuck in an infinite loop where the same inferences are applied time and again after being rendered non-redundant by *Simp* inferences.

Also, the *Simp* rule can only be applied a finite number of times to a finite branch because the clause ordering $\prec$ is well-founded by definition.

For the same reasons, applications of the *Del* rule that replace redundant clauses do not render redundant inferences non-redundant, but we still need to show that the same holds true for *Del* applications that remove non-properly subsumed clauses. Given a branch $\mathbf{B}$, a clause C is redundant w.r.t. $\mathbf{B}$ if each ground instance $C\gamma$ of C follows from the set $\mathbf{B}_{C\gamma}$ of those ground instances of clauses in $\mathbf{B}$ that are smaller than $C\gamma$. If a clause $D \in \mathbf{B}$ non-properly subsumes a clause $E \in \mathbf{B}$ (i.e., $D\sigma = E$ for some substitution σ), then the set of E's ground instances obviously is a subset of the set of D's ground instances. As a consequence, we have $\mathbf{B}_{C\gamma} = (\mathbf{B}\backslash\{E\})_{C\gamma}$, so any inference that was redundant w.r.t. $\mathbf{B}$ will still be redundant after E was removed from $\mathbf{B}$ by *Del*.

What remains to be shown is that the generation of new individuals and clauses by *at-least* does not lead to infinite derivations. *at-least* can only be applied once to any ground *at-least* clause; afterwards, the same inference will be redundant. Furthermore, it can only be applied to individuals that are not blocked. As the number of roles and concepts in a DLE-clause set is finite, there is only a finite number of tuples $(\mathcal{L}_{\mathbf{B}}(a_{pre}), \mathcal{L}_{\mathbf{B}}(a), \mathcal{L}_{\mathbf{B}}(a_{pre}, a), \mathcal{L}_{\mathbf{B}}(a, a_{pre}))$ for an unnamed individual a and its predecessor a_{pre} in a branch $\mathbf{B}$. Hence, after a finite number of derivation steps, for any newly generated individual b and its predecessor b_{pre} there will be an unnamed, non-blocked individual a with predecessor a_{pre} such that $\mathcal{L}_{\mathbf{B}}(a) = \mathcal{L}_{\mathbf{B}}(b)$, $\mathcal{L}_{\mathbf{B}}(a_{pre}) = \mathcal{L}_{\mathbf{B}}(b_{pre})$, $\mathcal{L}_{\mathbf{B}}(a_{pre}, a) = \mathcal{L}_{\mathbf{B}}(b_{pre}, b)$, and $\mathcal{L}_{\mathbf{B}}(a, a_{pre}) = \mathcal{L}_{\mathbf{B}}(b, b_{pre})$. This means that b will be directly blocked by a, i.e., if b occurs in an *at-least* clause, the *at-least* rule will not generate successors for b.

Hence, after a finite number of applications of derivation rules, no derivation rule can be applied anymore, so the DLE-Hyper Tableau calculus terminates. □

6.3 Completeness

6.3.1 Preliminaries

Definition 6.3.1 (Ground Variant). Given a ground clause C containing individuals $a_1, ..., a_m$, the clause obtained by replacing all occurrences of a_i with an individual $a'_i \in [a_i]$ (for all $1 \leq i \leq m$) is a *ground variant* of C. The *minimal ground variant* of C is obtained by replacing all individuals in C by the minimums of their respective equivalence classes.

We will sometimes refer to the *smallest ground variant* D of some ground clause C in a branch $\mathbf{B}$. This simply means that $\mathbf{B}$ does not contain a ground variant of C that is smaller in $\prec$ than D. However, D is not necessarily the minimal ground variant. For example, if $\mathbf{B}$ only contained the clauses $(A(b) \leftarrow)$ and $(a \simeq b \leftarrow)$, with $a \prec b$, then $(A(b) \leftarrow)$ would be the smallest ground variant of itself in $\mathbf{B}$ whereas $(A(a) \leftarrow)$, which has not been derived yet, would be the minimal ground variant. (With that said, we will later see that the smallest ground variant of a clause in an *exhausted branch* is also the minimal ground variant.)

Next, the notions of *successor* and *predecessor* are lifted to equivalence classes.

Definition 6.3.2 (Successors and Predecessors [Equivalence Classes]). $[b]$ is a *successor* of $[a]$ if every individual in $[b]$ is a descendant of an individual in $[a]$ and $[a]$ contains the predecessor of $min[b]$. $[a]$ is the *predecessor* of $[b]$ if $[b]$ is a successor of $[a]$. As with individuals, *descendant* and *ancestor* denote the transitive closures of *successor* and *predecessor*, respectively.

Note that if both $[a]$ and $[a']$ are predecessors of $[b]$, they must both contain the predecessor of $min[b]$, which implies $[a] = [a']$. Also, recall that an unnamed individual is always bigger in $\prec$ than its predecessor. This means that no equivalence class can be a successor of itself, as it would have to contain the predecessor of its own minimum, which must be even smaller than the minimum itself. Finally, since

named individuals have no predecessor, an equivalence class $[a]$ cannot be a successor of any class if $min[a]$ is a named individual.

Definition 6.3.3 (Paths [13]). A *path* is a finite sequence of pairs of individuals $p = [\frac{a_0}{a'_0}, ..., \frac{a_n}{a'_n}]$, with $tail(p) = a_n$ and $tail'(p) = a'_n$. $q = [p \mid \frac{a_{n+1}}{a'_{n+1}}]$ denotes the path obtained by appending $\frac{a_{n+1}}{a'_{n+1}}$ to p.

6.3.2 Model Construction

Definition 6.3.4 (Paths in a Branch). Given a DLE-Hyper Tableau branch $\mathbf{B}$, the *set of paths in* $\mathbf{B}$, $P_\mathbf{B}$, is defined as follows, where a and a' denote individuals in $N_\mathbf{B}$:

1. $[\frac{a}{a}] \in P_\mathbf{B}$ if a is named and $a \preceq [a]_\mathbf{B}$
2. $[p \mid \frac{a'}{a'}] \in P_\mathbf{B}$ if $p \in P_\mathbf{B}$, $a' \preceq [a']_\mathbf{B}$, a' is not blocked, and $[a']_\mathbf{B}$ is a successor of $[tail(p)]_\mathbf{B}$
3. $[p \mid \frac{a}{a'}] \in P_\mathbf{B}$ if $p \in P_\mathbf{B}$, $a' \preceq [a']_\mathbf{B}$, $a \preceq [a]_\mathbf{B}$, a' is directly blocked by a, and $[a']_\mathbf{B}$ is a successor of $[tail(p)]_\mathbf{B}$

Definition 6.3.5 (Induced Interpretation). Given an exhausted DLE-Hyper Tableau branch $\mathbf{B}$, the interpretation I that is *induced* by $\mathbf{B}$ is defined as follows:

$$
\begin{aligned}
\Delta^I &= P_\mathbf{B} \\
a^I &= \begin{cases} [\frac{a}{a}] & \text{if } a \text{ is a named individual with } a \preceq [a]_\mathbf{B} \\ (min[a]_\mathbf{B})^I & \text{if } a \text{ is a named individual with } a \not\preceq [a]_\mathbf{B} \end{cases} \\
A^I &= \{p \in \Delta^I \mid (A(tail(p)) \leftarrow) \in \mathbf{B}\} \\
R^I &= \{(a^I, b^I) \mid (R(a,b) \leftarrow) \in \mathbf{B} \text{ and } a \text{ and } b \text{ are both named}\} \\
&\quad \cup \{(p, [p \mid \tfrac{a}{a'}]) \in \Delta^I \times \Delta^I \mid (R(tail(p), a') \leftarrow) \in \mathbf{B}\} \\
&\quad \cup \{([p \mid \tfrac{a}{a'}], p) \in \Delta^I \times \Delta^I \mid (R(a', tail(p)) \leftarrow) \in \mathbf{B}\}
\end{aligned}
$$

a, a' and b are individuals, with A and R denoting atomic concepts and roles, respectively. Note that a^I is defined by induction on the term ordering $\prec$, starting with the smallest (named) individual.

Definition 6.3.5 introduces the model construction method that will be used to prove that the DLE-Hyper Tableau calculus is complete. In order to get a better understanding of the kinds of interpretations constructed this way, consider the following two examples. Note that, for both examples, we will assume that A and C are atomic concepts and R is an atomic role.

First, assume we have a normalized $\mathcal{ALCHIQ}$ knowledge base $\mathcal{K}$ consisting of the ABox $\mathcal{A} = \{A(a), a \simeq b\}$, an empty RBox and the TBox $\mathcal{T} = \{A \sqsubseteq \geq 1\ R.C\}$. The following set of DLE-clauses will be constructed from $\mathcal{K}$ and added to the initial branch of the tableau:

$$
\begin{aligned}
DLE(\mathcal{K}) = \{&\top(\iota) \leftarrow, && (1)\\
&dom(\iota) \leftarrow, && (2)\\
&dom(a) \leftarrow, && (3)\\
&dom(b) \leftarrow, && (4)\\
&A(a) \leftarrow, && (5)\\
&a \simeq b \leftarrow, && (6)\\
&\geq 1\ R.C_l(x) \leftarrow A(x)\} && (7)
\end{aligned}
$$

Assuming that a is smaller than b in the term ordering, the exhausted branch $\mathbf{B}$ will contain the above clauses as well as

$$\{ \geq 1\ R.C_l(a) \leftarrow \mathbf{t} \simeq \mathbf{t}, \tag{8}$$
$$\geq 1\ R.C_l(a) \leftarrow, \tag{9}$$
$$R(a, a.l.1) \leftarrow, \tag{10}$$
$$C(a.l.1) \leftarrow \tag{11}$$
$$dom(a.l.1) \leftarrow\} \tag{12}$$

Clause (8) was derived by *sup-left* with premises (7) and (5), *ref* then derived (9) from (8), and the clauses (10)-(12) were generated by applying *at-least* to (9). Since both ι and a are named and minimal in their respective equivalence classes (note that $[a]$ also contains b) and because $[a.l.1]$ is a successor of $[a]$, the set of all paths in $\mathbf{B}$ is

$$P_{\mathbf{B}} = \{[\tfrac{\iota}{\iota}], [\tfrac{a}{a}], [\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}]\}$$

The interpretations of the named individuals are defined as follows:

$$\iota^I = [\tfrac{\iota}{\iota}]$$
$$a^I = [\tfrac{a}{a}]$$
$$b^I = [\tfrac{a}{a}]$$

b^I is also set to $[\frac{a}{a}]$ since that is the interpretation of the minimal element in $[b]$, namely a. As for the concept interpretations, we have

$$A^I = \{[\tfrac{a}{a}]\}$$
$$C^I = \{[\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}]\}$$
$$dom^I = P_{\mathbf{B}}$$
$$\top^I = \Delta^I = P_{\mathbf{B}}$$

The elements in A^I and C^I were added because of clauses (5) and (11), respectively. Furthermore, $dom^I = P_{\mathbf{B}}$ because $(dom(tail(p)) \leftarrow) \in \mathbf{B}$ for every path $p \in P_{\mathbf{B}}$. $\top^I = \Delta^I = P_{\mathbf{B}}$ is implied by the definition of $\top$ as the universal concept.

Due to clause (10) and the second part of the definition of role interpretations, R will be interpreted as

$$R^I = \{([\tfrac{a}{a}], [\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}])\}$$

Clauses (1)-(4) are true in I because dom^I and $\top^I$ contain the interpretations of the respective named individuals. $a^I \in A^I$ makes (5) true in I, and $a^I = b^I$ implies that (6) also holds. Finally, due to $([\frac{a}{a}], [\frac{a}{a}, \frac{a.l.1}{a.l.1}]) \in R^I$ and $[\frac{a}{a}, \frac{a.1.1}{a.1.1}] \in C^I$, we have $(\geq 1\ R.C)^I = \{[\frac{a}{a}]\}$. This implies $A^I \subseteq (\geq 1\ R.C)^I$ holds, which means that I satisfies clause (7). Hence, I is a model of $DLE(\mathcal{K})$.

Next, consider again the DLE-clause set from section 5.4, which served to illustrate why blocking is necessary:

$$\{\top(\iota) \leftarrow, \tag{1}$$
$$dom(\iota) \leftarrow, \tag{2}$$
$$\geq 1\ R.A_l(x) \leftarrow A(x), \tag{3}$$
$$A(a) \leftarrow, \tag{4}$$
$$dom(a) \leftarrow\} \tag{5}$$

The corresponding exhausted branch $\mathbf{B}$ will contain the following additional clauses:

$$\{ \geq 1\ R.A_l(a) \leftarrow \mathbf{t} \simeq \mathbf{t}, \tag{6}$$
$$\geq 1\ R.A_l(a) \leftarrow, \tag{7}$$
$$R(a, a.l.1) \leftarrow, \tag{8}$$
$$A(a.l.1) \leftarrow, \tag{9}$$
$$dom(a.l.1) \leftarrow, \tag{10}$$
$$\geq 1\ R.A_1(a.l.1) \leftarrow \mathbf{t} \simeq \mathbf{t}, \tag{11}$$
$$\geq 1\ R.A_1(a.l.1) \leftarrow, \tag{12}$$
$$R(a.l.1, a.l.1.l.1) \leftarrow, \tag{13}$$
$$A(a.l.1.l.1) \leftarrow, \tag{14}$$
$$dom(a.l.1.l.1) \leftarrow, \tag{15}$$
$$\geq 1\ R.A_l(a.l.1.l.1) \leftarrow \mathbf{t} \simeq \mathbf{t}, \tag{16}$$
$$\geq 1\ R.A_l(a.l.1.l.1) \leftarrow\} \tag{17}$$

The only difference compared to Figure 5.1 is that, this time, pairwise anywhere blocking was used and the derivation terminated after clause (17) was derived.

As in the previous example, the paths $[\frac{\iota}{\iota}]$ and $[\frac{a}{a}]$ will be added to $P_{\mathbf{B}}$. Next, $[\frac{a}{a}, \frac{a.l.1}{a.l.1}]$ will be added because $[a.l.1]$ is a successor of $[a]$. Now consider the individual $a.l.1.l.1$. As shown in section 5.4, it is directly blocked by $a.l.1$. Since $[a.l.1.l.1]$ is also a successor of $[a.l.1]$, part three of the definition of paths applies and the path $[\frac{a}{a}, \frac{a.l.1}{a.l.1}, \frac{a.l.1}{a.l.1.l.1}]$ will be added to $P_{\mathbf{B}}$. The next paths to be added will then be $[\frac{a}{a}, \frac{a.l.1}{a.l.1}, \frac{a.l.1}{a.l.1.l.1}, \frac{a.l.1}{a.l.1.l.1}]$, $[\frac{a}{a}, \frac{a.l.1}{a.l.1}, \frac{a.l.1}{a.l.1.l.1}, \frac{a.l.1}{a.l.1.l.1}, \frac{a.l.1}{a.l.1.l.1}]$ and so on. I.e., the paths will grow indefinitely and the set of paths will be infinite:

$$\begin{aligned} P_{\mathbf{B}} = \{&[\tfrac{\iota}{\iota}], \\ &[\tfrac{a}{a}], \\ &[\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}], \\ &[\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}, \tfrac{a.l.1}{a.l.1.l.1}], \\ &[\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}, \tfrac{a.l.1}{a.l.1.l.1}, \tfrac{a.l.1}{a.l.1.l.1}], \\ &\dots\} \end{aligned}$$

Remember that a path p will be added to the interpretation of an atomic concept B if the branch contains $(B(tail(p)) \leftarrow)$. In this case, we have $(dom(tail(p)) \leftarrow) \in \mathbf{B}$ for all $p \in P_{\mathbf{B}}$, which implies $dom^I = P_{\mathbf{B}}$. As for A, $(A(tail(p)) \leftarrow) \in \mathbf{B}$ holds for all $p \in P_{\mathbf{B}}$ except ι^I, which implies $A^I = P_{\mathbf{B}} \backslash \{[\frac{\iota}{\iota}]\}$.

A tuple of paths $(p, [p \mid \frac{b}{b'}])$ will be added to R^I (the interpretation of the atomic role R) if $(R(tail(p), b') \leftarrow)$ is contained in the branch. In our example, this yields the infinite set

$$\begin{aligned} R^I = \{&([\tfrac{a}{a}], [\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}]), \\ &([\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}], [\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}, \tfrac{a.l.1}{a.l.1.l.1}]), \\ &([\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}, \tfrac{a.l.1}{a.l.1.l.1}], [\tfrac{a}{a}, \tfrac{a.l.1}{a.l.1}, \tfrac{a.l.1}{a.l.1.l.1}, \tfrac{a.l.1}{a.l.1.l.1}, \tfrac{a.l.1}{a.l.1.l.1}]), \\ &\dots\} \end{aligned}$$

Now consider the clauses in $DLE(\mathcal{K})$. $(\top(\iota) \leftarrow)$ is trivially true in I because, as always, $\top^I = \Delta^I = P_{\mathbf{B}}$. Similarly, the two *dom* concept clauses both hold in I due to $dom^I = P_{\mathbf{B}}$. $a^I \in A^I$ implies that I satisfies $(A(a) \leftarrow)$ as well. As for $(\geq 1\ R.A_1(x) \leftarrow A(x))$, this clause is true in I because for every $p \in A^I$ there exists a path $p' \in P_{\mathbf{B}}$ such that $(p, p') \in R^I$ and $p' \in A^I$, as implied by the definitions of A^I and R^I in this example. Thus, I is a model of $DLE(\mathcal{K})$.

6.3.3 Proof

From now on, we will assume that $\mathbf{B}^n$ is an exhausted branch in a tableau $\mathbf{T}_n$, which is the final tableau in the fair DLE-Hyper Tableau derivation $\mathbf{D} = (\mathbf{T}_0, \ldots, \mathbf{T}_n)$. Also, we will assume that $\mathbf{D}$ is a derivation from $DLE(\Delta(\Omega(\mathcal{K})))$, the set of DLE-clauses constructed from the $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$. $\mathbf{B}^i$ (for all $0 \leq i < n$) denotes the branch in $\mathbf{T}_i$ from which $\mathbf{B}^{i+1}$ in $\mathbf{T}_{i+1}$ was derived. If $\mathbf{T}_{i+1}$ was obtained from $\mathbf{T}_i$ by applying an extension or simplification rule to $\mathbf{B}^i$, then $\mathbf{B}^{i+1}$ denotes the resulting branch (or one of the resulting branches, if *Split* was applied). However, if an extension or simplification rule was applied to a different branch in $\mathbf{T}_i$, then $\mathbf{B}_i$ and $\mathbf{B}^{i+1}$ are identical.

Remember that the fact that $\mathbf{B}^n$ is exhausted implies that the clause set $\mathbf{B}^n$ (as always, we also interpret a branch as the set of its clauses) is saturated up to redundancy and does not contain the empty clause $\square$. In order to show that the DLE-Hyper Tableau calculus is complete, we will prove that the interpretation I that is induced by $\mathbf{B}^n$ is a model of $DLE(\Delta(\Omega(\mathcal{K})))$. The satisfiability of the original $\mathcal{SHIQ}$ KB $\mathcal{K}$ then follows from Theorem 3.5.1, which states $DLE(\Delta(\Omega(\mathcal{K})))$ is satisfiable if and only if $\mathcal{K}$ is satisfiable.

One final word before the actual proof begins: for the sake of simplicity, but without loss of generality, we will assume that the special constant $\mathbf{t}$ is the smallest term in the term ordering $\prec$.

Lemma 6.3.1. All clauses in a DLE-Hyper Tableau branch that have empty bodies, or bodies containing only trivial $\mathbf{t} \simeq \mathbf{t}$ literals, are ground.

Proof. All non-ground clauses in $DLE(\mathcal{K})$ are range-restricted as described in section 3.5. As a result, all positive unit clauses in $DLE(\mathcal{K})$ are ground. In order to remove non-trivial body literals from non-ground clauses using *sup-left*, the body literals must be unified with the head literal of a positive unit clause. By successively unifying all body literals in a non-ground, range-restricted DLE-clause C with head literals of ground positive unit clauses (and thus instantiating all body variables), all of C's head literals will have been grounded by the time the body is empty or contains only trivial $\mathbf{t} \simeq \mathbf{t}$ literals that were left by *sup-left*. $\square$

Lemma 6.3.2. If a clause C is redundant w.r.t. $\mathbf{B}^i$ for some $1 \leq i \leq n$, then C is redundant w.r.t. $\mathbf{B}^n$.

Proof. This Lemma is equivalent to Lemma A.8 in [7], which formally proves the above property for E-Hyper Tableaux. The proof easily carries over to DLE-Hyper Tableaux, though. The basic idea is the following: let $\mathbf{B}$ denote a branch w.r.t which C is redundant, and let $\mathbf{B}'$ denote the branch obtained from $\mathbf{B}$ by applying some derivation rule. If *Equality*, *Split* or *At-Least* were applied, then by monotonicity of first-order logic with equality, C must also be redundant w.r.t. $\mathbf{B}'$ because we have $\mathbf{B} \subseteq \mathbf{B}'$ (the clause set of $\mathbf{B}$ was extended). If *Del* or *Simp* were applied, then, by the definitions of the two rules, the clause that was replaced must still follow from smaller clauses in $\mathbf{B}'$ or it must be a ground instance of a clause in $\mathbf{B}'$. As a result, any clause that was redundant w.r.t. $\mathbf{B}$ will still be redundant w.r.t. $\mathbf{B}'$. Hence, by induction, the clause C in the Lemma's statement must be redundant w.r.t. $\mathbf{B}^n$. $\square$

As mentioned in section 5.5, it is possible to construct a naive model of an exhausted DLE-Hyper Tableau branch by generating a term rewrite system from the positive unit clauses in the branch using the model construction method presented in section 4.9. (Note, however, that we have yet to prove that any exhausted branch also has a proper model.) We will denote the rewrite system constructed from $\mathbf{B}^n$ by $TRS_{\mathbf{B}^n}$.

Remember that DLE-atoms can only take on three forms: $D(s) \simeq \mathbf{t}$, $R(s,t) \simeq \mathbf{t}$ and $s \simeq t$ for s and t individuals or variables, D a blocking-relevant concept and R an atomic role. Since all positive unit DLE-clauses are ground by Lemma 6.3.1, they must be of the form $(D(a) \simeq \mathbf{t} \leftarrow)$, $(R(a,b) \simeq \mathbf{t} \leftarrow)$ or $(a \simeq b \leftarrow)$ for a and b individuals. As a result, all rules in $TRS_{\mathbf{B}^n}$ must be of the form $D(a) \Rightarrow \mathbf{t}$, $R(a,b) \Rightarrow \mathbf{t}$ or $a \Rightarrow b$.

Recall that an equational clause C follows from $TRS_{\mathbf{B}^n}$ ($TRS_{\mathbf{B}^n} \models_E C$) iff both sides of the equation have the same normal form in $TRS_{\mathbf{B}^n}$.

Lemma 6.3.3. If $(a \simeq b \leftarrow)$ was redundant w.r.t. $\mathbf{B}^i$ for some $1 \leq i < n$ and a and b individuals, then $\mathbf{B}^n$ contains an equality sequence that connects a and b.

Proof. Lemma 6.3.2 implies that $(a \simeq b \leftarrow)$ must still be E-entailed by $\mathbf{B}^n$ under naive semantics. As a result, a and b must have the same normal form in $TRS_{\mathbf{B}^n}$. The only positive unit clauses that can be turned into rewrite rules that rewrite individuals are equality clauses. Hence, $\mathbf{B}^n$ must contain two equality sequences ES_a and ES_b, with ES_a connecting a to its normal form and ES_b connecting b to the same normal form as a. Note that said normal form must also be an individual. Since ES_a and ES_b indirectly connect a and b via their common normal form, the two sequences can easily be turned into a sequence that connects a and b (e.g., by reversing the order of the links in ES_b and swapping their left-hand and right-hand sides before appending ES_b to ES_a; with $ES_a = (a \simeq c \leftarrow)$ and $ES_b = (b \simeq c \leftarrow)$, where c is the normal form of a and b, we would thus obtain $(a \simeq c \leftarrow, c \simeq b \leftarrow)$). □

Lemma 6.3.4. If C was redundant w.r.t. $\mathbf{B}^i$ for some $1 \leq i < n$ and a concept clause or role clause C, then $\mathbf{B}^n$ contains a ground variant of C.

Proof. Lemma 6.3.2 implies that C must still follow from $\mathbf{B}^n$ under naive semantics, so the two sides of the equation must have the same normal form in $TRS_{\mathbf{B}^n}$. As $\mathbf{t}$ is the smallest term in $\prec$, it cannot be rewritten by $TRS_{\mathbf{B}^n}$. In other words, it already is in normal form. Hence, $TRS_{\mathbf{B}^n}$ can only rewrite the left-hand side of C, which must be of the form $D(a)$ or $R(a, b)$ (we will assume that C is a concept clause and focus on $D(a)$ since the other case is analogous). If $D(a)$ is rewritten directly into $\mathbf{t}$, this means that the corresponding rewrite rule was derived from $(D(a) \simeq \mathbf{t} \leftarrow) \in \mathbf{B}^n$, in which case the Lemma's statement would hold since each ground clause is a ground variant of itself. (Note that the rule cannot have been derived from a clause of the form $(D(x) \simeq \mathbf{t} \leftarrow)$ since all positive unit clauses are ground by Lemma 6.3.1.) The only other way to rewrite $D(a)$ is to rewrite the individual a, in which case the corresponding rewrite rules must have been derived from equality clauses in $\mathbf{B}^n$. For $D(a)$ to finally be rewritten into $\mathbf{t}$, however, $TRS_{\mathbf{B}^n}$ must contain a rule of the form $D(a') \Rightarrow \mathbf{t}$, with other rules rewriting a into a'. This implies that $\mathbf{B}^n$ contains the clause $(D(a') \leftarrow)$ as well as an equality sequence connecting a to a'. As a result, we have $[a] = [a']$, which means that $(D(a') \simeq \mathbf{t} \leftarrow)$ is in fact a ground variant of $(D(a) \simeq \mathbf{t} \leftarrow)$. □

Lemma 6.3.5. For every individual a in $\mathbf{B}^n$ with $a \neq min[a]$, there exists a non-redundant equality clause $(a \simeq a' \leftarrow) \in \mathbf{B}^n$ with $a' \prec a$.

Proof. Since a and $min[a]$ are connected by an equality sequence in $\mathbf{B}^n$, they must have the same normal form in $TRS_{\mathbf{B}^n}$. Furthermore, the rules that rewrite them can only have been derived from equality clauses in $\mathbf{B}^n$. By construction, the left-hand side of any rule in $TRS_{\mathbf{B}^n}$ is bigger than the right-hand side. If $min[a]$ could be rewritten, then it would have to appear on the left-hand side of some rule, with a smaller individual b constituting the right-hand side. However, that would imply that $\mathbf{B}^n$ contains an equality clauses with $min[a]$ on one side and the even smaller individual b on the other side. Hence, $min[a]$ would not be the minimum of the equivalence class $[a]$, a plain contradiction. As a result, $min[a]$ must be irreducible w.r.t. $TRS_{\mathbf{B}^n}$, which implies that it is the normal form of a. This means that $TRS_{\mathbf{B}^n}$ contains a rule of the form $a \Rightarrow c$ with $a \succ c$. The fact that the rule was added to $TRS_{\mathbf{B}^n}$ implies that the equality clause $(a \simeq c \leftarrow) \in \mathbf{B}^n$ from which $a \Rightarrow c$ was derived is not redundant, which completes the proof.

□

Lemma 6.3.6. For any concept clause $(D(a) \leftarrow) \in \mathbf{B}^n$ or role clause $(R(a, b) \leftarrow) \in \mathbf{B}^n$, $\mathbf{B}^n$ also contains $(D(min[a]) \leftarrow) \in \mathbf{B}^n$ or $(R(min[a], min[b]) \leftarrow)$, respectively.

Proof. Let C denote a concept or role clause in $\mathbf{B}^n$, with C_{min} denoting the smallest ground variant of C in $\mathbf{B}^n$. Then, C_{min} cannot be redundant because no smaller ground variant of it exists in $\mathbf{B}^n$. If every individual in C_{min} is minimal in its equivalence class, nothing remains to be shown. Otherwise, let a denote an individual in C_{min} that is not the minimum of its equivalence class. By Lemma 6.3.5, there exists a non-redundant equality clause $E = (a \simeq a' \leftarrow) \in \mathbf{B}^n$ with $a' \prec a$. It would then be possible to apply *unit-sup-right* with left premise C_{min} and right premise E, neither of which are redundant. The resulting ground variant of C_{min} would not be redundant because it would be even smaller than C_{min} and $\mathbf{B}^n$ contains no smaller ground variant of it. Since that would contradict the assumption that $\mathbf{B}^n$ is saturated up to redundancy, all individuals in C_{min} must already be the minimal elements of their respective equivalence classes. □

Lemma 6.3.7. If the individual a is labeled with the blocking-relevant concept D in $\mathbf{B}^n$ (i.e., $D \in \mathcal{L}_{\mathbf{B}^n}(a)$), then $\mathbf{B}^n$ contains $(D(min[a]) \leftarrow)$.

Proof. By the definition of blocking, $D \in \mathcal{L}_{\mathbf{B}^n}(a)$ implies that there exists an individual $a' \in [a]$ such that $(D(a') \leftarrow) \in \mathbf{B}^n$. This implies $(D(min[a]) \leftarrow) \in \mathbf{B}^n$ by Lemma 6.3.6. □

Lemma 6.3.8. If the pair of individuals (a, b) is labeled with the atomic role R in $\mathbf{B}^n$ (i.e., $R \in \mathcal{L}_{\mathbf{B}^n}(a, b)$), then $\mathbf{B}^n$ contains $(R(min[a], min[b]) \leftarrow)$.

Proof. By the definition of blocking, $R \in \mathcal{L}_{\mathbf{B}^n}(a, b)$ implies that there exist individuals $a' \in [a]$ and $b' \in [b]$ such that $(R(a', b') \leftarrow) \in \mathbf{B}^n$. This implies $(R(min[a], min[b]) \leftarrow) \in \mathbf{B}^n$ by Lemma 6.3.6. □

Lemma 6.3.9. Every role clause $(R(a, b) \leftarrow)$ in a DLE-Hyper Tableau branch $\mathbf{B}$ satisfies one of the following properties:

1. $min[a]$ and $min[b]$ are named
2. $[b]$ is a successor of $[a]$
3. $[a]$ is a successor of $[b]$

Proof. The proof is by induction on the applications of the inference rules. Before any inference rule is applied, a DLE-clause set contains only named individuals, so all role clauses must satisfy (1). In order to prove the Lemma's statement, we need to consider all inference rules that introduce new role clauses or equality clauses. Note that the latter may affect the equivalence class of an individual occurring in a role clause.

There are three ways new role clauses can be derived: (a) from an existing role clause, using *unit-sup-right*, or (b) from an at-least clause using *at-least*, or (c) from a non-unit clause whose head contains a role literal, using *ref* (we will later see that *split* need not be considered).

As for (a), while *unit-sup-right* may infer new role clauses if its left premise is a role clause and its right premise is an equality clause (e.g., $(R(b, c) \leftarrow), (b \simeq a \leftarrow) \Rightarrow_{\text{unit-sup-right}(\epsilon)} (R(a, c) \leftarrow)$), the conclusion will simply be a ground variant of the left premise. Since the left premise must satisfy one of the three role clause properties, the resulting ground variant must satisfy the same property. After all, the equivalence classes of the individuals in the conclusion are the same as the ones of the individuals in the left premise (e.g., in the above example $(b \simeq a \leftarrow)$ implies $[a] = [b]$).

Now consider case (b). The role clauses introduced by *at-least* are of the form $(ar(R, a, a') \leftarrow)$, where a' is a successor of a. Since a' is a fresh constant that is the only individual in its equivalence class right after being generated, $[a']$ is a successor of $[a]$ (every individual in $[a'] = \{a'\}$ is a descendant of some individual in $[a]$, and $min[a'] = a'$ is a direct successor of an individual in $[a]$). This means that the above role clause satisfies property (2) or (3), depending on whether R is atomic or inverse.

In case (c), a role clause is derived from a non-unit clause. As seen in Table 3.2, the only non-unit DL(E)-clauses whose heads contain a role literal are the ones derived from RBox axioms. The original clauses are of the form $C = (ar(R, x, y) \leftarrow ar(S, x, y))$ for R and S roles. In order to derive a role clause from C, *sup-left* must first unify the body literal with the head literal of a role clause such as $D = (ar(S, a, b) \leftarrow)$. The conclusion will be of the form $E_1 = (ar(R, a, b) \leftarrow \mathbf{t} \simeq \mathbf{t})$ if R and S are both atomic or inverse or $E_2 = (ar(R, b, a) \leftarrow \mathbf{t} \simeq \mathbf{t})$ if one of the roles is atomic while the other is inverse. In the case of E_1, the role clause that can be derived by removing the trivial body literal with the help of *ref* will simply inherit the role clause property of D. A role clause derived from E_2 will satisfy (1) if D satisfies (1), whereas it will satisfy (2) if D satisfies (3) and vice versa.

As shown in the above case analysis, adding role clauses to a branch preserves the validity of the Lemma's statement. However, we still need to show that the same holds true for adding equality clauses, which may affect the equivalence classes of individuals in role clauses.

New equality clauses can be derived in two ways: (a) from two existing equality clauses, using *unit-sup-right*, or (b) from a clause whose head contains a literal of the form $a \simeq b$, using *ref* (if said literal is the only head literal and the body only contains the trivial literal $\mathbf{t} \simeq \mathbf{t}$) or *split* (if there is more than one head literal and the body is empty).

Case (a) is easy to deal with. *unit-sup-right* may introduce new equality clauses if both premises are equality clauses (e.g., $(c \simeq a \leftarrow), (c \simeq b \leftarrow) \Rightarrow_{\text{unit-sup-right}(\epsilon)} (a \simeq b \leftarrow)$), but all this does, essentially, is directly 'connect' two individuals that were already connected by an equality sequence and were thus members of the same equivalence class. Hence, the equivalence classes are not affected at all by this kind of inference and the validity of the Lemma's statement is preserved.

Now consider case (b). Before any inference rules are applied, head literals of the form $y_1 \simeq y_2$, for y_1 and y_2 variables, can only appear in the head of a DL(E)-clause derived from a TBox axiom, as shown in Table 3.2. The clause's body must contain two literals of the form $ar(R, x, y_1)$ and $ar(R, x, y_2)$ in this case. This means that whenever an equality clause $(b \simeq c \leftarrow)$ is derived from such a DLE-clause, the two corresponding role literals in the body must have been unified with role clauses of the form $(ar(R, a, b) \leftarrow)$ and $(ar(R, a', c) \leftarrow)$, where a and a' are members of the same equivalence class. By considering the properties these role clauses may satisfy, we will show that after adding $(b \simeq c \leftarrow)$ to the branch, all role clauses containing individuals from $[b]$ or $[c]$ (the two equivalence classes that are merged by deriving $(b \simeq c \leftarrow)$) will still satisfy one of the three role clause properties. Consider the following clauses:

1. $ar(R, a, b) \leftarrow$
2. $ar(R, a, c) \leftarrow$
3. $ar(S, b, d) \leftarrow$
4. $ar(S, c, d) \leftarrow$

We will assume that clauses 1 and 2 are ground variants of the role clauses with which the body literals in the aforementioned DLE-clause were unified before $(b \simeq c \leftarrow)$ was derived. We may assume $[b] \neq [c]$, for otherwise, adding $(b \simeq c \leftarrow)$ would have no effect on any equivalence class. Clauses 3 and 4 represent arbitrary role clauses containing individuals from $[b]$ and $[c]$.

Let $\mathbf{B}$ denote the branch obtained from $\mathbf{B}'$ by deriving $(b \simeq c \leftarrow)$ using *Equality* (with an underlying *ref* inference) or *Split*. When referring to equivalence classes in $\mathbf{B}'$, the branch identifier will be omitted in the rest of this proof. E.g., $[b]$ and $[c]$ stand for $[b]_{\mathbf{B}'}$ and $[c]_{\mathbf{B}'}$, respectively. We will, however, use the identifier $\mathbf{B}$ whenever we talk about the (changed) equivalence classes in $\mathbf{B}$. In order to emphasize that $[b]$ and $[c]$ (in the old branch $\mathbf{B}'$) were merged, we will use the notation $[b + c]_{\mathbf{B}}$ to refer to the resulting equivalence class in the new branch $\mathbf{B}$. Note that $[b + c]_{\mathbf{B}} = [b]_{\mathbf{B}} = [c]_{\mathbf{B}}$. Also, to keep things

shorter, we will write $[a]$ succ $[b]$ to express that $[a]$ is a successor of $[b]$; likewise, 'desc' is short for 'is a descendant of'.

Remember that several useful properties were mentioned after the definition of *successors* and *predecessors* in the context of equivalence classes in section 6.3.1:

- $[a]$ succ $[b]$ implies that $min[a]$ is unnamed
- $[a]$ succ $[a]$ is a contradiction, so $[a]$ succ $[b]$ implies $[a] \neq [b]$ and hence $a \neq b$
- $[a]$ succ $[b]$ and $[a]$ succ $[c]$ implies $[b] = [c]$

It is also worth noting that when two equivalence classes are merged and the minimum of at least one of the classes was named, the minimum of the resulting class will be named, too, because named individuals are always smaller than unnamed ones.

We will now consider every relevant combination of properties that clauses 1-4 may satisfy and check which properties clauses 3 and 4 satisfy afterwards. We will see that in all possible cases, 3 and 4 must still satisfy property (1), (2) or (3) after the new equality clause was introduced, which will complete the Lemma's proof. The notation $x.y$ means that we assume that clause x satisfies property y in the current case.

First, we will consider the combination of cases 1.1 and 2.1.

1.1: $(ar(R, a, b) \leftarrow)$ satisfies (1), meaning that $min[a]$ and $min[b]$ are named

2.1: $(ar(R, a, c) \leftarrow)$ satisfies (1), meaning that $min[a]$ and $min[c]$ are named

Note that this implies that $min[b + c]_{\mathbf{B}}$ must also be named. Now consider the various properties that $(ar(S, b, d) \leftarrow)$ may satisfy before $[b]$ and $[c]$ are merged. For each case, we will check which property the clause satisfies afterwards:

1. This property implies that $min[d]_{\mathbf{B}}$ is named; since $min[b + c]_{\mathbf{B}}$ is also named, property (1) holds
2. This means that $min[d]$ is unnamed, which implies $[d] \neq [b]$ and $[d] \neq [c]$ since $min[b]$ and $min[c]$ are named; due to $[d] \neq [b]$ and $[d] \neq [c]$, the equivalence class $[d]$ is not affected by the merging of $[b]$ and $[c]$ (i.e., $[d] = [d]_{\mathbf{B}}$), so $[d]$ succ $[b]$ implies $[d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}}$, which implies that property (2) holds
3. Impossible: implies $[b]$ succ $[d]$, meaning $[b]$ is unnamed, but it is actually named (as implied by 1.1)

As for $(ar(S, b, d) \leftarrow)$, the cases are analogous to the above ones. This completes the treatment of the combination of cases 1.1 and 2.1.

The next combination to consider is 1.1 and 1.2.

1.1: $(ar(R, a, b) \leftarrow)$ satisfies property (1), meaning that $min[a]$ and $min[b]$ are named

2.2: $(ar(R, a, c) \leftarrow)$ satisfies property (2), which implies $[c]$ succ $[a]$ and hence $[c] \neq [a]$

Now consider the properties that $(ar(S, b, d) \leftarrow)$ may satisfy:

1. Implies that $min[b]$ and $min[d]$ (and hence $min[d]_{\mathbf{B}}$) are named, so $min[b + c]_{\mathbf{B}}$ must be named as well $\Rightarrow$ property (1) holds
2. Implies $[d]$ succ $[b]$, meaning $min[d]$ is unnamed $\Rightarrow$ $[d] \neq [b]$ (because $min[b]$ is named); case distinction:

- $[d] = [c]$: since $min[b]$ is named, $min[b + c]_{\mathbf{B}}$ ($= min[d]_{\mathbf{B}}$) must be named, too $\Rightarrow$ property (1) holds
- $[d] \neq [c]$: $[d]$ succ $[b] \Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}} \Rightarrow$ property (2) holds

3. Impossible: implies $[b]$ succ $[d]$, but $min[b]$ is named

As for $(ar(S, c, d) \leftarrow)$:

1. This means that $min[d]$ (and hence $min[d]_{\mathbf{B}}$) is named; since $min[b]$ is also named, $min[b + c]_{\mathbf{B}}$ must be named as well $\Rightarrow$ property (1) holds
2. Implies $[d]$ succ $[c]$, meaning that $min[d]$ is unnamed $\Rightarrow [d] \neq [b]$ (as $min[b]$ is named); case distinction:
   - $[d] = [c]$: since $min[b]$ is named, $min[b + c]_{\mathbf{B}}$ ($= min[d]_{\mathbf{B}}$) is also named $\Rightarrow$ property (1) holds
   - $[d] \neq [c]$: $[d]$ succ $[c] \Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}} \Rightarrow$ property (2) holds
3. Here, we have $[c]$ succ $[d]$ and $[c]$ succ $[a] \Rightarrow [a] = [d]$; as $min[a]$ is named, $min[d]$ (and hence $min[d]_{\mathbf{B}}$) is also named; $min[b]$ is named, too, so $min[b + c]_{\mathbf{B}}$ must also be named $\Rightarrow$ property (1) holds

Note that the combination 1.1 and 2.3 need not be considered because the former property implies that $min[a]$ is named whereas the latter implies that it is unnamed, a plain contradiction. Hence, the combination of 1.2 and 2.2 is next.

1.2: $(ar(R, a, b) \leftarrow)$ satisfies property (2), which implies $[b]$ succ $[a]$ ($\Rightarrow min[b]$ is unnamed) and hence $[b] \neq [a]$

2.2: $(ar(R, a, c) \leftarrow)$ satisfies property (2), which implies $[c]$ succ $[a]$ ($\Rightarrow min[c]$ is unnamed) and hence $[c] \neq [a]$

As for $(ar(S, b, d) \leftarrow)$:

1. Impossible: implies that $min[b]$ is named, but it is actually unnamed
2. Implies $[d]$ succ $[b]$ ($\Rightarrow [d] \neq [b]$); case distinction:
   - $[d] = [c]$: with $[d]$ succ $[b]$, we get $[c]$ succ $[b]$; $[c]$ succ $[a]$ (2.2) then implies $[a] = [b]$, which, together with $[b]$ succ $[a]$ (1.2), implies $[a]$ succ $[a] \Rightarrow$ impossible
   - $[d] \neq [c]$: $[d]$ succ $[b] \Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}} \Rightarrow$ property (2) holds
3. Implies $[b]$ succ $[d]$ ($\Rightarrow [d] \neq [b]$); with $[b]$ succ $[a]$ (1.2) we get $[a] = [d]$; case distinction:
   - $[d] = [c]$: we have $[c]$ succ $[a]$ (2.2) and $[a] = [d] = [c] \Rightarrow [c]$ succ $[c] \Rightarrow$ impossible
   - $[d] \neq [c]$: we have $[b]$ succ $[a]$ (1.2), $[c]$ succ $[a]$ (2.2) and $[a] = [d] \Rightarrow [b]$ succ $[d]$ and $[c]$ succ $[d]$, which together implies $[b + c]_{\mathbf{B}}$ succ $[d]_{\mathbf{B}} \Rightarrow$ property (3) holds

As for $(ar(S, c, d) \leftarrow)$, the three cases are analogous to the above ones.

1.2: $(ar(R, a, b) \leftarrow)$ satisfies property (2), which implies $[b]$ succ $[a]$ ($\Rightarrow min[b]$ is unnamed) and hence $[b] \neq [a]$

2.3: $(ar(R, a, c) \leftarrow)$ satisfies property (3), which implies $[a]$ succ $[c]$ ($\Rightarrow$ *min*$[a]$ is unnamed) and hence $[c] \neq [a]$

Note that $[b]$ succ $[a]$ and $[a]$ succ $[c]$ imply $[b]$ desc $[c]$ (but not $[b]$ succ $[c]$!).
As for $(ar(S, b, d) \leftarrow)$:

1. Impossible: implies that *min*$[b]$ is named, but it is actually unnamed
2. Implies $[d]$ succ $[b]$ ($\Rightarrow [d] \neq [b]$); case distinction:
   - $[d] = [c]$: $[d]$ succ $[b] \Rightarrow [c]$ succ $[b]$; with $[a]$ succ $[c]$ (2.3), this implies $[a]$ desc $[b] \Rightarrow$ impossible (contradicts $[b]$ succ $[a]$ (1.2))
   - $[d] \neq [c]$: $[d]$ succ $[b] \Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}} \Rightarrow$ property (2) holds
3. Implies $[b]$ succ $[d]$ ($\Rightarrow [d] \neq [b]$); with $[b]$ succ $[a]$ (1.2), we get $[a] = [d]$; $[a]$ succ $[c]$ (2.3) then implies $[d]$ succ $[c]$ ($\Rightarrow [d] \neq [c]$) $\Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}}$ (note that this does not contradict $[b]$ succ $[d]$) $\Rightarrow$ property (2) holds

As for $(ar(S, c, d) \leftarrow)$:

1. This means that *min*$[c]$ and *min*$[d]$ (and hence *min*$[d]_{\mathbf{B}}$) are named; hence, *min*$[b + c]_{\mathbf{B}}$ must be named, too $\Rightarrow$ property (1) holds
2. Implies $[d]$ succ $[c]$ ($\Rightarrow [d] \neq [c]$); case distinction:
   - $[d] = [b]$: $[d]$ succ $[c] \Rightarrow [b]$ succ $[c]$; with $[b]$ succ $[a]$ (1.2), we get $[a] = [c]$; $[a]$ succ $[c]$ (2.3) then implies $[a]$ succ $[a] \Rightarrow$ impossible
   - $[d] \neq [b]$: $[d]$ succ $[c] \Rightarrow [d]_{\mathbf{B}}$ succ $[b + c]_{\mathbf{B}} \Rightarrow$ property (2) holds
3. Implies $[c]$ succ $[d]$; with $[b]$ desc $[c]$ (1.2 and 2.3), we can conclude *min*$[c] \prec$ *min*$[b] \Rightarrow$ *min*$[b + c]_{\mathbf{B}}$ = *min*$[c]$ ($\in [d]$) $\Rightarrow [b+c]_{\mathbf{B}}$ succ $[d]_{\mathbf{B}}$ (all individuals in $[b+c]_{\mathbf{B}}$ are descendants of individuals in $[d]_{\mathbf{B}}$ and $[d]_{\mathbf{B}}$ contains the predecessor of *min*$[b + c]_{\mathbf{B}}$) $\Rightarrow$ property (3) holds

While the combination of 1.3 and 2.3 is possible, it would imply $[a]$ succ $[b]$, $[a]$ succ $[c]$ and hence $[b] = [c]$. As a result, the equivalence classes of b and c would not be affected by adding the clause $(b \simeq c \leftarrow)$, so this case is guaranteed to preserve the properties of all role clauses. □

Lemma 6.3.10. If $(\geq m\ R.C(a') \leftarrow) \in \mathbf{B}^n$ for an individual a' with $a' \preceq [a']$, a role R and $C = A$ or $C = \neg A$ for A an atomic concept, then $p_a \in (\geq m\ R.C)^I$ for every path $p_a \in P_{\mathbf{B}^n}$ with $tail'(p_a) = a'$.

Proof. Let $a = tail(p_a)$. By the definitions of $P_{\mathbf{B}^n}$ (6.3.4) and blocking (5.4.6), a cannot be blocked. If a' is directly blocked by a, the two individuals must have identical labels, so $(\geq m\ R.C(a') \leftarrow) \in \mathbf{B}^n$ and Lemma 6.3.7 imply $(\geq m\ R.C(a) \leftarrow) \in \mathbf{B}^n$. If a' is not blocked, we have $a = a'$ by the definition of $P_{\mathbf{B}^n}$, in which case $(\geq m\ R.C(a) \leftarrow) \in \mathbf{B}^n$ follows trivially from $(\geq m\ R.C(a') \leftarrow) \in \mathbf{B}^n$. Since $\mathbf{B}^n$ is saturated up to redundancy and since a cannot be blocked, *at-least* must have generated m successors $a_1, \ldots, a_m$ of a. (Note that, in practice, the names would contain the number restriction's label, but it was omitted here to keep the notation simple.) Consider the clauses that must have been added by *at-least* and what they imply about the clauses in the exhausted branch $\mathbf{B}^n$:

- $(ar(R, a, a_i) \leftarrow)$ for $1 \leq i \leq m$: By Lemma 6.3.6, $\mathbf{B}^n$ must thus contain the role clauses $(ar(R, a, min[a_i]) \leftarrow)$ for $1 \leq i \leq m$. (Note that $a = min[a]$ by the definition of $P_{\mathbf{B}^n}$.)

- $cc(C, a_i)$ for $1 \leq i \leq m$: If $C = A$, then $cc(C, a_i) = (A(a_i) \leftarrow)$ and we must have $(A(min[a_i]) \leftarrow) \in \mathbf{B}^n$ by Lemma 6.3.6. In the second case, $C = \neg A$, we have $cc(C, a_i) = (\leftarrow A(a_i))$. Hence, $(\leftarrow A(a_i))$ must be contained in $\mathbf{B}^n$ or, if it was removed by *Del* or *Simp*, it must at least follow from $\mathbf{B}^n$. This implies $(A(min[a_i]) \leftarrow) \notin \mathbf{B}^n$ since no contradiction was derived.

- $(\leftarrow a_i \simeq a_j)$ for all i, j with $1 \leq i < j \leq m$: These clauses imply $[a_i] \neq [a_j]$ for all i, j with $1 \leq i < j \leq m$.

(The *dom*-clauses generated by *at-least* are irrelevant in this context.)

We will now show that the above conclusions imply that there exist m different paths p_i such that $(p_a, p_i) \in R^I$ and $p_i \in C^I$ for all $1 \leq i \leq m$. According to Table 2.1 (semantics of $\mathcal{SHIQ}$), this implies $p_a \in (\geq m\ R.C)^I$, which is the statement we want to prove. By Lemma 6.3.9, each of the m role clauses $(ar(R, a, min[a_i]) \leftarrow)$ must satisfy one of the following properties:

1. $min[a]$ and $min[a_i]$ are named: Let $p_i = \left[\frac{min[a_i]}{min[a_i]}\right]$ denote the interpretation of the named individual $min[a_i]$. In this case, $(ar(R, a, min[a_i]) \leftarrow) \in \mathbf{B}^n$ implies that the tuple (p_a, p_i) must have been added to R^I by the first case of the definition of role interpretations. If we have $C = A$, $(A(min[a_i]) \leftarrow) \in \mathbf{B}^n$ implies $p_i \in A^I$ by the definition of concept interpretations. If $C = \neg A$, $(A(min[a_i]) \leftarrow) \notin \mathbf{B}^n$ implies $p_i \notin A^I$. Either way, we have $p_i \in C^I$.

2. $[a_i]$ is a successor of $[a]$: Let $p_i = \left[p_a \mid \frac{b}{min[a_i]}\right]$, where $b = min[a_i]$ or b directly blocks $min[a_i]$. Due to $tail(p_a) = a$, $p_a \in P_{\mathbf{B}^n}$ as well as the fact that $[a_i]$ is a successor of $[a]$ imply $p_i \in P_{\mathbf{B}^n}$ by the definition of $P_{\mathbf{B}^n}$. Then, as a result of $(ar(R, a, min[a_i]) \leftarrow) \in \mathbf{B}^n$, the tuple (p_a, p_i) must have been added to R^I by the second case of the definition of role interpretations.

   By the definitions of $P_{\mathbf{B}^n}$ and blocking, the labels of b and $min[a_i]$ must be identical. If $C = A$, $(A(min[a_i]) \leftarrow) \in \mathbf{B}^n$ and Lemma 6.3.7 thus imply $(A(b) \leftarrow) \in \mathbf{B}^n$ and hence $p_i \in A^I$. If $C = \neg A$, $(A(min[a_i]) \leftarrow) \notin \mathbf{B}^n$ implies that neither $min[a_i]$ nor b can be labeled with A, so we must have $(A(b) \leftarrow) \notin \mathbf{B}^n$ and thus $p_i \notin A^I$. In either case, we have $p_i \in C^I$.

3. $[a]$ is a successor of $[a_i]$: If a' is not blocked by a, we have $a = a'$. Let $p_a = [p_i \mid \frac{a}{a}]$, with $tail(p_i) = min[a_i]$. In this case, $(ar(R, a, min[a_i]) \leftarrow) \in \mathbf{B}^n$ implies that the tuple (p_a, p_i) must have been added to R^I by the third case of the definition of role interpretations. If $C = A$, then $(A(min[a_i]) \leftarrow) \in \mathbf{B}^n$ and $tail(p_i) = min[a_i]$ imply $p_i \in A^I$ by the definition of concept interpretations. In the case $C = \neg A$, $(A(min[a_i]) \leftarrow) \notin \mathbf{B}^n$ implies $p_i \notin A^I$. Again, we have $p_i \in C^I$ in either case.

   Otherwise, if a' is blocked by a, a' must have some predecessor a'_{pre} by the definition of blocking. Let a_{pre} denote the predecessor of a; as a is minimal in $[a]$ and since $[a]$ is a successor of $[a_i]$, we must have $a_{pre} \in [a_i]$. By the definition of blocking, the labels of (a_{pre}, a) and (a'_{pre}, a') as well as those of (a, a_{pre}) and (a', a'_{pre}) are identical. Due to $(ar(R, a, min[a_i]) \leftarrow) \in \mathbf{B}^n$, we must have $R \in \mathcal{L}_{\mathbf{B}^n}(a, a_{pre}) = \mathcal{L}_{\mathbf{B}^n}(a', a'_{pre})$ (if R is atomic) or $S \in \mathcal{L}_{\mathbf{B}^n}(a_{pre}, a) = \mathcal{L}_{\mathbf{B}^n}(a'_{pre}, a')$ (if R is inverse and $R^- = S$). Lemma 6.3.8 then implies $(ar(R, a', min[a'_{pre}]) \leftarrow) \in \mathbf{B}^n$. Let $p_a = [p_i \mid \frac{a}{a'}]$, with $tail(p_i) = min[a'_{pre}]$. Hence, the tuple (p_a, p_i) must have been added to R^I by the second or third case of the definition of role interpretations (depending on whether R is atomic or inverse).

   By the definition of blocking, the labels of a_{pre} and a'_{pre} must also be identical. In the case $C = A$, we have $(A(min[a_i]) \leftarrow) \in \mathbf{B}^n$; $a_{pre} \in [a_i]$ then implies that both a_{pre} and a'_{pre} are labeled with A. Hence, we must also have $(A(min[a'_{pre}]) \leftarrow) \in \mathbf{B}^n$ by Lemma 6.3.7, which implies $p_i \in A^I$ by the definition of concept interpretations. Using the same reasoning, we can conclude $p_i \notin A^I$ if $C = \neg A$. In either case, we have $p_i \in C^I$.

In the above case distinction, we have shown that there are m distinct paths p_i (for $1 \leq i \leq m$) such that $(p_a, p_i) \in R^I$ and $p_i \in C^I$. As a result, $p_a \in (\geq m\ R.C)^I$ must hold. □

For the following theorems, recall that I denotes the induced interpretation that was constructed according to Definition 6.3.5 from the exhausted branch $\mathbf{B}^n$ in a derivation from $DLE(\Delta(\Omega(\mathcal{K})))$, the set of DLE-clauses into which the $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$ was translated. We will now prove that I satisfies every clause in $DLE(\Delta(\Omega(\mathcal{K})))$, thus showing that the original knowledge base $\mathcal{K}$ must be satisfiable, too.

Theorem 6.3.1. *I satisfies the clauses in $DLE(\Delta(\Omega(\mathcal{K})))$ that were derived from the ABox $\Delta(\Omega(\mathcal{K}))_{\mathcal{A}}$ as well as the dom clauses that were added for range restriction purposes.*

Proof. Consider the different forms these clauses can take. Let a and b denote named individuals, A an atomic concept ($A = dom$ is possible), and R an atomic role. Remember that the initial branch $\mathbf{B}^0$ is labeled with the clauses in $DLE(\Delta(\Omega(\mathcal{K})))$.

- $A(a) \leftarrow$: $(A(a) \leftarrow) \in \mathbf{B}^0$ implies $(A(min[a]) \leftarrow) \in \mathbf{B}^n$ by Lemma 6.3.6. Since a is named, we have $a^I = \left[\frac{min[a]}{min[a]}\right]$; the definition of concept interpretations then implies $a^I \in A^I$.

- $\leftarrow A(a)$: If $a^I \in A^I$ *did* hold, $\mathbf{B}^n$ would have to contain $(A(min[a]) \leftarrow)$. This is impossible since $(\leftarrow A(a))$ must still follow from $\mathbf{B}^n$ (even if it was removed) and no contradiction was derived. As a result, we must have $a^I \notin A^I$, which makes the clause true in I.

- $a \simeq b \leftarrow$: By Lemma 6.3.3 and the definition of I we have $a^I = b^I$, which makes the assertion true.

- $R(a, b) \leftarrow$: By Lemma 6.3.6, we have $(R(min[a], min[b]) \leftarrow) \in \mathbf{B}^n$. By the first part of the definition of role interpretations, we must then have $((min[a])^I, (min[b])^I) \in R^I$. Due to $a^I = (min[a])^I$ and $b^I = (min[b])^I$ by the definition of the interpretation of named individuals, the clause must be true in I.

- $\leftarrow a \simeq b$: Since no contradiction was derived, $\mathbf{B}^n$ cannot contain an equality sequence that connects a to b. As a result, we must have $(a, b) \notin EQ_{\mathbf{B}^n}$ and thus $[a] \neq [b]$, which implies $a^I \neq b^I$.

- $\geq n\ R.C(a) \leftarrow$: We have $a^I = \left[\frac{min[a]}{min[a]}\right]$ and, by Lemma 6.3.6, $(\geq n\ R.C(min[a]) \leftarrow) \in \mathbf{B}^n$. Lemma 6.3.10 then implies $a^I \in (\geq n\ R.C)^I$, which makes the clause true in I.

□

Theorem 6.3.2. *I satisfies the clauses in $DLE(\Delta(\Omega(\mathcal{K})))$ that were derived from the RBox $\Delta(\Omega(\mathcal{K}))_{\mathcal{R}}$.*

Proof. As shown in Table 3.2, DLE-clauses derived from RBox axioms can only take the form $C = (ar(R, x, y) \leftarrow ar(S, x, y))$ for R and S atomic or inverse roles and x and y variables. $(p_a, p_b) \in S^I$ implies $D = (ar(S, tail(p_a), tail'(p_b)) \leftarrow) \in \mathbf{B}^n$ by the definition of role interpretations. This means that $E = (ar(R, tail(p_a), tail'(p_b)) \leftarrow)$ must be contained in $\mathbf{B}^n$, for otherwise it could be derived from C and D. (Note that E cannot be redundant because it has no smaller ground variants.) $E \in \mathbf{B}^n$ and the definition of role interpretations then imply $(p_a, p_b) \in R^I$, which means that C is true in I. □

Theorem 6.3.3. *I satisfies the clauses in $DLE(\Delta(\Omega(\mathcal{K})))$ that were derived from the TBox $\Delta(\Omega(\mathcal{K}))_{\mathcal{T}}$.*

Proof. (Note that large parts of this proof are analogous to Motik et al.'s proof of completeness in [13].) As seen in Table 3.2, all DLE-clauses derived from TBox axioms must be of the form

$$\bigvee_{i,j} y_i \simeq y_j \vee \bigvee_i D_i(x) \vee \bigvee_i E_i(y_i) \leftarrow \bigwedge_i A_i(x) \wedge \bigwedge_i ar(R_i, x, y_i) \wedge \bigwedge_i C_i(y_i) \tag{6.1}$$

for A_i, C_i and E_i atomic concepts, R_i roles and D_i blocking-relevant concepts. Now consider any variable mapping μ with $\mu(x) = p_x$ and $\mu(y_i) = p_{y_i}$ such that $p_x \in A_i^I$, $p_{y_i} \in C_i^I$, and $(p_x, p_{y_i}) \in R_i^I$. In other words, p_x and p_{y_i} satisfy the body of the clause. We will first show that this implies that certain clauses must be present in $\mathbf{B}^n$. The next step will then be to prove that the presence of these clauses implies $p_x \in D_i^I$ for at least one i, $p_{y_i} \in E_i^I$ for at least one i, or $p_{y_i} = p_{y_j}$ for some $i \neq j$, which would make the whole clause true in I.

For each pair p_x and p_{y_i}, there are various cases we need to distinguish. In all of them, let $s = tail(p_x)$ and $s' = tail'(p_x)$. $p_x \in A_i^I$ implies $(A_i(tail(p_x)) \leftarrow) \in \mathbf{B}^n$ by the definition of I. If $tail'(p_x)$ is not directly blocked by $tail(p_x)$, we have $tail(p_x) = tail'(p_x)$, which trivially implies $(A_i(tail'(p_x)) \leftarrow) \in \mathbf{B}^n$. If $tail'(p_x)$ is directly blocked by $tail(p_x)$, $(A_i(tail'(p_x)) \leftarrow) \in \mathbf{B}^n$ still follows from the definition of blocking, according to which $tail(p_x)$ and $tail'(p_x)$ must have identical labels, and Lemma 6.3.7. As a result, we have $(A_i(s) \leftarrow) \in \mathbf{B}^n$ and $(A_i(s') \leftarrow) \in \mathbf{B}^n$ in all of the following cases. By the same reasoning, we can conclude $(C_i(tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ and $(C_i(tail'(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ from $p_{y_i} \in C_i^I$. To make further statements about the clauses in $\mathbf{B}^n$, we need to distinguish five cases. Each case represents a condition that p_x and p_{y_i} must have satisfied in order to be added to R^I according to the definition of I. In case 1, the tuple must have been added to R^I by the first line of the definition of role interpretations. In cases 2 and 3, it must have been added by the second line, whereas the third line must have applied in cases 4 and 5:

1. p_x and p_{y_i} are interpretations of named individuals.

Since p_{y_i} is the interpretation of a named individual, we have $tail(p_{y_i}) = tail'(p_{y_i})$. If we let $t_i = tail'(p_{y_i})$, this gives us $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$. Finally, $(p_x, p_{y_i}) \in R_i^I$ together with the first line of the definition of role interpretations implies $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$.

2. s' is not blocked and $[tail'(p_{y_i})]$ is a successor of $[s]$.

By letting $t_i = tail'(p_{y_i})$ as in the first case, we have $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$. Furthermore, the fact that s' is not blocked implies $s = s'$ and, equivalently, $tail(p_x) = tail'(p_x)$. Since we assumed that $[tail'(p_{y_i})]$ $(= [t_i])$ is a successor of $[s']$ $(= [s])$, $(p_x, p_{y_i}) \in R_i^I$ implies $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$ by the second line of the definition of role interpretations.

3. s' is directly blocked by s and $[tail'(p_{y_i})]$ is a successor of $[s]$.

Once more, let $t_i = tail'(p_{y_i})$, which implies $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$. $(p_x, p_{y_i}) \in R_i^I$, with $[tail'(p_{y_i})]$ being a successor of $[tail(p_x)]$, implies $(ar(R_i, tail(p_x), tail'(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ by the second line of the definition of role interpretations, and, equivalently, $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$.

4. s' is not blocked and $[tail(p_{y_i})]$ is the predecessor of $[s']$.

Since s' is not blocked, we must have $s = s'$. By setting $t_i = tail(p_{y_i})$ we thus have $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$. $(ar(R_i, s, tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ follows from $(p_x, p_{y_i}) \in R_i^I$ by the third line of the definition of role interpretations, with $t_i = tail(p_{y_i})$ implying $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$.

5. s' is directly blocked by s and $[tail(p_{y_i})]$ is the predecessor of $[s']$.

We must have $(ar(R_i, s', tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ due to $(p_x, p_{y_i}) \in R_i^I$ by the third line of the definition of role interpretations. According to the definition of blocking, s (the blocking individual) must have some predecessor s_{pre}. Let $t_i = min[s_{pre}]$. Remember that the definition of blocking implies that the label of the predecessor (some $u \in [tail(p_{y_i})]$) of a blocked individual (s') must be identical to the label of the predecessor (s_{pre}) of the blocking individual (s). $(C_i(tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ implies that u is labeled with C_i, which entails that s_{pre} is labeled with C_i as well. By Lemma 6.3.7 and due to $t_i = min[s_{pre}]$, we thus must have $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$.

By the definition of blocking, the labels of (u, s') and (s', u) must be identical to the labels of (s_{pre}, s) and (s, s_{pre}), respectively. Hence, we can conclude $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$ from $(ar(R_i, s', tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ and Lemma 6.3.6.

In summary, due to the different definitions of t_i, we have always have $(A_i(s) \leftarrow) \in \mathbf{B}^n$, $(C_i(t_i) \leftarrow) \in \mathbf{B}^n$ as well as $(ar(R_i, s, t_i) \leftarrow) \in \mathbf{B}^n$.

Let F denote the ground DL-clause obtained from clause 6.1 by instantiating x with s and each y_i with its respective t_i. As shown above, all of F's body literals also occur in the heads of positive unit clauses in $\mathbf{B}^n$, so the calculus must have derived a positive unit clause containing one of F's head literals from clause 6.1 unless at least one head literal already followed from other clauses in $\mathbf{B}^n$. We will briefly analyzes what that implies about the clauses in $\mathbf{B}^n$.

If some $D_i(s)$ or $E_i(t_i)$ follows from other clauses in $\mathbf{B}^n$, then $\mathbf{B}^n$ must contain a ground variant of $(D_i(s) \leftarrow)$ or $(E_i(t_i) \leftarrow)$, respectively. But then, by Lemma 6.3.6, $\mathbf{B}^n$ would also contain $(D_i(s) \leftarrow)$ or $(E_i(t_i) \leftarrow)$, respectively.

First, we will assume that $(D_i(s) \leftarrow) \in \mathbf{B}^n$ holds, which implies $(D_i(s') \leftarrow) \in \mathbf{B}^n$ by the definition of blocking (if s' is not blocked, this follows trivially from $s = s'$). If D_i is an atomic concept, then $p_x \in D_i^I$ is implied by the definition of I. Otherwise, if D_i is a $\geq$-number restriction, $p_x \in D_i^I$ follows from Lemma 6.3.10.

Now assume $(E_i(t_i) \leftarrow) \in \mathbf{B}^n$ and recall that t_i can take on three different values in the five cases we distinguished:

- $t_i = tail'(p_{y_i})$ (cases 1-3): $(E_i(tail(p_{y_i})) \leftarrow)$ then follows from $(E_i(t_i) \leftarrow) = (E_i(tail'(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ by the definition of blocking (the blocker $tail(p_{y_i})$ and the blocked individual $tail'(p_{y_i})$ must have identical labels; if $tail'(p_{y_i})$ is not blocked, the statement follows trivially from $tail(p_{y_i}) = tail'(p_{y_i})$).
- $t_i = tail(p_{y_i})$ (case 4): $(E_i(tail(p_{y_i})) \leftarrow)$ follows trivially from $(E_i(t_i) \leftarrow) \in \mathbf{B}^n$.
- $t_i = min[s_{pre}]$ (case 5): By the definition of blocking, t_i and $tail(p_{y_i})$ must have identical labels since an individual in $[t_i]$ is the predecessor of s and an individual in $[tail(p_{y_i})]$ is the predecessor of s', which is directly blocked by s. $(E_i(t_i) \leftarrow) \in \mathbf{B}^n$ then implies $E_i \in \mathcal{L}_{\mathbf{B}^n}(t_i) = \mathcal{L}_{\mathbf{B}^n}(tail(p_{y_i}))$. Hence, by Lemma 6.3.7, we must have $(E_i(tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$.

Since we always have $(E_i(tail(p_{y_i})) \leftarrow) \in \mathbf{B}^n$ and since E_i is an atomic concept, $p_{y_i} \in E_i^I$ must hold according to the definition of concept interpretations in I.

If none of the concept atoms in the head of F follow from $\mathbf{B}^n$, one of the equality atoms must be E-entailed by $\mathbf{B}^n$. As a result, the individuals on the left-hand and right-hand side must be connected by an equality sequence, which means that they are members of the same equivalence class. Since all t_i are minimal in their respective equivalence class (as implied by the definitions in cases 1-5 above; remember that all individuals in a path are minimal), this entails $t_i = t_j$ for some $i \neq j$. In other words, the head contains an obvious tautology and the clause is redundant, which means the trivial literal will not appear in a unit clause in $\mathbf{B}^n$. This difference compared to the concept atoms does not matter, though, as we will now see that $t_i = t_j$ implies $p_{y_i} = p_{y_j}$, which still makes the axiom true in I. Once more, we will distinguish various cases (more precisely: combinations of the five cases in which t_i and t_j may have been defined) and show that $p_{y_i} = p_{y_j}$ must hold in all of them:

- $t_i = tail'(p_{y_i})$ (cases 1-3) and $t_j = tail'(p_{y_j})$ (cases 1-3): If both p_{y_i} and p_{y_j} are interpretations of named individuals (i.e., both t_i and t_j were defined in case 1), we have $p_{y_i} = \left[\frac{t_i}{t_i}\right]$ and $p_{y_j} = \left[\frac{t_j}{t_j}\right]$, in which case $t_i = t_j$ implies $p_{y_i} = p_{y_j}$. Otherwise, if $[t_i]$ $(= [t_j])$ is a successor of $[s]$ (i.e., t_i and t_j were defined in case 2 or 3), we must have $p_{y_i} = p_{y_j} = \left[p_x \mid \frac{t_i}{t_i}\right]$ according to the definition of

role interpretations. It is impossible for either p_{y_i} or p_{y_j} to denote the interpretation of a named individual if $[t_i]$ is a successor of $[s]$, as that would imply that the named individual t_i has a predecessor in $[s]$. Thus, the remaining combinations of cases cannot occur and need not be considered.

- $t_i = tail'(p_{y_i})$ (cases 1-3) and $t_j = tail(p_{y_j})$ (case 4): Since t_j was defined in case 4, s' cannot be blocked (which rules out case 3 for t_i) and $[t_j]$ must be the predecessor of $[s']$, so s' cannot be named. As p_x was assumed to be the interpretation of a named individual in case 1 (which would imply that s' is named), t_i cannot have been assigned in that case. This leaves us with case 2 as the only case that can apply to t_i. But then, $[t_i]$ must be a successor of $[s]$ (= $[s']$), which is impossible since we already assumed that $[t_j]$ (= $[t_i]$) is the *predecessor* of $[s']$. Hence, all of these combinations of cases are impossible.

- $t_i = tail'(p_{y_i})$ (cases 1-3) and $t_j = min[s_{pre}]$, where s_{pre} is the predecessor of s (case 5): The fact that s directly blocks s' in case 5 rules out cases 1 and 2 for t_i, so we only need to consider case 3. Hence $[t_i]$ must be a successor of $[s]$, which implies $s \prec t_i = t_j$. However, since predecessors are always smaller in $\prec$ than their successors, we can also conclude $t_i = t_j = min[s_{pre}] \preceq s_{pre} \prec s$, a plain contradiction. Thus, this case is impossible.

- Both $[tail(p_{y_i})]$ and $[tail(p_{y_j})]$ are predecessors of $[s']$ (cases 4-5): Note that case 4 and case 5 cannot have occurred together because s' is blocked in 5 but not in 4. When either 4 or 5 applies to *both* t_i and t_j, we must have $p_x = \left[p_{y_i} \mid \frac{s}{s'}\right]$ and $p_x = \left[p_{y_j} \mid \frac{s}{s'}\right]$ (with $s = s'$ in case 4) since an equivalence class can only have one predecessor, which implies $p_{y_i} = p_{y_j}$.

Note that in all the cases that can actually occur, we have $p_{y_i} = p_{y_j}$, which would make the whole DL-clause 6.1 true in I. □

Theorem 6.3.4 (**Completeness).** Given a $\mathcal{SHIQ}$ knowledge base $\mathcal{K}$, if there exists a fair DLE-Hyper Tableau derivation from $DLE(\Delta(\Omega(\mathcal{K})))$ that contains an exhausted branch, then $\mathcal{K}$ is satisfiable.

Proof. By Theorem 3.5.1, $\mathcal{K}$ and $DLE(\Delta(\Omega(\mathcal{K})))$ are equisatisfiable. By Theorems 6.3.1, 6.3.2 and 6.3.3, a model of $DLE(\Delta(\Omega(\mathcal{K})))$ can be constructed from an exhausted branch in a DLE-Hyper Tableau derivation from $DLE(\Delta(\Omega(\mathcal{K})))$. As a result, $\mathcal{K}$ must be satisfiable as well. □

Chapter 7

Related Work and Ideas for Future Extensions

This chapter serves to present related work and point out how some of the ideas found in it may be adapted to extend the DLE-Hyper Tableau calculus in the future.

As indicated by the number of times [13] was cited in this thesis, the *at-least* inference rule as well as the completeness proof in chapter 6 were heavily inspired by Motik et al.'s work on the original HermiT calculus. Hence, the following part mentions some of the commonalities and differences between HermiT and the DLE-Hyper Tableau calculus.

Unlike the DLE-Hyper Tableau calculus, Motik et al.'s calculus works with hyper tableaux over ABoxes, combined with a fixed set of DL-clauses. Using the assertions in an ABox, the body literals of a DL-clause are resolved away and the resulting head atoms are added to new ABoxes (one per atom). This hyper-resolution step basically corresponds to the E-Hyper Tableau calculus's *Split* rule.

The way $\geq$-number restrictions are handled is similar to the *at-least* rule, but instead of blindly generating successors for an individual a occurring in a number restriction of the form $\geq n\ R.C$, Motik et al.'s $\geq$-rule first checks whether a already has n distinct R-successors that satisfy C. If this is not the case, n successors are generated. The DLE-Hyper Tableau calculus's 'blind' approach was chosen because the

Another important difference lies in the way equality is handled. When encountering an assertion of the form $a \approx b$ (for $a \neq b$), HermiT's calculus replaces either a or b by the respective other individual in every ABox assertion, and all assertions containing successors of the individual that was replaced are deleted. This process is called *merging*. In contrast, the DLE-Hyper Tableau calculus, which inherited the built-in equality handling of the E-Hyper Tableau calculus, would derive new clauses from $a \simeq b$ using its superposition-based inference rules. It would not delete existing clauses right away, although this may later be done by *Del* or *Simp*. As a result, an exhausted DLE-Hyper Tableau branch may contain different individuals that are to be interpreted as equivalent, which is why we needed to work with equivalence classes and adapt the definition of pairwise anywhere blocking from [13] accordingly.

In 2009, Motik et al. [14] presented an extension of the calculus described above and proved that it is a decision procedure for $\mathcal{SHOIQ}^+$, an even more expressive Description Logic than $\mathcal{SHIQ}$. It remains to be seen whether it will be possible to adapt the DLE-Hyper Tableau calculus in a similar manner.

While Bry and Torge's work on finite satisfiability [10] focused on first-order logic rather than Description Logic, their 'Extended Positive tableaux' (EP tableaux) calculus - a decision procedure for finite satisfiability - uses an interesting way of expanding existential quantifiers in *PRQ formulas*. PRQ formulas are a fragment of first-order logic that has the same expressive power as full first-order logic. Their heads, like the head of a DLE-clause, may also contain existential quantifiers, which is why Bry

and Torge's calculus is relevant to our interests. In the following, we will briefly analyze how existential quantifiers are treated in EP tableaux.

Let $c_1, \ldots, c_k$ denote the constants occurring in a given branch. When encountering an existentially quantified formula of the form $\exists x E(x)$, the calculus creates $k + 1$ new leaf nodes, with the first k nodes containing the formulas $E[x/c_1], \ldots, E[x/c_k]$, respectively. The last node, however, will contain $E[x/c_{new}]$, where c_{new} represents a new constant not in $\{c_1, \ldots, c_k\}$. The calculus will only start working on this last branch when all preceding branches have been closed. This essentially means that it first checks whether an existing constant can be used to instantiate x and construct a model of the given clause set; if the search was not successful, the calculus checks whether the branch containing the fresh constant c_{new} is satisfiable.

Remember that *at-least* simply generates n successors for a given non-blocked individual if it has not generated them yet. In contrast, Motik et al.'s $\geq$-rule first checks whether an individual already has n distinct successors before generating new ones. However, even if the individual already has $n - 1$ distinct successors, their calculus will generate n new successors instead of trying to use existing individuals as successors. Adapting Bry and Torge's technique to $\geq$-number restrictions would mean that an expansion rule for these number restrictions would first check whether any set of n existing individuals can be used as successors. If this was impossible, the rule might add fresh successors one at a time and check whether a model can be found for the k fresh successors it has already added and any set of $n - k$ existing individuals that might used as successors. As a less computationally complex alternative, it could simply generate n fresh successors right away if no subset of the existing individuals can be used. Future research will have to show whether it would make sense to add any of these techniques to the DLE-Hyper Tableau calculus.

While this thesis was in the making, Markus Bender extended both the theoretical E-Hyper Tableau calculus as well as its implementation in E-KRHyper by adding support for *distinct object identifiers* (DOI) [8]. It is now possible to declare a set of DOIs (constants) for which the *unique names assumption* holds; that is, two DOIs a and b are treated as different from one another simply because their names differ. Hence, adding inequality clauses such as $(\leftarrow a \simeq b)$ would be unnecessary and new inference rules are used to detect contradictions such as $(a \simeq b \leftarrow)$ (this clause cannot hold in any model because a and b must have different interpretations). Right now, however, the E-Hyper Tableau calculus can only handle one single set of DOIs.

Remember that the *at-least* rule generates $\binom{n}{2}$ inequality clauses every time it is applied to an at-least clause of the form $(\geq n\ R.C_l(a) \leftarrow)$. It would be nice to be able to declare a new class of DOIs that is unique for a and l such that the successors $a.l.1, \ldots, a.l.n$ are recognized as distinct from one another without the need to add the corresponding inequality clauses. The new inference rules would then take care of handling contradictory atoms such as $a.l.1 \simeq a.l.2$. This would result in multiple classes of DOIs being created on-the-fly. Of course, individuals from different classes may still be equivalent; they would only be implicitly distinct from other members of their own class. Depending on the amount of $\geq$-number restrictions in a clause set, this could significantly reduce the number of clauses being generated and hopefully speed up the reasoning process.

Chapter 8

Conclusion

In this thesis, I presented the DLE-Hyper Tableau calculus, an extension of the E-Hyper Tableau calculus that is capable of deciding the satisfiability of $\mathcal{SHIQ}$ knowledge bases. After describing the transformation of a $\mathcal{SHIQ}$ knowledge base into a set of DLE-clauses, most of which is based on Motik et al.'s work [13], I covered both the existing E-Hyper Tableau calculus as well as the extensions that were made to obtain the DLE-Hyper Tableau calculus. Afterwards, I proved that the calculus is indeed sound, terminating and complete, which means that it is a decision procedure.

As mentioned in the previous chapter, there are various ideas floating around regarding how to further extend the calculus and increase its deductive power and efficiency. In the short term, however, the focus will be on refining the actual implementation in E-KRHyper. As a couple of changes were made to the calculus shortly before this thesis was finished, the current implementation is still preliminary. Once this has been taken care of, extensive tests should be performed to evaluate E-KRHyper's performance compared to other reasoners such as HermiT. The results will be presented in future articles and theses.

Bibliography

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[2] F. Baader, I. Horrocks, and U. Sattler. Description Logics as Ontology Languages for the Semantic Web. In *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, volume 2605 of *Lecture Notes in Artificial Intelligence*, pages 228–248. Springer, 2005.

[3] F. Baader, I. Horrocks, and U. Sattler. Description Logics. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008.

[4] L. Bachmair and H. Ganzinger. Equational Reasoning in Saturation-Based Theorem Proving. In W. Bibel and P. H. Schmitt, editors, *Automated Deduction - A Basis for Applications, vol. I*, chapter 11, pages 352–397. Kluwer, 1998.

[5] P. Baumgartner and R. Schmidt. Blocking and Other Enhancements for Bottom-Up Model Generation Methods. In U. Furbach and N. Shankar, editors, *Automated Reasoning – Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 125–139. Springer, 2006.

[6] P. Baumgartner, U. Furbach, and I. Niemelä. Hyper Tableaux. In *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA 1996)*, volume 1126 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1996.

[7] P. Baumgartner, U. Furbach, and B. Pelzer. Hyper Tableaux with Equality. In *Proceedings of the 21st International Conference on Automated Deduction (CADE 2007)*, volume 4603 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.

[8] M. Bender. Extending the E-Hyper Tableau Calculus for Reasoning with the Unique Name Assumption. Diplomarbeit (diploma thesis), 2012.

[9] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[10] F. Bry and S. Torge. A Deduction Method Complete for Refutation and Finite Satisfiability. In *Proceedings of the European Workshop on Logics in Artificial Intelligence (JELIA 1998)*, volume 1489 of *Lecture Notes in Computer Science*, pages 122–138, 1998.

[11] R. Manthey and F. Bry. SATCHMO: A Teorem Prover implemented in Prolog. In *Proceedings of the 9th International Conference on Automated Deduction (CADE 1988)*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.

[12] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe, 2006.

[13] B. Motik, R. Shearer, and I. Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In *Proceedings of the 21st International Conference on Automated Deduction (CADE 2007)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 67–83. Springer, 2007.

[14] B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.

[15] B. Pelzer. E-KRHyper - Extending the KRHyper Theorem Prover with Equality Reasoning. Diplomarbeit (diploma thesis), 2007.

[16] B. Pelzer and C. Wernhard. System Description: E-KRHyper. In *Proceedings of the 21st International Conference on Automated Deduction (CADE 2007)*, volume 4603 of *Lecture Notes in Artificial Intelligence*. Springer, 2007.

[17] J. A. Robinson. Automated Deduction with Hyper-Resolution. *International Journal of Computer Mathematics*, 1:227–234, 1965.