Towards richer rule languages with PTime data complexity for the Semantic Web

Linh Anh Nguyen\textsuperscript{a,b,*} and Thi-Bich-Loc Nguyen\textsuperscript{c} and Andrzej Szalas\textsuperscript{b,d}

\textsuperscript{a} Division of Knowledge and System Engineering for ICT, Faculty of Information Technology, Ton Duc Thang University, No. 19, Nguyen Huu Tho Street, Tan Phong Ward, District 7, Ho Chi Minh City, Vietnam
\textsuperscript{b} Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
\textsuperscript{c} Department of Information Technology, Hue University of Sciences, 77 Nguyen Hue, Hue city, Vietnam
\textsuperscript{d} Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden

Abstract

We introduce a Horn description logic called Horn-DL, which is strictly and essentially richer than Horn-\textsuperscript{Reg}, Horn-\textsuperscript{SHIQ} and Horn-\textsuperscript{SROIQ}, while still has PTime data complexity. In comparison with Horn-\textsuperscript{SROIQ}, Horn-DL additionally allows the universal role and assertions of the form \texttt{irreflexive}(s), \texttt{\neg s(a,b), a \neq b}. More importantly, in contrast to all the well-known Horn fragments \textsuperscript{EL}, \textsuperscript{DL-Lite}, \textsuperscript{DLP}, Horn-\textsuperscript{SHIQ}, Horn-\textsuperscript{SROIQ} of description logics, Horn-DL allows a form of the concept constructor “universal restriction” to appear at the left hand side of terminological inclusion axioms. Namely, a universal restriction can be used in such places in conjunction with the corresponding existential restriction. We develop the first algorithm with PTime data complexity for checking satisfiability of Horn-DL knowledge bases.

Keywords: description logics, Semantic Web, rule-based languages, Horn fragments, Horn-DL

1. Introduction

The Semantic Web is a rapidly growing research area that has received lots of attention in the last decade (e.g., [39] [13] [14] [35] [31]). One of the layers of the Semantic Web is OWL (Web Ontology Language). The first version of OWL is based on the description logic \textsuperscript{SHIQ}, the second version OWL 2, recommended by W3C in 2009, is based on the description logic \textsuperscript{SROIQ} [17]. These logics are highly expressive but have intractable combined complexity (\textsuperscript{ExpTime}-complete and \textsuperscript{N2ExpTime}-complete, respectively) and data complexity (NP-hard) for basic reasoning problems. Thus, W3C also recommended profiles OWL 2 \textsuperscript{EL}, OWL 2 \textsuperscript{QL} and OWL 2 \textsuperscript{RL}, which are restricted sublanguages of OWL 2 Full with PTime data complexity\textsuperscript{1}.

Description logics (DLs) are formal languages suitable for representing terminological knowledge. They are of particular importance in providing a logical formalism for ontologies and the Semantic Web. DLs represent the domain of interest in terms of concepts, individuals, and roles. A concept is interpreted as a set of individuals, while a role is interpreted as a binary relation among individuals. A knowledge base in a DL consists of axioms about roles (grouped into an RBox), terminology axioms (grouped into a TBox), and assertions about individuals (grouped into an ABox). A DL is usually specified by: i) a set of constructors that allow building complex concepts and complex roles from concept names, role names and individual

\textsuperscript{*}Corresponding author.

Email addresses: nguyenanhlinh@tdt.edu.vn (Linh Anh Nguyen), nguyen@minuw.edu.pl (Linh Anh Nguyen), ntbichloc@hueuni.edu.vn (Thi-Bich-Loc Nguyen), andsz@minuw.edu.pl (Andrzej Szalas)

\textsuperscript{1}The full names of the complexity classes mentioned in this paper are as follows: PTime means deterministic polynomial time, NP means nondeterministic polynomial time, \textsuperscript{ExpTime} means deterministic exponential time, \textsuperscript{N2ExpTime} means nondeterministic double exponential time. The data complexity is measured in the size of the ABox, which is in the “reduced” form, while assuming that the RBox, the TBox and the query are fixed.

Concept constructors of $\text{ALC}$

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>complement</td>
<td>$\neg C$</td>
<td>$\neg \text{Male}$</td>
</tr>
<tr>
<td>intersection</td>
<td>$C \cap D$</td>
<td>$\text{Human} \cap \text{Male}$</td>
</tr>
<tr>
<td>union</td>
<td>$C \sqcup D$</td>
<td>$\text{Doctor} \sqcup \text{Lawyer}$</td>
</tr>
<tr>
<td>existential restriction</td>
<td>$\exists r.C$</td>
<td>$\exists \text{hasChild}.\text{Male}$</td>
</tr>
<tr>
<td>universal restriction</td>
<td>$\forall r.C$</td>
<td>$\forall \text{hasChild}.\text{Female}$</td>
</tr>
</tbody>
</table>

Additional constructors/features of other DLs

<table>
<thead>
<tr>
<th>Constructor/Feature</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>inverse roles ($\mathcal{I}$)</td>
<td>$r^-$</td>
<td>$\text{hasChild}$ (i.e., $\text{hasParent}$)</td>
</tr>
<tr>
<td>qualified number restrictions ($\mathcal{Q}$)</td>
<td>$\geq n R.C$</td>
<td>$\geq 3 \text{hasChild}.\text{Male}$</td>
</tr>
<tr>
<td>$\leq n R.C$</td>
<td>$\leq 2 \text{hasParent}.\top$</td>
<td></td>
</tr>
<tr>
<td>nominals ($\mathcal{O}$)</td>
<td>${a}$</td>
<td>${\text{John}}$</td>
</tr>
<tr>
<td>hierarchies of roles ($\mathcal{H}$)</td>
<td>$R \sqsubseteq S$</td>
<td>$\text{hasChild} \sqsubseteq \text{hasDescendant}$</td>
</tr>
<tr>
<td>transitive roles ($\mathcal{S}$)</td>
<td>$R \circ R \sqsubseteq R$</td>
<td>$\text{hasDescendant} \circ \text{hasDescendant} \sqsubseteq \text{hasDescendant}$</td>
</tr>
</tbody>
</table>

Table 1: Concept constructors for $\text{ALC}$ and some additional constructors/features of other DLs.

names, ii) allowed forms of axioms and assertions. The basic DL $\text{ALC}$ allows basic concept constructors listed in Table 1 but does not allow role constructors nor role axioms. The most common additional features for extending $\text{ALC}$ are also listed in Table 1: $\mathcal{I}$ is a role constructor, $\mathcal{Q}$ and $\mathcal{O}$ are concept constructors, while $\mathcal{H}$ and $\mathcal{S}$ are allowed forms of role axioms.

1.1. Rule languages in description logics

Rule languages in DLs have attracted lots of attention due to their applications to the Semantic Web. The profiles OWL 2 EL, OWL 2 QL and OWL 2 RL of OWL 2 are based on the families of DLs $\mathcal{EL}$ [2, 3], DL-Lite [5] and DLP [16], which are monotonic rule languages with PTime data complexity. Such monotonic rule languages are Horn fragments of the corresponding full languages with appropriate restrictions adopted to eliminate nondeterminism.

A number of Horn fragments of DLs with PTime data complexity have also been investigated in [18, 21, 23, 37, 35]. The fragments Horn-SHIQ [18] and Horn-SROIQ [35] are notable, with considerable rich sets of allowed constructors and features, where Horn-SROIQ is richer than Horn-SHIQ. The combined complexities of Horn fragments of DLs were considered, amongst others, in [22]. Some tractable Horn fragments of DLs without ABoxes have also been isolated in [2, 4]. Combinations of rule languages like Datalog or its extensions with DLs have also been studied in a considerable number of works [8, 24, 26, 38, 20, 25, 9, 12, 6].

To eliminate nondeterminism, all $\mathcal{EL}$ [2, 3], DL-Lite [5], DLP [16], Horn-SHIQ [18] and Horn-SROIQ [35] disallow (any form of) the universal restriction at the left hand side of $\sqsubseteq$ in terminological axioms. The problem is that roles are not required to be serial (i.e., satisfying the condition $\forall x \exists y R(x, y)$) and the general Horn fragment of the basic DL $\text{ALC}$ that allows $\forall R.C$ at the left hand side of $\sqsubseteq$ has NP-complete data complexity [30].

In [28] Nguyen introduced the deterministic Horn fragment of $\text{ALC}$, in which the constructor $\forall R.C$ is allowed at the left hand side of $\sqsubseteq$ in the combination with $\exists R.C$ (in the form $\forall R.C \sqcap \exists R.C$, denoted by $\forall \exists R.C$ [4]). He proved that that fragment has PTIME data complexity by providing a bottom-up method for constructing a (logically) least model for a given deterministic positive knowledge base in the restricted language. In [30] Nguyen applied the method of [28] to regular DL $\mathcal{Re}g$, which extends $\text{ALC}$ with regular role inclusion axioms characterized by finite automata. Let us denote the Horn fragment of $\mathcal{Re}g$ that allows the constructor $\forall R.C$ at the left hand side of $\sqsubseteq$ by Horn-$\mathcal{Re}g$. As not every positive Horn-$\mathcal{Re}g$ knowledge base has a (logically) least model, Nguyen [30] proposed to approximate the instance checking problem in Horn-$\mathcal{Re}g$ by using weakenings with PTIME data complexity.
In [33] we studied a Horn fragment called Horn-Reg$^I$ of the regular DL with inverse Reg$^I$ and provided an algorithm with PTIME data complexity for checking satisfiability of Horn-Reg$^I$ knowledge bases. This fragment extends Horn-Reg with inverse roles. Our work [33] overcomes the difficulties encountered in [28, 30] by using the top-down rather than bottom-up approach, and thus enables to show that both Horn-Reg and Horn-Reg$^I$ have PTIME data complexity, solving an open problem of [30].

1.2. Constructivism in defining rules
We now justify the usefulness of the concept constructor ∀∃. It is related to constructivism in defining rules [30]. Consider the TBox $T$ consisting of the following rules:

\[
\begin{align*}
\exists \text{hasChild}.\top & \sqsubseteq \text{Parent} \\
\forall \text{hasChild} \cdot \text{Happy} & \sqsubseteq \text{HappyParent} \\
\text{HappyParent} & \sqsubseteq \text{Happy}.
\end{align*}
\]

The symbol $\top$ stands for truth and the first rule states that if someone has a child then he or she is a parent. The second rule states that if all children of someone are happy then that person is a happy parent. The meaning of third rule is clear.

Let $R$ be the empty RBox and $A$ be the ABox specified by:

\[
A = \{ \text{hasChild}(\text{Jane}, \text{Peter}), \text{hasChild}(\text{Peter}, \text{Christ}), \leq 1 \text{hasChild}.\top)(\text{Peter}), \text{Happy}(\text{Christ}) \}.
\]

The first assertion states that Peter is a child of Jane, the second assertion states that Christ is a child of Peter, the third assertion states that Peter has no more than one child, and the fourth assertion states that Christ is happy.

Consider the knowledge base $\langle R, T, A \rangle$. As expected, we have $\langle R, T, A \rangle \models (\text{Parent} \sqcup \text{HappyParent})(\text{Peter})$, which means that Peter is a happy parent is a logical consequence of the knowledge base; but unexpectedly, we also have $\langle R, T, A \rangle \models (\text{Parent} \sqcup \text{HappyParent})(\text{Christ})$.

The reason is that the premise of the rule (2) is not “constructive”. In other words, the instance $\exists \text{hasChild}.\top \sqcup \forall \text{hasChild}.\neg \top$ of the “law of excluded middle” holds for every individual $a$. If $a$ satisfies $\exists \text{hasChild}.\top$ then it is an instance of Parent; else it satisfies $\forall \text{hasChild}.\neg \top$, and therefore satisfies any formula $\forall \text{hasChild}.C$, and hence is an instance of HappyParent (by (2)).

The rule (2) should be changed to

\[
\text{Parent} \sqcap \forall \text{hasChild} \cdot \text{Happy} \sqsubseteq \text{HappyParent}
\]

or to

\[
\forall \exists \text{hasChild} \cdot \text{Happy} \sqsubseteq \text{HappyParent}.
\]

This shows that, in premises of rules, the constructor $\forall \exists$ is more “constructive” than $\forall$.

1.3. Our contributions
This work is an extension of our conference papers [32, 33]. In this work, we introduce a Horn description logic called Horn-DL, which is strictly and essentially richer than Horn-Reg$^I$, Horn-SHITQ and Horn-SROIQ, while still has PTIME data complexity.

In comparison with Horn-Reg$^I$ [33], Horn-DL additionally allows nominals, qualified number restrictions, the $\exists \text{Self}$ constructor, the universal role as well as assertions of the form Disj$(s, s')$, irreflexive$(s)$, $\neg s(a, b)$, $a \neq b$. 

3
In comparison with Horn-SROIQ, Horn-DL additionally allows the universal role and assertions of the form irrelexive(s), \( \neg s(a,b) \), \( a \neq b \). More importantly, like Horn-\( \mathcal{R} \mathcal{E} \)\(^1\), but in contrast to all the well-known Horn fragments \( EL \)  \[ 2, 3 \], DL-Lite  \[ 4 \], DLP  \[ 10 \], Horn-SHIQ  \[ 13 \], Horn-SROIQ  \[ 35 \] of DLs, Horn-DL allows the concept constructor \( \forall \mathcal{R} \mathcal{C} \) to appear at the left hand side of terminological inclusion axioms.

We develop a direct algorithm with PTIME data complexity for checking satisfiability of Horn-DL knowledge bases. It is based on constructing a graph using a global caching technique similar to the one of \[ 25, 29, 30, 34, 111, 113 \]. The keys for our algorithm are to follow the top-down approach of \[ 33 \] (instead of the bottom-up approach of \[ 30 \]) and use special techniques to deal with non-seriality, inverse roles and other concept constructors of Horn-DL.

1.4. The structure of this paper

The rest of this paper is structured as follows. In Section 2 we present syntax and semantics of Horn-DL. In Section 3 we give a detailed comparison between Horn-DL and Horn-SROIQ. In Section 4 we present our algorithm of checking satisfiability of Horn-DL knowledge bases, starting with a recall of automaton-modal operators. Section 5 contains examples illustrating the algorithm and Section 6 concludes this work.

2. Syntax and semantics of Horn-DL

Our language uses a finite set \( C \) of concept names, a finite set \( \mathcal{R}_+ \) of role names including a subset of simple role names, and a finite set \( I \) of individual names. We use letters like \( a, b \) to denote individual names, letters like \( A, B \) to denote concept names, and letters like \( r, s \) to denote role names.

For \( r \in \mathcal{R}_+ \), we call the expression \( \tau \) the inverse of \( r \). Let \( \mathcal{R}_r = \{ \tau \mid r \in \mathcal{R}_+ \} \) and \( \mathcal{R} = \mathcal{R}_+ \cup \mathcal{R}_r \). For \( R = \tau \), let \( \overline{R} \) stand for \( r \). We call elements of \( \mathcal{R} \) basic roles and use letters like \( R, S \) to denote them. We define a simple role to be either a simple role name or the inverse of a simple role name.

2.1. Syntax of RBoxes

A role inclusion axiom (RIA for short) is an expression of the form \( S_1 \circ \cdots \circ S_k \sqsubseteq R \), where \( k \geq 0 \). In the case \( k = 0 \), the left hand side of the inclusion axiom stands for the identity binary relation \( \varepsilon \). A role assertion is an expression of the form \( \text{Irr}(S) \) or \( \text{Disj}(S, S') \), where \( S \) and \( S' \) are simple roles, \( \text{Irr} \) stands for “irreflexive”, and \( \text{Disj} \) stands for “disjoint”.

A context-free semi-Thue system \( S \) over \( R \) is a finite set of context-free production rules over alphabet \( R \). It is symmetric if, for every rule \( R \to S_1 \ldots S_k \) of \( S \), the rule \( \overline{R} \to \overline{S_k} \ldots \overline{S_1} \) is also in \( S \)\(^2\). It is regular if, for every \( R \in \mathcal{R} \), the set of words derivable from \( R \) using the system is a regular language over \( R \).

A context-free semi-Thue system is like a context-free grammar, but it has no designated start symbol and there is no distinction between terminal and non-terminal symbols. We assume that, for \( R \in \mathcal{R} \), the word \( R \) is derivable from \( R \) using such a system.

A regular box of RIAs is a finite set \( \mathcal{R}_h \) of RIAs such that
\[
\{ R \to S_1 \ldots S_k \mid (S_1 \circ \cdots \circ S_k \sqsubseteq R) \in \mathcal{R}_h \}
\]
is a symmetric regular semi-Thue system \( S \) over \( R \) such that if \( R \in \mathcal{R} \) is a simple role then only words with length 1 or 0 are derivable from \( R \) using \( S \). We assume that \( \mathcal{R}_h \) is given together with a mapping \( \mathcal{A} \) that associates every \( R \in \mathcal{R} \) with a finite automaton \( \mathcal{A}_R \) recognizing the words derivable from \( R \) using \( S \). We call that \( \mathcal{A} \) the RIA-automaton-specification of \( \mathcal{R}_h \).

A regular RBox is a set \( \mathcal{R} = \mathcal{R}_a \cup \mathcal{R}_h \), where \( \mathcal{R}_a \) is a finite set of role assertions and \( \mathcal{R}_h \) is a regular box of RIAs.

Let \( \mathcal{R} \) be a regular RBox and \( \mathcal{A} \) be the RIA-automaton-specification of \( \mathcal{R}_h \). For \( R, S \in \mathcal{R} \), we say that \( R \) is a subrole of \( S \) w.r.t. \( \mathcal{R} \), denoted by \( R \sqsubseteq_{\mathcal{R}} S \), if the word \( R \) is accepted by \( \mathcal{A}_S \).

\(^1\)As will be seen later, only the universal role is not basic and only simple roles are allowed in number restrictions.

\(^2\)In the case \( k = 0 \), the right hand side of the production rules stand for the empty word, also denoted by \( \varepsilon \).
Recall that a finite automaton $A$ over alphabet $R$ is a tuple $(R, Q, q_0, \delta, F)$, where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, $\delta \subseteq Q \times R \times Q$ is the transition relation, and $F \subseteq Q$ is the set of accepting states. A run of $A$ on a word $R_1 \ldots R_k$ over alphabet $R$ is a finite sequence of states $q_0, q_1, \ldots, q_k$ such that $\delta(q_{i-1}, R_i, q_i)$ holds for every $1 \leq i \leq k$. It is an accepting run if $q_k \in F$. We say that $A$ accepts a word $w$ if there exists an accepting run of $A$ on $w$.

**Example 2.1.** Let $R_+ = \{\text{link}, \text{path}\}$ and let $R_h$ consist of the following RIAs:

\[
\begin{align*}
\text{link} & \subseteq \text{path}, \\
\text{link} \circ \text{path} & \subseteq \text{path}, \\
\text{path} \circ \text{link} & \subseteq \text{path}.
\end{align*}
\]

The set $R_h$ is a regular box of RIAs. Its RIA-automaton-specification $A$ is specified by:

\[
\begin{align*}
A_{\text{link}} &= (R, \{0, 1\}, 0, \{0, \text{link}, 1\}, \{1\}) \\
A_{\text{link}} &= (R, \{0, 1\}, 1, \{1, \text{link}, 0\}, \{0\}) \\
A_{\text{path}} &= (R, \{0, 1\}, 0, \{0, \text{link}, 0\}, \{0, \text{link}, 1\}, \{1\}) \\
A_{\text{path}} &= (R, \{0, 1\}, 1, \{0, \text{link}, 0\}, \{1, \text{link}, 0\}, \{1, \text{path}, 0\}, \{0\}).
\end{align*}
\]

**Example 2.2.** DLs are variants of modal logics. Individuals in DLs are like states in Kripke models, and roles in DLs are like accessibility relations in Kripke models. Let $R_+ = \{B, G, I\}$, where $B$, $G$ and $I$ stand for the accessibility relations of belief, goal and intention, respectively. Positive and negative introspection of these modalities w.r.t. belief as well as the axiom “intention implies goal” correspond to the following RIAs [10]:

\[
\begin{align*}
B \circ B & \subseteq B, \\
\overline{B} \circ B & \subseteq \overline{B}, \\
B \circ G & \subseteq G, \\
\overline{B} \circ B & \subseteq \overline{G}, \\
B \circ I & \subseteq I, \\
\overline{B} \circ B & \subseteq \overline{I}, \\
G & \subseteq I, \\
\overline{G} & \subseteq I
\end{align*}
\]

The set consisting of the above RIAs is a regular box of RIAs, because the sets of words derivable from the symbols $B$, $G$, $I$ using the corresponding semi-Thue system are characterized, respectively, by the following regular expressions:

\[
\begin{align*}
\bullet (B \cup \overline{B})^* \ast B \\
\bullet (B \cup \overline{B})^* \ast G \\
\bullet (B \cup \overline{B})^* \ast (I \cup G).
\end{align*}
\]

\[\square\]

2.2. **Syntax of concepts and TBoxes**

Concepts are defined by the following BNF grammar, where $A \in C$, $R \in R$, $a \in I$, $n$ is a natural number and $S$ is a simple role:

\[
C ::=} \top | \bot | A | \neg C | C \sqcap C | C \sqcup C | \forall R.C | \exists R.C | \{a\} | \forall U.C \mid \exists U.C \mid \exists S.\text{Self} \mid \geq n \ S.C \mid \leq n \ S.C
\]

The symbol $U \not\in R$ is called the universal role.

We use letters like $C$, $D$ to denote concepts.

Let $\forall R.C$ stand for $\forall R.C \sqcap \exists R.C$. **Left-hand-side (Horn-DL) concepts**, called **LHS concepts** for short, are defined by the following BNF grammar, where $A \in C$, $R \in R$, $a \in I$, and $S$ is a simple role:

\[
C ::= \top | A | C \sqcap C | C \sqcup C | \forall R.C | \exists R.C | \{a\} | \forall U.C | \exists U.C | \exists S.\text{Self}
\]
Right-hand-side (Horn-DL) concepts, called RHS concepts for short, are defined by the following BNF grammar, where \( A \in C, D \) is an LHS concept, \( R \in \mathbf{R} \), \( a \in \mathbf{I} \) and \( S \) is a simple role:

\[
C \ ::= \top | \bot | A | \neg D | C \cap C | \neg D \cup C | \forall R.C | \exists R.C | \exists R.S.C | \{a\} | \forall U.C | \exists U.C | \exists R.Self | \geq n.S.C | \leq 1.S.D
\]

A (Horn-DL) terminological axiom, also called a TBox (inclusion) axiom, is an expression of the form \( C \subseteq D \), where \( C \) is an LHS concept and \( D \) is an RHS concept.

A (Horn-DL) TBox is a finite set of Horn-DL terminological axioms.

A Horn-DL clause is a TBox axiom of the form \( C_1 \land \ldots \land C_k \subseteq D \) or \( \top \subseteq D \), where:

- each \( C_i \) is of the form \( A, \forall R.A, \exists R.A, \{a\}, \forall U.A, \exists U.A \) or \( \exists s.Self \),
- \( D \) is of the form \( \top, A, \forall R.A, \exists R.A, \{a\}, \forall U.A, \exists U.A, \exists r.Self, \geq n.S.A \) or \( \leq 1.S.A \),
- \( k \geq 1, A \in C, R \in \mathbf{R}, a \in \mathbf{I}, r \in \mathbf{R}^+, s \) is a simple role name and \( S \) is a simple role.

A clausal (Horn-DL) TBox is a TBox consisting of Horn-DL clauses.

2.3. Syntax of ABoxes and knowledge bases

A (Horn-DL) ABox is a finite set of assertions of the form \( C(a), r(a,b), \neg s(a,b), a \equiv b \) or \( a \neq b \), where \( C \) is an RHS concept, \( r \in \mathbf{R}^+ \) and \( s \) is a simple role name.

A reduced (Horn-DL) ABox is a finite set of assertions of the form \( A(a), \neg A(a), r(a,b), \neg s(a,b) \) or \( a \neq b \), where \( r \in \mathbf{R}^+ \) and \( s \) is a simple role name.

A (Horn-DL) knowledge base is a tuple \( \langle R, T, A \rangle \), where \( R \) is a regular RBox, \( T \) is a Horn-DL TBox and \( A \) is a Horn-DL ABox. When \( T \) is a clausal Horn-DL TBox and \( A \) is a reduced Horn-DL ABox, we call such a knowledge base a clausal (Horn-DL) knowledge base.

Example 2.3. Let

\[
C = \{\text{Male, Female, Mother, Father, Parent, Grandparent, OnlyChild}, \text{ParentWithOnlyChild, ParentWithOnlySons, ParentWithOnlyDaughters}\}
\]

\[
R_+ = \{\text{hasChild, hasParent, hasDescendant}\}
\]

\[
I = \{\text{Alice, Bob, Confucius, Dave, Ellie, Frank, George}\},
\]

where hasChild is a simple role name, and let

\[
A = \{\text{Male(Bob), Male(Confucius), Male(Dave), Male(Frank), Male(George), Female(Alice), Female(Ellie), hasChild(Alice, Dave), hasChild(Alice, Ellie), hasChild(Bob, Dave), hasChild(Bob, Ellie), hasChild(Dave, Frank), hasChild(Ellie, George), OnlyChild(George), ParentWithOnlyChild(Dave)}\} \cup \{x \neq y \mid x, y \in I, x \neq y\}
\]

\[
R = \{\text{hasChild} \sqsubseteq \text{hasParent}, \text{hasChild} \sqsubseteq \text{hasParent}, \text{hasChild} \sqsubseteq \text{hasDescendant}, \text{hasChild} \circ \text{hasDescendant} \sqsubseteq \text{hasDescendant}, \text{hasChild} \circ \text{hasDescendant} \sqsubseteq \text{hasDescendant}\}
\]

and let \( T \) be the TBox consisting of the following axioms:

\[
\exists \text{hasChild}.\top \sqsubseteq \text{Parent} \quad (4)
\]

\[
\exists \text{hasChild}.\text{Parent} \sqsubseteq \text{Grandparent} \quad (5)
\]
The ABox \( A \) is illustrated in Figure 1, where the edges denote instances of the role \( \text{hasChild} \). Observe that \( \text{KB} = \langle \mathcal{R}, \mathcal{T}, A \rangle \) is a Horn-DL knowledge base. Treating \( \text{hasChild} \) as a “primitive” role and \( \text{Male}, \text{Female} \) as “primitive” concepts, using the approach of logic programming, the RBox \( \mathcal{R} \) “defines” the roles \( \text{hasParent} \) and \( \text{hasDescendant} \), while the TBox axioms (4)–(10) “define” the concepts \( \text{Parent}, \text{Grandparent}, \text{Father}, \text{Mother}, \text{ParentWithOnlySons}, \text{ParentWithOnlyDaughters} \) and \( \text{DescendantOfConfucius} \). That is, \( \text{hasParent} \) is “defined” to be the inverse of \( \text{hasChild} \), \( \text{hasDescendant} \) is “defined” to be the transitive closure of \( \text{hasChild} \), and \( \sqsubseteq \) in the axioms (4)–(10) can be treated as equivalence. The axiom (11) is not a definition of \( \text{OnlyChild} \). It states only a necessary condition but not a sufficient condition for being an instance of \( \text{OnlyChild} \). The problem is that the converse of (11) is not allowed in Horn-DL. The axioms (12) and (13) define \( \text{ParentWithOnlyChild} \) to be equivalent to \( \exists \text{hasChild}. \text{OnlyChild} \). We use (12) as a constraint about \( \text{OnlyChild} \). The axiom (14) is a constraint stating that \( \text{Male} \) and \( \text{Female} \) are “disjoint” concepts.

2.4. Semantics

An interpretation is a pair \( \mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle \), where \( \Delta^\mathcal{I} \) is a non-empty set called the domain of \( \mathcal{I} \) and \( \cdot^\mathcal{I} \) is a mapping called the interpretation function of \( \mathcal{I} \) that associates each individual \( a \in \mathcal{I} \) with an element \( a^\mathcal{I} \in \Delta^\mathcal{I} \), each concept name \( A \in \mathcal{C} \) with a set \( A^\mathcal{I} \subseteq \Delta^\mathcal{I} \), and each role name \( r \in \mathcal{R}_+ \) with a binary relation \( r^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I} \). Define \( e^\mathcal{I} = \{ (x, x) \mid x \in \Delta^\mathcal{I} \} \), \( U^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I} \), and \( (\forall r^\mathcal{I})^{-1} = \{ (y, x) \mid (x, y) \in r^\mathcal{I} \} \) for \( r \in \mathcal{R}_+ \). The interpretation function \( \cdot^\mathcal{I} \) is extended to complex concepts as shown in Figure 2.

Note that, \( (\forall U.C)^\mathcal{I} = \Delta^\mathcal{I} \) or \( (\forall U.C)^\mathcal{I} = \emptyset \), and \( (\exists U.C)^\mathcal{I} = \Delta^\mathcal{I} \) or \( (\exists U.C)^\mathcal{I} = \emptyset \).
\( \top^\mathcal{I} = \Delta^\mathcal{I}, \ \bot^\mathcal{I} = \emptyset, \ \{a\}^\mathcal{I} = \{a^\mathcal{I}\}, \ \langle \exists R. \text{Self} \rangle^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid (x, x) \in R^\mathcal{I}\}, \)
\((-C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}, \ \langle C \cap D \rangle^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}, \ \langle C \cup D \rangle^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}, \)
\((\forall R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \forall y((x, y) \in R^\mathcal{I} \Rightarrow y \in C^\mathcal{I})\}, \)
\((\exists R.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \exists y((x, y) \in R^\mathcal{I} \land y \in C^\mathcal{I})\}, \)
\((\geq n \ S.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \exists\{y \mid (x, y) \in S^\mathcal{I} \land y \in C^\mathcal{I}\} \geq n\}, \)
\((\leq n \ S.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \exists\{y \mid (x, y) \in S^\mathcal{I} \land y \in C^\mathcal{I}\} \leq n\}, \)
\((\forall U.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid \Delta^\mathcal{I} \subseteq C^\mathcal{I}\}, \ \langle \exists U.C \rangle^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid C^\mathcal{I} \neq \emptyset\} . \)

Figure 2: Semantics of complex concepts.

Given an interpretation \( \mathcal{I} \) and an axiom/assertion \( \varphi \), the satisfaction relation \( \mathcal{I} \models \varphi \) is defined as follows:

\[ \mathcal{I} \models \text{lr}(S) \quad \text{if} \quad S^\mathcal{I} \text{ is irreflexive} \]
\[ \mathcal{I} \models \text{Disj}(S, S') \quad \text{if} \quad S^\mathcal{I} \text{ and } S'^\mathcal{I} \text{ are disjoint} \]
\[ \mathcal{I} \models S_1 \circ \cdots \circ S_k \subseteq R \quad \text{if} \quad S_1^\mathcal{I} \circ \cdots \circ S_k^\mathcal{I} \subseteq R^\mathcal{I} \]
\[ \mathcal{I} \models \varepsilon \subseteq R \quad \text{if} \quad \varepsilon^\mathcal{I} \subseteq R^\mathcal{I} \]
\[ \mathcal{I} \models C \subseteq D \quad \text{if} \quad C^\mathcal{I} \subseteq D^\mathcal{I} \]
\[ \mathcal{I} \models C(a) \quad \text{if} \quad a^\mathcal{I} \in C^\mathcal{I} \]
\[ \mathcal{I} \models r(a, b) \quad \text{if} \quad \langle a^\mathcal{I}, b^\mathcal{I} \rangle \in r^\mathcal{I} \]
\[ \mathcal{I} \models \neg s(a, b) \quad \text{if} \quad \langle a^\mathcal{I}, b^\mathcal{I} \rangle \notin s^\mathcal{I} \]
\[ \mathcal{I} \models a \neq b \quad \text{if} \quad a^\mathcal{I} \neq b^\mathcal{I} . \]

If \( \mathcal{I} \models \varphi \) then we say that \( \mathcal{I} \) validates \( \varphi \).

An interpretation \( \mathcal{I} \) is a model of an RBox \( \mathcal{R} \), a TBox \( \mathcal{T} \) or an ABox \( \mathcal{A} \) if it validates all the axioms/assertions of that “box”. It is a model of a knowledge base \( \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle \) if it is a model of all \( \mathcal{R}, \mathcal{T} \) and \( \mathcal{A} \).

A knowledge base is satisfiable if it has a model. For a knowledge base \( KB \), we write \( KB \models \varphi \) to mean that every model of \( KB \) validates \( \varphi \). If \( KB \models C(a) \) then we say that \( a \) is an instance of \( C \) w.r.t. \( KB \).

**Example 2.4.** Let us continue Example 2.3. One can ask, for example:

- whether Alice is an instance of the concept DescendantOfConfucius w.r.t. KB,
- whether Bob is an instance of the concept Grandparent w.r.t. KB,
- whether Ellie is an instance of the concept ParentWithOnlySons w.r.t. KB.

To answer any of these questions, one can extend \( KB \) with the negation of the question and check whether the extended knowledge base is unsatisfiable. The answer is “no” for the first question and “yes” for the two remaining questions.

We justify the answer for the first question. Let \( A' = A \cup \{ \neg \text{DescendantOfConfucius}(Alice) \} \) and \( KB' = \langle R, T, A' \rangle \). This extended knowledge base has a model \( \mathcal{I} = \langle \Delta^\mathcal{I}, r^\mathcal{I} \rangle \) specified by:

- \( \Delta^\mathcal{I} = \mathbb{I}, \ \text{Male}^\mathcal{I} = \{x \mid \text{Male}(x) \in A\}, \ \text{Female}^\mathcal{I} = \{x \mid \text{Female}(x) \in A\}, \)
- \( \text{hasChild}^\mathcal{I} = \{ (x, y) \mid \text{hasChild}(x, y) \in A \}, \ \text{hasParent}^\mathcal{I} = (\text{hasChild}^\mathcal{I})^{-1}, \)
- \( \text{hasDescendant}^\mathcal{I} = (\text{hasChild}^\mathcal{I})^+ \) (the transitive closure of \( \text{hasChild}^\mathcal{I} \)),

where \( R, T, A \) are the signature of the knowledge base.
- Parent\(^J\) = \{Alice, Bob, Dave, Ellie\}, Father\(^J\) = \{Bob, Dave\}, Mother\(^J\) = \{Alice, Ellie\},
- OnlyChild\(^J\) = \{Frank, George\}, ParentWithOnlyChild\(^J\) = \{Dave, Ellie\},
- ParentWithOnlySons\(^J\) = \{Dave, Ellie\}, ParentWithOnlyDaughters\(^J\) = \{\},
- Grandparent\(^J\) = \{Alice, Bob\}, DescendantOfConfucius\(^J\) = \{\}.

To make things more interesting consider the Horn-DL knowledge base \(KB_2 = \langle R, T_2, A \rangle\), where

\[ T_2 = T \cup \{ T \sqsubseteq (\exists \text{hasParent}.\text{Father} \sqcap \exists \text{hasParent}.\text{Mother}) \}. \]

The additional axiom states that everyone has a father and a mother. To answer the question whether \(KB_2 \models \text{DescendantOfConfucius}(Alice)\) one can proceed in a similar way as before. Let \(A_2^J = A \cup \{ \neg \text{DescendantOfConfucius}(Alice) \}\) and \(KB_2' = \langle R, T_2, A_2^J \rangle\). This extended knowledge base is satisfiable. Every “natural” model of \(KB_2\) should be infinite (unless one assumes that there were the first people in the world). We give here a finite model \(J = \langle \Delta^J, \cdot^J \rangle\) of \(KB_2'\), which is specified by:

- \(\Delta^J = I \cup \{ Adam, Eve \} \)
- \(\text{Male}^J = \{ Adam \} \cup \{ x \mid \text{Male}(x) \in A \}\), \(\text{Female}^J = \{ Eve \} \cup \{ x \mid \text{Female}(x) \in A \}\),
- hasChild\(^J\) = \\{(x, y) \mid \text{hasChild}(x, y) \in A\} \cup \{(Adam, x), (Eve, x) \mid x \in \{Adam, Eve, Confucius, Alice, Ellie\}\},
- hasParent\(^J\) = (\text{hasChild}^J)^{-1}, \text{hasDescendant}^J = (\text{hasChild}^J)^+,
- Parent\(^J\) = \{ Adam, Eve, Alice, Bob, Dave, Ellie \},
- Father\(^J\) = \{ Adam, Bob, Dave \}, Mother\(^J\) = \{ Eve, Alice, Ellie \},
- OnlyChild\(^J\) = \{ Frank, George \}, ParentWithOnlyChild\(^J\) = \{ Dave, Ellie \},
- ParentWithOnlySons\(^J\) = \{ Dave, Ellie \}, ParentWithOnlyDaughters\(^J\) = \{\},
- Grandparent\(^J\) = \{ Adam, Eve, Alice, Bob \}, DescendantOfConfucius\(^J\) = \{\}.

2.5. Data complexity

The size of a concept, an assertion or an axiom \(\varphi\) is defined to be the number of bits needed to encode \(\varphi\) in the usual way. The size of an ABox is the sum of the sizes of its assertions. The size of a TBox is the sum of the sizes of its axioms.

The data complexity class of Horn-DL is defined to be the complexity class of the problem of checking satisfiability of a Horn-DL knowledge base \(\langle R, T, A \rangle\), measured in the size of \(A\) when assuming that \(R\) and \(T\) are fixed and \(A\) is a reduced Horn-DL ABox.

**Proposition 2.1.** Let \(KB = \langle R, T, A \rangle\) be a Horn-DL knowledge base.

1. If \(C\) is an LHS concept then \(KB \models C(a)\) iff the Horn-DL knowledge base \(\langle R, T \cup \{ C \subseteq A \}, A \cup \{ \neg A(a) \}\rangle\) is unsatisfiable, where \(A\) is a fresh concept name.
2. \(KB\) can be converted in polynomial time in the sizes of \(T\) and \(A\) to a Horn-DL knowledge base \(KB' = \langle R, T', A' \rangle\) with \(A'\) being a reduced Horn-DL ABox such that \(KB\) is satisfiable iff \(KB'\) is satisfiable.
3. \(KB\) can be converted in polynomial time in the size of \(T\) to a Horn-DL knowledge base \(KB' = \langle R, T', A \rangle\) with \(T'\) being a clausal Horn-DL TBox such that \(KB\) is satisfiable iff \(KB'\) is satisfiable.

**Proof.** The first assertion is clear. For the second assertion, we start with \(T' := T\) and \(A' := A\) and then modify them as follows:
• for each \( C(a) \in \mathcal{A}' \) where \( C \) is not a concept name, replace \( C(a) \) in \( \mathcal{A}' \) by \( A(a) \), where \( A \) is a fresh concept name, and add to \( \mathcal{T}' \) the axiom \( A \sqsubseteq C \),

• for each \( (a \neq b) \in \mathcal{A}' \), delete that assertion from \( \mathcal{A}' \) and then replace every occurrence of \( b \) in \( \mathcal{T}' \) and \( \mathcal{A}' \) by \( a \).

It is easy to check that the resulting Horn-DL knowledge base \( KB' = \langle \mathcal{R}, \mathcal{T}', \mathcal{A}' \rangle \) is satisfiable iff \( KB \) is satisfiable.

For the third assertion, we apply the technique that replaces complex concepts by fresh concept names. For example, if \( \forall \exists R.C \sqsubseteq \exists S.D \) is an axiom of \( \mathcal{T} \), where \( C \) and \( D \) are complex concepts, then we replace it by axioms \( C \sqsubseteq A_C, \exists \exists R.A \sqsubseteq \exists S.A_D \) and \( A_D \sqsubseteq D \), where \( A_C \) and \( A_D \) are fresh concept names. Furthermore, every \( \exists \exists . \text{Self} \) is replaced by \( \exists r . \text{Self} \).

\[\text{Corollary 2.2.} \text{ Every Horn-DL knowledge base } KB = \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle \text{ can be converted in polynomial time in the sizes of } \mathcal{T} \text{ and } \mathcal{A} \text{ to a clausal Horn-DL knowledge base } KB' = \langle \mathcal{R}, \mathcal{T}', \mathcal{A}' \rangle \text{ such that } KB \text{ is satisfiable iff } KB' \text{ is satisfiable.}\]

3. Comparing Horn-DL with Horn-SROIQ

We first recall the syntax of Horn-SROIQ \cite{35}:

• An RBox in Horn-SROIQ consists of role assertions \( \text{Disj}(S, S') \), where \( S \) and \( S' \) are simple roles, and RIAs of the following forms, where \( k \geq 1 \) and \( R, R_1, \ldots, R_k \) are roles strictly less than \( r \) in a fixed partial order:

\[
\begin{align*}
\mathcal{T} & \sqsubseteq r & R_1 \circ \cdots \circ R_k & \sqsubseteq r \\
R & \sqsubseteq r & r \circ R_1 \circ \cdots \circ R_k & \sqsubseteq r \\
r \circ r & \sqsubseteq r & R_1 \circ \cdots \circ R_k \circ r & \sqsubseteq r.
\end{align*}
\]

• A TBox in Horn-SROIQ consists of axioms of the following forms, where \( C, C', C'' \) are concepts of the form \( A, \{a\} \) or \( \exists S. \text{Self} \), and \( S \) is a simple role:

\[
\begin{align*}
C \sqcap C' & \sqsubseteq C'' & \exists R.C & \sqsubseteq C' \\
C & \sqsubseteq \forall R.C' & C & \sqsubseteq 1 S.C'' \\
C & \sqsubseteq \exists R.C' & C & \sqsubseteq n S.C''.
\end{align*}
\]

• An ABox in Horn-SROIQ consists of assertions of the form \( A(a) \) or \( \forall(b). \)

Comparing Horn-SROIQ with Horn-DL, observe that:

• Every set of RIAs in Horn-SROIQ is a regular box of RIAs, but the converse does not hold. For example, the regular box of RIAs given in Example 2.2 is not a set of RIAs in Horn-SROIQ (because \( B \) and \( B \) are “dependent” on each other). Roughly speaking using the notion of regular expressions, “regularity” of RIAs in Horn-SROIQ allows only a bounded nesting depth of the star operator \( * \), while “regularity” of RIAs in Horn-DL is not so restricted. Furthermore, reflexivity of a role is expressible in our case, but not in the case of Horn-SROIQ.

• In contrast to Horn-DL, Horn-SROIQ does not allow the universal role and assertions of the form \( \text{Uniq}(S), \neg s(a, b), a \neq b \). Also, in Horn-SROIQ a concept \( \exists R. \text{Self} \) can occur at the right hand side of \( \sqsubseteq \) only when \( R \) is a simple role, while in Horn-DL such \( R \) can be an arbitrary role.

• The most remarkable difference is that, in contrast to Horn-DL, Horn-SROIQ does not allow the concept constructor \( \forall R.C \) in the left hand side of TBox inclusion axioms.

Therefore, every RBox (resp. TBox or ABox) in Horn-SROIQ is also a (regular) RBox (resp. TBox or ABox) in Horn-DL, but not vice versa. Thus, Horn-DL is strictly and essentially richer than Horn-SROIQ.\footnote{It can be proved that, when fixing the sets \( C \) and \( R \), extending Horn-SROIQ only by allowing the concept constructor \( \forall R.C \) in the left hand side of TBox inclusion axioms already increases its expressiveness. This can be done by using bisimulation as in \cite{3}.}
Checking satisfiability of Horn-DL knowledge bases

In this section we present an algorithm that, given a clausal Horn-DL knowledge base \( \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle \) together with the RIA-automaton-specification \( \mathcal{A} \) of \( \mathcal{R}_h \), checks whether the knowledge base is satisfiable.

4.1. Automaton-modal operators

Given an interpretation \( \mathcal{I} \) and a finite automaton \( \mathcal{A} \) over alphabet \( \mathcal{R} \), define \( \mathcal{I}^x = \{(x, y) \in \Delta^x \times \Delta^y \mid \text{there exist a word } R_1 \ldots R_k \text{ accepted by } \mathcal{A} \text{ and elements } x_0 = x, x_1, \ldots, x_k = y \text{ of } \Delta^x \text{ such that } \langle x_{i-1}, x_i \rangle \in R_i^x \text{ for all } 1 \leq i \leq k\} \).

We will use auxiliary modal operators \( [\mathcal{A}] \) and \( (\mathcal{A}) \), where \( \mathcal{A} \) is a finite automaton over alphabet \( \mathcal{R} \). We call \( [\mathcal{A}] \) (resp. \( (\mathcal{A}) \)) a universal (resp. existential) automaton-modal operator. Such operators were used earlier, among others, in [15, 29, 11, 30].

In the extended language, if \( C \) is a concept then \( [\mathcal{A}]C \) and \( (\mathcal{A})C \) are also concepts. The semantics of \( [\mathcal{A}]C \) and \( (\mathcal{A})C \) in an interpretation \( \mathcal{I} \) are defined as follows:

\[
\begin{align*}
([\mathcal{A}]C)^x &= \{ x \in \Delta^x \mid \forall y (\langle x, y \rangle \in \mathcal{A}^x \text{ implies } y \in C^y) \} \\
((\mathcal{A})C)^x &= \{ x \in \Delta^x \mid \exists y (\langle x, y \rangle \in \mathcal{A}^x \text{ and } y \in C^y) \}.
\end{align*}
\]

For a finite automaton \( \mathcal{A} \) over \( \mathcal{R} \), assume that \( \mathcal{A} = \langle \mathcal{R}, Q, q_0, \delta, q_A, F_A \rangle \).

If \( q \) is a state of a finite automaton \( \mathcal{A} \) then by \( A_q \) we denote the finite automaton obtained from \( \mathcal{A} \) by replacing the initial state by \( q \).

Lemma 4.1. Let \( \mathcal{I} \) be a model of a regular box \( \mathcal{R}_h \) of RIA\#s, \( \mathcal{A} \) be the RIA-automaton-specification of \( \mathcal{R}_h \), \( C \) be a concept, and \( R \in \mathcal{R} \). Then \( \forall R.C)^x = ([A_R]C)^x \), \( (\exists R.C)^x = ((A_R)C)^x \), \( C^x \subseteq ([A_{\overline{R}}](A_R)C)^x \) and \( C^x \subseteq ([A_{\overline{\Delta}}]\exists R.C)^x \).

The proof of this lemma is straightforward.

4.2. The data structure and auxiliary notions

For convenience, we also refer to TBox axioms as concepts and treat a TBox axiom \( C \subseteq D \) as the concept \( \neg C \cup D \) (which stands for a global requirement for all elements in the domain). Observe that, for any interpretation \( \mathcal{I} \), we have \( \mathcal{I} \models C \subseteq D \text{ iff } (\neg C \cup D)^x = \Delta^x \).

Let \( \text{EdgeLabels} \) be the power set of \( \{ \exists R.A, \exists R.\top \mid R \in \mathcal{R}, A \in \mathcal{C} \} \). For \( E \in \text{EdgeLabels} \), we define \( \text{Roles}(E) = \{ R \in \mathcal{R} \mid \text{E contains } R \text{ or some } \exists R.C \} \) and call \( \overline{E} = \{ \overline{R} \mid R \in \text{Roles}(E) \} \) the inverse of \( E \).

Our algorithm for checking satisfiability of \( \langle \mathcal{R}, \mathcal{T}, \mathcal{A} \rangle \) uses a data structure

\[
G = \langle \mathcal{R}', \mathcal{T}', \mathcal{A}', \Delta_0, \Delta_1, \Delta_2, \Delta, \text{Label}, \text{Succ}, \text{Next}_U, \text{Replacement}, \text{Status} \rangle,
\]

called a Horn-DL graph, where:

- \( \mathcal{R}', \mathcal{T}', \mathcal{A}' \) are modified versions of \( \mathcal{R}, \mathcal{T}, \mathcal{A} \), respectively,
- \( \Delta \) is a non-empty set of nodes, \( \Delta_0 \subseteq \Delta_1 \subseteq \Delta, \Delta_2 \subseteq \Delta \) and \( \Delta_2 \cap \Delta_0 = \emptyset \),
- \( \text{Label} \) is a mapping that associates each \( x \in \Delta \) with a set of concepts,
- \( \text{Succ} : \Delta \times \text{EdgeLabels} \to \Delta \) is a partial mapping,
- \( \text{Next}_U : \{ \exists U.\top, \exists U.A \mid A \in \mathcal{C} \} \to \Delta \) is a partial mapping,
- \( \text{Replacement} : \mathcal{I} \to \mathcal{I} \) is a mapping,
- \( \text{Status} \in \{ \text{unknown, unsat, sat} \} \).
Informally, $\Delta_0$ consists of all individual names occurring in $\mathcal{A}'$ and $\mathcal{T}'$, $\Delta_1 \setminus \Delta_0$ consists of the nodes created for realizing “requirements” of the form $\exists U.C$, and $\Delta_2$ consists of so called unnamed noncloneable nodes. For a node $x \in \Delta$, $Label(x)$ is called the label of $x$. If $Succ(x, E) = y$ then $y$ is called a successor of $x$, and $x$ is called a predecessor of $y$. If $Next_U(\exists U.A) = y$ then $A \in Label(y)$ and the “requirement” $\exists U.A$ at any $x$ is “realized” by using $y$. When defined, $Next_U(\exists U.T)$ stands for the “logically smallest object”. If $Replacement(a) = b$ then $a \neq b$ follows from the knowledge base and $b$ is the representative of its abstract class of the equivalence relation $\simeq$. A fact $Status = unsat$ (resp. $Status = sat$) means the knowledge base is unsatisfiable (resp. satisfiable).

The partial mapping $Succ$ has the property that if $Succ(x, \{\exists R_1.C_1, \ldots, \exists R_k.C_k, S_1, \ldots, S_h\}) = y$ then $\{\exists R_1.C_1, \ldots, \exists R_k.C_k\} \subseteq Label(x)$, $\{C_1, \ldots, C_k\} \subseteq Label(y)$, and $x$ is “connected” to $y$ by all the roles $R_1, \ldots, R_k$, $S_1, \ldots, S_h$. Furthermore, 

if both $Succ(x, E)$ and $Succ(x, E')$ are defined then $E \cap E' \subseteq R$. \hfill (15)

That is, for each $\exists R.C$ with $C \in C \cup \{\top\}$, there exists at most one $E \in EdgeLabels$ such that $\exists R.C \in E$ and $Succ(x, E)$ is defined.

We say that $x \in \Delta$ is reachable from $\Delta_1$ if there exist $x_0, \ldots, x_k \in \Delta$ and $E_1, \ldots, E_k$ such that $x_0 \in \Delta_1$, $k \geq 0$, $x_k = x$ and $Succ(x_{i-1}, E_i) = x_i$ for all $1 \leq i \leq k$.

Let $Next : \Delta \times \{\exists R.A, \exists R.\top \mid R \in R, A \in C\} \rightarrow \Delta$ denote the partial mapping defined by: $Next(x, \exists R.C) = y$ if there exists $E \in EdgeLabels$ with $\exists R.C \in E$ and $Succ(x, E) = y$. By the assumption [15], $Next$ is well defined. The meaning of $Next(x, \exists R.C) = y$ is that $\exists R.C \in Label(x)$, $C \subseteq Label(y)$ and the “requirement” $\exists R.C$ is realized at $x$ by going to $y$. If $Next(x, \exists R.\top) = y$ then $y$ stands for the “logically smallest $R$-neighbor” of $x$.

Define $NamedEdges$ to be the smallest subset of $\Delta_0 \times R \times \Delta_0$ such that:

- if $r(a, b) \in \mathcal{A}'$ then $\langle a, r, b \rangle \in NamedEdges$,
- if $Succ(a, E) = b$ and $R \in Roles(E)$ then $\langle a, R, b \rangle \in NamedEdges$,
- if $\exists R.\text{Self} \in Label(a)$ then $\langle a, R, a \rangle \in NamedEdges$,
- if $\langle a, R, b \rangle \in NamedEdges$ and $R \sqsubseteq R S$ then $\langle a, S, b \rangle \in NamedEdges$,
- if $\langle a, R, b \rangle \in NamedEdges$ then $\langle b, \overline{R}, a \rangle \in NamedEdges$.

Let $X$ be a set of concepts. The saturation of $X$ (w.r.t. $\mathcal{A}$ and $\mathcal{T}$), denoted by $\text{Satr}(X)$, is defined to be the least extension of $X$ such that:

- for every $R \in R$, if $A_R$ accepts $e$ then $\exists R.\text{Self} \in \text{Satr}(X)$,
- if $\exists R.\text{Self} \in \text{Satr}(X)$ then $\overline{R}.\text{Self} \in \text{Satr}(X)$,
- if $\exists R.\text{Self} \in \text{Satr}(X)$ and $R \sqsubseteq R S$ then $\exists S.\text{Self} \in \text{Satr}(X)$,
- if $\exists S.A \in \text{Satr}(X)$, where $n \geq 1$, then $\exists S.\text{A} \in \text{Satr}(X)$,
- if $\forall R.A \in \text{Satr}(X)$ and $R \in R$ then $[A_R]A \in \text{Satr}(X)$,
- if $\forall U.A \in \text{Satr}(X)$ then $A \in \text{Satr}(X)$,
- if $[A[C \in \text{Satr}(X)$ and $q_a \in F_a$ then $C \in \text{Satr}(X)$,
- if $[[A[C, \exists R.\text{Self}] \subseteq \text{Satr}(X)$ and $\langle q_a, R, q \rangle \in \delta_a$ then $[A_q]C \in \text{Satr}(X)$,
- for every $R \in R$, if $\forall \exists R.A$ occurs in $\mathcal{T}$ for some $A$ then $[A_{\overline{R}}]\exists R.\top \in \text{Satr}(X)$,
• for every $R \in \mathbf{R}$, if $A \in \text{Satr}(X)$ and $\exists R.A$ occurs in the left hand side of $\subseteq$ in some clause of $T$ then $[A_T](A_R)A \in \text{Satr}(X)$.

For $E \in \text{EdgeLabels}$, the transfer of $X$ through $E$ is defined to be

$$\text{Trans}(X, E) = \{[A_q]C \mid [A]C \in X, (q_A, R, q) \in \delta_A, R \in \text{Roles}(E)\}.$$  

4.3. A natural approach

How can we try to construct a model for $(\mathbf{R}, T, \mathbf{A})$? Let us first describe a natural approach without the termination property. (In the next subsection we will revise the approach to form an algorithm with the termination property.) The intended model is based on a Horn-DL graph, which is initialized by Procedure InitializeGraph (on page 13), with $\mathbf{A'} := \mathbf{A}$ and $T' := T$. Each individual $a$ occurring in $\mathbf{A}$ or $T$ is a node of the initial graph, with the label consisting of $\{a\}$, all the concepts $A$ such that $A(a) \in \mathbf{A}$, and all the TBox axioms of $T$. During the construction, for each node $x$ of the graph, we treat $\text{Label}(x)$ as a set of requirements to be realized at $x$. We realize a requirement $\varphi \in \text{Label}(x)$ as follows:

• Case $\varphi = \exists R.C$: We connect $x$ to a new node $y$ with $\text{Label}(y) = \{C\} \cup T'$ using an edge labeled by $\{\exists R.C\}$ by setting $\text{Succ}(x, \{\exists R.C\}) := y$.

• Case $\varphi = \exists U.C$: If $\text{Next}(U, \{\exists U.C\})$ is not defined then we create a new node $y$ with $\text{Label}(y) = \{C\} \cup T'$ and set $\text{Next}(U, \{\exists U.C\}) := y$.

• Case $\varphi = \forall R.A$: We include $[A_R]A$ in $\text{Label}(x)$ by saturating $\text{Label}(x)$. Since $\forall R.A$ is semantically equivalent to $[A_R]A$, we realize the requirement $[A_R]A$ (instead of $\forall R.A$) at $x$ as in the following case.

• Case $\varphi = [A]C$:
  - If $\text{Succ}(x, E) = y$ then we extend $\text{Label}(y)$ with $\text{Trans}(\text{Label}(x), E)$.
  - If $\text{Succ}(y, E) = x$ then we extend $\text{Label}(y)$ with $\text{Trans}(\text{Label}(x), E)$.
  - If $(x, R, y)$ or $(y, \mathbf{T}, x)$ belongs to $\text{NamedEdges}$ then we extend $\text{Label}(y)$ with $\text{Trans}(\text{Label}(x), \{R\})$.
  - If $\exists R.\text{Self} \in \text{Label}(x)$ then we extend $\text{Label}(x)$ with $\text{Trans}(\text{Label}(x), \{R\})$.
  - If $q_A \in \text{Label}(x)$ then we extend $\text{Label}(x)$ with $C$. The last two operations are done by saturating $\text{Label}(x)$. Notice that some changes are done set-at-a-time, not depending on a concrete concept $[A]C$.

• Case $\varphi = \forall U.A$: We extend the label of every node of the graph with $A$.

• Case $\varphi = \{a\}$: If $a \neq x$ then we merge $a$ and $x$ in an appropriate way (if $x$ is also a named individual then we may make appropriate changes for $\Delta_0$, $T'$, $A'$, $\text{Label}$, $\text{Succ}$, $\text{Next}$ and $\text{Replacement}$).

• Case $\varphi = \exists \text{Self}$: The requirement is implicitly realized when dealing with requirements of the form $[A]C$ and by using saturation.

• Case $\varphi = (\geq n \text{S.A})$: Dismissing the constraint just for this discussion, similarly to the case of $\exists R.C$, we connect $x$ to new nodes $y_1, \ldots, y_n$ with label $\{A\} \cup T'$ using edges labeled by $\{\exists S.A\}$.

• Case $\varphi = (\leq 1 \text{S.A})$: If $y$ and $y'$ are different “$S$-neighbors” of $x$ whose labels contain $A$ then we “merge” them together. This will be discussed in more detail later. Note that the complex form of edge labels is due to realization of requirements of the form $\leq S.A$, when edge labels are merged.

• Case $\varphi = (C \subseteq D)$: If $C$ “holds” at $x$ then we extend $\text{Label}(x)$ with $D$. Suppose $C = C_1 \cap \ldots \cap C_k$. This premise “holds” at $x$ if $C_i$ “holds” at $x$ for each $1 \leq i \leq k$. There are the following cases:
  - Case $C_i = A$: $C_i$ holds at $x$ if $A \in \text{Label}(x)$. 

13
- Case $C_i = \{a\}$: $C_i$ holds at $x$ if $x = a$.
- Case $C_i = \exists x.Self$: $C_i$ holds at $x$ if either $x \notin \Delta_0$ and there exists $\exists R.Self \in \text{Label}(x)$ with $R \subseteq R$ and $(x, s, x) \in \text{NamedEdges}$.
- Case $C_i = \forall x.R.A$: $C_i$ “holds” at $x$ if both $\forall x.R.A$ and $\exists R.\top$ “hold” at $x$. If $\exists R.\top$ “holds” at $x$ by the evidence of a path connecting $x$ to a node $z$ with (forward or backward) edges labeled by $E_1, \ldots, E_k$ such that $S_j \in \text{Roles}(E_j)$ for each $1 \leq j \leq k$ and the word $S_1 \ldots S_k$ is accepted by $A_R$ then, since $A_R[\exists R.\top]$ is included in $\text{Label}(z)$ by saturation, we can expect that $\exists R.\top \in \text{Label}(x)$ and $\text{Next}(x, \exists R.\top)$ is defined. To check whether $C_i$ “holds” at $x$ we just check whether $\exists R.\top \in \text{Label}(x)$, $\text{Next}(x, \exists R.\top)$ is defined and $A \in \text{Label}(\text{Next}(x, \exists R.\top))$. The intuition is that, $y = \text{Next}(x, \exists R.\top)$ is the “least $R$-neighbor” of $x$, and if $A \in \text{Label}(y)$ then $A$ will occur in all “$R$-neighbors” of $x$. The technique of using $\text{Next}(x, \exists R.\top)$ comes from \cite{27}.
- Case $C_i = \exists x.R.A$: If $\exists x.R.A$ “holds” at $x$ by the evidence of a path connecting $x$ to a node $z$ with (forward or backward) edges labeled by $E_1, \ldots, E_k$ such that $S_j \in \text{Roles}(E_j)$ for each $1 \leq j \leq k$ and the word $S_1 \ldots S_k$ is accepted by $A_R$ and $A \in \text{Label}(z)$ then, since $A_R[\exists x.R.A]$ is included in $\text{Label}(z)$ by saturation, we can expect that $A_R[\exists x.R.A] \in \text{Label}(x)$. To check whether $C_i = \exists x.R.A$ “holds” at $x$, we just check whether $A_R[\exists x.R.A] \in \text{Label}(x)$. (Semantically, $A_R[\exists x.R.A] = \exists x.R.A$ is equivalent to $\exists x.R.A$.)
- Case $C_i = \forall U.A$: $C_i$ “holds” at $x$ if $A \in \text{Label}(\text{Next}_U(\exists U.\top))$. The intuition is that $\text{Next}_U(\exists U.\top)$ stands for the “logically smallest node”.
- Case $C_i = \exists U.A$: $C_i$ “holds” at $x$ if there exists a node whose label contains $A$.

- Case $\varphi = \bot$: We set $\text{Status} := \text{unsat}$ and terminate the process with the conclusion that the considered knowledge base is unsatisfiable.
- Case $\varphi = \neg A$: If $A \in \text{Label}(x)$ then we also set $\text{Status} := \text{unsat}$ and terminate the process.

4.4. Global caching

The graph constructed in the previous subsection may be infinite. What we want to check is whether at some step the attribute $\text{Status}$ receives value $\text{unsat}$. To guarantee the termination property we apply a global caching technique, which is similar to the one of \cite{28,29,30,34,11}.

Roughly speaking, if two nodes that are not named individuals have the same label, then we merge them together. In other words, for every finite set $X$ of concepts, the graph contains at most one node $z \in \Delta \setminus \Delta_0$ such that $\text{Label}(z) = X$. The function $\text{Find}(X)$ (on page \cite{15}) returns such a node $z$ if it exists, or creates such a node $z$ otherwise. Global caching, however, causes some problems, which are discussed and solved below.

Suppose we want to extend the label of $z \in \Delta$ with a set $X$ of concepts.

- Case $z \in \Delta_0$ (i.e., $z$ is a named individual occurring in $A$): as $z$ is “fixed” by the ABox $A$, we have no choice but to extend $\text{Label}(z)$ directly with $X$ and the concepts resulting from saturation.
- Case $z \not\in \Delta_0$ and the requirements $X$ are directly caused by $z$ itself or its successors: if we directly extend the label of $z$ (with $X$) then $z$ may have the same label as another node not belonging to $\Delta_0$ and global caching is not fulfilled. Hence, we simulate changing the label of $z$ by using $z_* = \text{Find}(\text{Sat}(\text{Label}(z) \cup X))$ for playing the role of $z$. In particular: for each $y$ and $E$ such that $\text{Succ}(y, E) = z$, we set $\text{Succ}(y, E) := z_*$; for each $C$ such that $\text{Next}_U(\exists U.C) = z$, we set $\text{Next}_U(\exists U.C) := z_*$; and if $z \in \Delta_1$ then we replace $z$ in $\Delta_1$ by $z_*$.

\footnote{If an edge is labeled by $E$ then its inverse is an edge labeled by $E$.}
Extending the label of \( z \) for the above two cases is done by Procedure \texttt{ExtendLabel}(z, X) (on page 15). The third case is considered below.

Suppose \( \text{Succ}(x, E) = y \). Then, to realize the requirements at \( x \), the label of \( y \) should be extended with \( X = \text{Trans}(\text{Label}(x), E) \). How can we realize such an extension? There may exist another \( \text{Succ}(x', E') = y \)
Algorithm 1: checking satisfiability of a clausal Horn-DL knowledge base

Input: a clausal Horn-DL knowledge base \( \langle R, T, A \rangle \) and the RIA-automaton-specification \( A \) of \( R_h \).

Output: true if \( \langle R, T, A \rangle \) is satisfiable, or false otherwise.

Data structure: \( \langle R', T', A', \Delta_0, \Delta_1, \Delta_2, \Delta, Label, Succ, Next_U, Replacement, Status \rangle \).

1. InitializeGraph;
2. while some rule in Tables 3 and 4 can change the graph do
   3. choose such a rule, where \( (O) \) has the highest priority, and execute it; /* any strategy can be used */
   4. if Status = unsat then return false;
5. return true;

\[
\text{Table 2: Expansion rules for Horn-DL graphs - Part 1}
\]

with \( x' \neq x \). That is, by global caching, we may use \( y \) as a successor for two different nodes \( x \) and \( x' \). If we directly modify the label of \( y \) to realize the requirements of \( x \), such a modification may affect \( x' \) unjustifiably. The solution is to delete the edge connecting \( x \) to \( y \) via \( E \) and reconnect \( x \) to \( y_* = \text{Find}(\text{Satr}(\text{Label}(y) \cup X)) \) by setting \( \text{Succ}(x, E) := y_* \).

Another problem is that, when \( x \notin \Delta_0 \), \( \text{Succ}(x, E) = x \) and \( s \in \text{Roles}(E) \) may not imply that \( \exists s. \text{Self} \) “holds” at \( x \). We may have \( \text{Succ}(x, E) = x \) just by global caching. As a solution, for \( x \notin \Delta_0 \), we explicitly keep in \( \text{Label}(x) \) all concepts of the form \( \exists s. \text{Self} \) that should “hold” at \( x \). This is done by saturating \( \text{Label}(x) \) and appropriate treatments when executing mergings forced by requirements of the form \( \leq 1 S.A. \)

When realizing a requirement of the form \( \geq n S.A \) at a node \( x \), as all the “\( S \)-successors” of \( x \) created for the realization would be cached by using the same node, instead of \( n \) “\( S \)-successors” we create only one “\( S \)-successor” of \( x \) for that purpose. This is done by saturating \( \text{Label}(x) \) with \( \exists S.A. \). The constraint \( \{15\} \) is thus resumed. For checking possible inconsistency caused by requirements of the form \( \geq n S.A. \), the rules \( \{(1)\} \) and \( \{(1)\} \) in Table 3 are used.

Reconsider realization of a requirement \( \leq 1 S.A \) at a node \( x \). As mentioned earlier, the approach is: if \( y \) and \( y' \) are different “\( S \)-neighbors” of \( x \) whose labels contain \( A \) then we “merge” them together. A simple
(Q₁) if \((≤ 1 \text{S}.A \in \text{Label}(x), \{(x, R, y), (x, R', y')\} \subseteq \text{NamedEdges}, y \neq y', R \sqsubseteq_R S, R' \sqsubseteq_R S\), then \text{ExtendLabel}(y, \{y\}); // this will merge \(y'\) to \(y\)

(Q₂) if \((≤ 1 \text{S}.A \in \text{Label}(x), (x, R, y) \in \text{NamedEdges}, y' = \text{Succ}(x, E') \notin \Delta_0, R' \in \text{Roles}(E'), R \sqsubseteq_R S, R' \sqsubseteq_R S\), then \text{Succ}(x, E') := y, \text{ExtendLabel}(y, \text{Label}(y));

(Q₃) if \((≤ 1 \text{S}.A \in \text{Label}(x), (x, R, y) \in \text{NamedEdges}, x = \text{Succ}(y', E'), y' \notin \Delta_0, y' \text{ is reachable from } \Delta_1, R' \in \text{Roles}(E'), R \sqsubseteq_R S, \overline{R'} \sqsubseteq_R S\), then \text{ExtendLabel}(y', \{y\}); // this will simulate merging \(y'\) to \(y\)

(Q₄) if \(x \notin \Delta_0\) is reachable from \(\Delta_1\), \(≤ 1 \text{S}.A, \exists R'.\text{Self} \subseteq \text{Label}(x)\), \(y = \text{Succ}(x, E), y \neq x, R \in \text{Roles}(E), R \sqsubseteq_R S, R' \sqsubseteq_R S\), then \text{ExtendLabel}(y, \text{Label}(y));

(Q₅) if \(x \notin \Delta_0\), \(≤ 1 \text{S}.A, \exists R'.\text{Self} \subseteq \text{Label}(x)\), \(x = \text{Succ}(y, E), x \neq y, R \in \text{Roles}(E), R \sqsubseteq_R S, R' \sqsubseteq_R S\), then \text{ExtendLabel}(y, \text{Label}(y));

(Q₆) if \(x \in \text{Label}(x)\), \(E \neq E', \text{Succ}(x, E) = y, \text{Succ}(x, E') = y', R \in \text{Roles}(E), R' \in \text{Roles}(E')\), then \text{Succ}(x, E \cup E') := \text{Find}(\text{Satr}(y \cup \text{Label}(y)'));

(Q₇) if \(x \notin \Delta_0\), \(≤ 1 \text{S}.A \subseteq \text{Label}(x)\), \(x \in \text{Label}(x)\), \(y \notin \Delta_1, \text{Succ}(y, E) = x, \text{Succ}(x, E') = y', R \in \text{Roles}(E), R' \in \text{Roles}(E')\), then \text{ExtendLabel}(y', \text{Label}(y'))

(Q₈) if \(x \in \Delta_0 \cup \Delta_2\), \(≤ 1 \text{S}.A \subseteq \text{Label}(x)\), then \text{ExtendLabel}(y, \text{Label}(y));

(Q₉) if \(x \in \Delta_0 \cup \Delta_2\), \(≤ 1 \text{S}.A \subseteq \text{Label}(x)\), then \text{ExtendLabel}(y', \text{Label}(y));

(Q₁₀) if \(n \geq 2, \text{Succ}(x, E) = y, \exists S.A \in E \text{ and } y \notin \Delta_0 \cup \Delta_2\), then \text{Status} := \text{unsat};

(Q₁₁) if \(n \geq 2\), \(≤ 1 \text{S}.A \subseteq \text{Label}(x)\), then \text{Status} := \text{unsat};

Table 3: Expansion rules for Horn-DL graphs - Part II

17
Procedure Merge($x, a$)

Input: $x \in \Delta$ and $a \in \Delta_0$ s.t. $x$ is reachable from $\Delta_1$, \{a\} $\in$ Label($x$) and $x \neq a$.

1. if $x \in \Delta_0$ then $a_0 := x$;
2. else $a_0 := a$;
3. let $a_1, \ldots, a_k$ be all the individuals $\neq a_0$ s.t. \{\$a_i\$\} $\in$ Label($x$) for $1 \leq i \leq k$;

   // merge all $a_1, \ldots, a_k$ to $a_0$
4. foreach $b$ such that Replacement($b$) $\in$ \{\$a_1, \ldots, a_k\$\} do Replacement($b$) := $a$;
5. foreach $1 \leq i \leq k$ do
6.   Replacement($a_i$) := $a_0$, Label($a_0$) := Satr(Label($a_0$) $\cup$ Label($a_i$));
7. foreach $y$ and $E$ such that Succ($y, E$) = $a_i$ do Succ($y, E$) := $a_0$;
8. foreach $C$ such that Next$_{\cup C}$($\exists U.C.$) = $a_i$ do Next$_{\cup C}$($\exists U.C.$) := $a_0$;
9. replace all occurrences of $a_1, \ldots, a_k$ in $\mathcal{A}'$ and $\mathcal{T}'$ by $a_0$;
10. foreach $x \in \Delta$ do replace all occurrences of $a_1, \ldots, a_k$ in Label($x$) by $a_0$;

CompactGraph;
12. delete the nodes $a_1, \ldots, a_k$ from $\Delta_0, \Delta_1, \Delta$ and delete the tuples of $\text{Label, Succ, Next}_{\cup C}$ that are directly involved with those nodes;
13. if $x \notin \Delta_0$ then // merge $x$ to $a_0$
14.   Label($a_0$) := Satr(Label($a_0$) $\cup$ Label($x$));
15.   foreach $y$ and $E$ such that Succ($y, E$) = $x'$ do Succ($y, E$) := $a_0$;
16.   foreach $C$ such that Next$_{\cup C}$($\exists U.C.$) = $x'$ do Next$_{\cup C}$($\exists U.C.$) := $a_0$;
17.   if $x \in \Delta_1$ then $\Delta_1$ := $\Delta_1 \setminus \{x\}$;

Procedure CompactGraph

1. foreach $x, x' \in \Delta \setminus \Delta_0$ reachable from $\Delta_1$ such that Label($x$) = Label($x'$) do
2.   // merge $x'$ to $x$
3.   foreach $y$ and $E$ such that Succ($y, E$) = $x'$ do Succ($y, E$) := $x$;
4.   foreach $C$ such that Next$_{\cup C}$($\exists U.C.$) = $x'$ do Next$_{\cup C}$($\exists U.C.$) := $x$;
5.   if $x' \in \Delta_1$ then add $x$ to $\Delta_1$;
6.   delete $x'$ from $\Delta$ and $\Delta_1$ (if $x' \in \Delta_2$) and delete the tuples of $\text{Label, Succ, Next}_{\cup C}$ that are directly involved with $x'$;

The case when Succ($x, E$) = $y$, Succ($x, E'$) = $y'$, $E \neq E'$, $R \in \text{Roles}(E)$, $R' \in \text{Roles}(E')$, $R \sqsubseteq_R S$, $R' \sqsubseteq_R S$, $A \in \text{Label}(y)$ and $A \in \text{Label}(y')$ is solved by undefining Succ($x, E$) and Succ($x, E'$) and setting Succ($x, E \cup E'$) := Find(Satr(Label($y$) $\cup$ Label($y'$))). This is done by the rule (Q8) in Table 3. The rules (Q1)–(Q5), (Q7), (Q8) in Table 3 deal with the other cases. To see that the rules (Q1)–(Q8) cover all of possible cases, observe that:

- that $y$ and $y'$ are “$S$-neighbors” of $x$ is directly related to NamedEdges, Self, Succ or “inverse of Succ”;
- by definition, a condition $\langle x, R, y \rangle \in \text{NamedEdges}$ holds in a number of situations, e.g., when $x \in \Delta_0$, $y = x$ and $\exists R, \text{Self} \in \text{Label}(x)$;
- the case when $x \notin \Delta_0 \cup \Delta_2$, ($\leq 1 \text{ S.A.} \in \text{Label}(x), x = \text{Succ}(y, E) = \text{Succ}(y', E'), R \in \text{Roles}(E)$, $R' \in \text{Roles}(E')$, $R \sqsubseteq_R S$, $R' \sqsubseteq_R S$, $A \in \text{Label}(y)$ and $A \in \text{Label}(y')$ does not require any treatment, because it occurs just as a side effect of global caching and $x \notin \Delta_0 \cup \Delta_2$ means $x$ is cloneable (which allows to use one copy of $x$ for $y$ and another one for $y'$).

Let us explain the special rule (Q7) (the other rules among (Q1)–(Q8) deal with simpler cases in the usual way). Suppose $x \notin \Delta_0$, ($\leq 1 \text{ S.A.} \in \text{Label}(x), \text{Succ}(y, E) = x$, $\text{Succ}(x, E') = y'$, $R \in \text{Roles}(E)$,
$R' \in \text{Roles}(E')$, $\overline{R} \sqsubseteq R \sqsubseteq R$, $A \in \text{Label}(y)$ and $A \in \text{Label}(y')$. With global caching, we cannot merge nodes in the usual way. We cannot merge $y$ and $y'$ together nor simulate merging $y$ to $y'$ because $y'$ may be used as a successor of other nodes than $x$. What we can do is to keep $y'$ (for the sake of the other nodes that have $y'$ as a successor) and simulate merging $y'$ to $y$ (for the sake of $y$). The simulation is done as follows:

- If $\text{Label}(y') \not\subseteq \text{Label}(y)$ then we execute $\text{ExtendLabel}(y, \text{Label}(y'))$, which simulates changing $y$ by using a replacement $y_*$, and what further should be done is a matter of $y_*$.

- Else, with $E_2 = E \cup \{ \overline{R'} \mid R' \in \text{Roles}(E') \}$ and $X = \text{Trans}(\text{Label}(y), E_2) \cup \{ \forall S.B \mid B \in \text{Label}(y), B \text{ occurs in } T \}$, if $E \not\subseteq E_2$ or $X \not\subseteq \text{Label}(x)$ then we undefine $\text{Succ}(y, E)$ and set $\text{Succ}(y, E_2) := \text{Find}(\text{Satr}(\text{Label}(x) \cup X))$. This can be understood as follows: merging $y'$ to $y$ would add the inverses of the roles of $\text{Roles}(E')$ to $E$ and add the concepts of $X$ to $\text{Label}(x)$ (concepts $\forall S.B$, where $B \in \text{Label}(y)$ and $B$ occurs in $T$, can be added to $\text{Label}(x)$ because $y$ would be the only “$S$-neighbor” of $x$). But we do not want to modify $\text{Label}(x)$, so such a change is only simulated.

- In the remaining case, we do nothing. The intuition is that when constructing a model for the considered knowledge base and following the paths going through $y$ and $x$ we will simply ignore the edge specified by $\text{Succ}(x, E') = y'$. All the conditions of this case guarantee that ignoring the edge specified by $\text{Succ}(x, E') = y'$ does not cause any problem (for checking premises of TBox axioms at $y$ by using $\text{CheckPremise}$). In particular, $\text{Next}(x, \exists S.T)$ may be $y'$, but the presence of all the concepts $\forall S.B$ in $\text{Label}(x)$ where $B \in \text{Label}(y)$ and $B$ occurs in $T$ makes sure that checking whether a condition $\forall S.B$ “holds” at $x$ is done correctly.

The rule $(Q_9)$ in Table $3$ is used to update the set of “unnamed nonclonable nodes”: if $x \in \Delta_0 \cup \Delta_2$, $(\leq 1.S.A) \in \text{Label}(x)$, $x = \text{Succ}(y, E)$, $R \in \text{Roles}(E)$, $\overline{R} \sqsubseteq R \sqsubseteq S$, $y \notin \Delta_0 \cup \Delta_2$ and $A \in \text{Label}(y)$ then we add $y$ to $\Delta_2$.

When realizing requirements that are nominals, we may have to replace a named individual by another one, which may cause some nodes in $\Delta \setminus \Delta_0$ to have the same label. In that case, we compact the graph by Procedure $\text{CompactGraph}$ (on page 18).

Algorithm $1$ (on page 16) is our formal algorithm for checking satisfiability of a clausal Horn-DL knowledge base. It initializes a Horn-DL graph and expands it by using the rules in Tables $2$ and $3$. It returns $\text{false}$ (unsatisfiable) when the attribute $\text{Status}$ receives value $\text{unsat}$. When the graph cannot be expanded anymore, the answer true (satisfiable) is returned.

When expanding the graph, we need to pay attention only to the nodes reachable from $\Delta_1$. The rules $(\perp_1)$–$(\perp_5)$ are used to detect inconsistency caused by ABox assertions and role assertions. To make these rules simple, we expand $R$ to $R'$ as in Procedure $\text{InitializeGraph}$ and define the saturation operator $\text{Satr}$ appropriately (for dealing with $\text{Self}$). Apart from the mentioned procedures/functions, our algorithm also uses Function $\text{CheckPremise}$ (on page 15) and Procedure $\text{Merge}$ (on page 18). Notice that all the expansion rules in Table $3$ are devoted to number restrictions, and Procedures $\text{Merge}$ and $\text{CompactGraph}$ are devoted to nominals. In the case without number restrictions and nominals, our algorithm would be much simpler.

**Theorem 4.2.** Algorithm $1$ correctly checks satisfiability of clausal Horn-DL knowledge bases and has $\text{PTime}$ data complexity.

This theorem follows from Lemmas $\text{Appendix A.1}$ $\text{Appendix A.2}$ and Corollary $\text{Appendix A.3}$ which are presented and proved in the appendix. The following corollary immediately follows from Theorem $4.2$ and Proposition $2.1$.

**Corollary 4.3.** The problem of checking satisfiability of Horn-DL knowledge bases has $\text{PTime}$ data complexity.

5. **Illustrative examples**

This section contains examples illustrating Algorithm $1$. They show our techniques of dealing with automaton-modal operators, inverse roles and the concept constructor $\forall R.C$. For simplicity, the examples
Example 5.1. Let $R_+ = \{r, s\}$, $C = \{A, B, C, D, E\}$, $I = \{a, b, c\}$,

$$\mathcal{R} = \{r \circ r \sqsubseteq r, r \circ r \subseteq s, \tau \circ r \subseteq \tau, r \circ r \subseteq \tau\},$$

and let $T$ be the TBox consisting of the following axioms:

$$A \sqsubseteq \exists r.C \quad (16)$$
$$C \sqsubseteq \forall \tau.D \quad (17)$$
$$D \sqsubseteq C \quad (18)$$
$$A \cap \exists \exists r.C \sqsubseteq E \quad (19)$$
$$A \cap \exists s.B \sqsubseteq E. \quad (20)$$

Let the RIA-automaton-specification $A$ of $\mathcal{R}$ be specified by:

$$A_r = (\mathcal{R}, \{0, 1\}, 0, \{(0, r, 1), (1, r, 1)\}, \{1\})$$
$$A_r = (\mathcal{R}, \{0, 1\}, 1, \{(1, \tau, 0), (1, \tau, 1)\}, \{0\})$$
$$A_s = (\mathcal{R}, \{0, 1, 2, 3\}, 0, \{(0, s, 1), (0, r, 2), (2, r, 2), (2, s, 3), (1, \tau, 3), (3, \tau, 3)\}, \{1, 3\})$$
$$A_T = (\mathcal{R}, \{0, 1, 2, 3\}, 0, \{(0, s, 1), (0, r, 2), (2, r, 2), (2, s, 3), (1, \tau, 3), (3, \tau, 3)\}, \{1, 3\}).$$

Note that $A_r = (A_r)_0$, $A_r = (A_T)_1$, $A_s = (A_s)_0$ and $A_T = (A_T)_0$.

Consider the Horn-DL knowledge base $KB = (\mathcal{R}, T, A)$ with

$$A = \{A(b), B(b), A(c), \neg E(c), r(b, a), s(b, c)\}.$$
Consider the Horn-DL knowledge base $KB$. This example illustrates global caching. Let $\exists r.\top$, $\forall r.\top$, $C$, $A,B$, $A$, $\top$. Figure 3 illustrates the Horn-DL graph constructed by Algorithm 1 for $KB$. The steps of the run of Algorithm 1 for $KB_2$ are as follows:

1: Applying $\exists r.\top$ to the nodes $x = a$ and $y = b$.
2: Applying $\exists r.\top$ to the nodes $x = a$ and $y = b$.

As no more expansion rules are applicable to the graph and the Status of the graph is different from unsat (none of the rules $(\bot_1)\ldots(\bot_7)$ was applicable), by Theorem 4.2 the knowledge base $KB$ is satisfiable.

**Example 5.2.** This example illustrates global caching. Let $R_+, C$, $I$, $R$, $T$, $A$ be as in Example 5.1. Consider the Horn-DL knowledge base $KB_2 = (R, T, A_2)$ with

$$A_2 = \{A(a), B(a), A(b), r(a, b)\}.$$ 

Figure 4 illustrates the Horn-DL graph constructed by Algorithm 1 for $KB_2$ in a similar way as Figure 3 does for $KB$. The steps of the run of Algorithm 1 for $KB_2$ are as follows:

1: Applying $\forall r.\top$ to the nodes $x = a$ and $y = b$.
2: Applying $\forall r.\top$ to the nodes $x = a$ and $y = b$.
3: Applying $\exists r.\top$ to the node $x = a$ using the clause (16).
4: Applying $\exists r.\top$ to the node $x = a$ using the clause (20).
5: Applying (⊑) to the node $x = b$ using the clause [16].
6: Applying (∃) to the node $x = a$ and the concept $∃r.C$.
7: Applying (∃) to the node $x = b$ and the concept $∃r.C$.
8: Applying (⊑) to the node $x = u$ using the clause [17].
9: Applying (∀I) to the nodes $x = a$ and $y = u'$.
10: Applying (⊑) to the node $x = a$ using the clause [18].
11: Applying (⊑) to the node $x = a$ using the clause [17].
12: Applying (∀I) to the nodes $x = b$ and $y = u'$.
13: Applying (⊑) to the node $x = b$ using the clause [18].
14: Applying (⊑) to the node $x = b$ using the clause [17].
15: Applying (⊑) to the node $x = b$ using the clause [20].
16: Applying (∃) to the node $x = a$ and the concept $∃r.⊤$.
17: Applying (∃) to the node $x = b$ and the concept $∃r.⊤$.

As no more expansion rules are applicable to the graph and the Status of the graph is different from unsat (none of the rules $(⊥₁)-(⊥₇)$ was applicable), by Theorem 4.2, the knowledge base $KB₂$ is satisfiable.

**Example 5.3.** Let us continue Example 5.2. It can be seen from Figure 4 that $KB₂ \models E(b)$. To formally show this, let $A'_2 = A₂ \cup \{¬E(b)\}$ and consider the Horn-DL knowledge base $KB'_2 = \langle R, T, A'_2 \rangle$. There exists a short run of Algorithm 1 for $KB'_2$ as demonstrated in Figure 5. The steps are as follows:
0: Initialization.
1: Applying (\forall) to the nodes \(x = a\) and \(y = b\).
2: Applying (\sqsubseteq) to the node \(x = b\) using the clause (16).
3: Applying (\exists) to the node \(x = b\) and the concept \(\exists r.C\).
4: Applying (\forall I) to the nodes \(x = b\) and \(y = u\).
5: Applying (\sqsubseteq) to the node \(x = b\) using the clause (20).
6: Applying (\bot 1) to the node \(x = b\) to establish \(\text{Status} = \text{unsat}\).

Thus, by Theorem 4.2 the knowledge base \(KB_2^\prime\) is unsatisfiable, and hence \(KB_2 \models E(b)\).

6. Conclusion and discussion

The rule language Horn-DL developed by us for the Semantic Web has PTime data complexity and, as discussed in Sections 1.3 and 5, is strictly and essentially richer than Horn-Reg\[33\] and the well-known rule languages Horn-SHIF\[18\] and Horn-SROIQ\[35\]. We have developed the first algorithm with PTime data complexity for checking satisfiability of Horn-DL knowledge bases.

Our algorithm has been implemented in C++ by a student [19]. The implemented program receives as input an Horn-DL knowledge base \(KB\) that need not to be in the clausal form. It automatically translates \(KB\) to the clausal form. The program can check whether \(KB\) is satisfiable and answer queries of the form \(C(x)\) (i.e., whether \(x\) is an instance of \(C\) w.r.t. \(KB\)), where \(C\) is an LHS concept. Possible applications of Horn-DL are similar to applications of the profiles OWL 2 EL, OWL 2 QL and OWL 2 RL of OWL 2. An ontology in Horn-DL, possibly using an OWL-like syntax, can be used for the Semantic Web in the usual way as a knowledge base. Having PTime data complexity, Horn-DL is a query language whose processing is scalable w.r.t. the set of individual assertions (i.e. data). One can implement our algorithm and its naive extension for answering conjunctive queries\[6\] as Java API and incorporate them into the Jena framework. The naive extension just systematically instantiates variables in a conjunctive query by individual names. A direct implementation like [19] may be a bit time-consuming, but should not be a hard task. A real problem is to develop an efficient algorithm for processing conjunctive queries in Horn-DL. This remains as an open problem\[7\].

---

\[6\] A conjunctive query \(\varphi\) to a knowledge base \(KB\) is a conjunction of formulas of the form \(C(x)\) or \(R(x, y)\), where \(x\) and \(y\) are variables for individual names. An answer for the query is a substitution \(\theta\) that maps each variable to an individual name such that \(KB \models \varphi\).

\[7\] With respect to efficiency, our algorithm for Horn-DL is like a naive evaluation method of Datalog. It is expected to develop an advanced algorithm for Horn-DL like the one for Datalog that combines the magic-sets transformation with the improved semi-naive evaluation method.
Acknowledgments

This work was supported by the Polish National Science Centre (NCN) under Grants No. 2011/01/B/ST6/02759 and 2011/01/B/ST6/02769. We would like to thank the anonymous reviewers for helpful comments.

References


Appendix A. Proofs

Define $\text{closure}_A(T')$ to be the smallest set of concepts and TBox axioms such that:

- TBox axioms of $T'$ belong to $\text{closure}_A(T')$,
- concepts and subconcepts occurring in $T'$ belong to $\text{closure}_A(T')$,
- if $R \in R$ then $\{\top, \exists R, \text{Self}, [A_1][\exists R, \top]\} \subseteq \text{closure}_A(T')$,
- if $\forall R. C \in \text{closure}_A(T')$ then $[A_R]C \in \text{closure}_A(T')$,
- if $A \in \text{closure}_A(T')$ and $R \in R$ then $[A_R][A_R]A \in \text{closure}_A(T')$,
- if $[A][C] \in \text{closure}_A(T')$ and $q \in Q_A$ then $[A_q]C \in \text{closure}_A(T')$,
- if $\geq n.S.A \in \text{closure}_A(T')$ then $\exists S.A \in \text{closure}_A(T')$,
- if $S$ is a simple role and $A \in \text{closure}_A(T')$ then $\forall S.A \in \text{closure}_A(T')$.

Lemma Appendix A.1. Algorithm 4 runs in polynomial time in the size of $A$ (when assuming that $R$ and $T$ are fixed).

Proof. Let $n$ be the size of $A$. Since $R$ and $T$ are fixed, the size of $\text{closure}_A(T')$ is bounded by a constant. Observe that, at any step during an execution of Algorithm 4, for $x \in \Delta \setminus \Delta_0$, $\text{Label}(x) \subseteq \text{closure}_A(T')$, and for $a \in \Delta_0$, $\text{Label}(a) \setminus \{C : C(a) \in A\} \subseteq \text{closure}_A(T')$. Hence the sizes of these two sets are also bounded by a constant. Since each $x \in \Delta \setminus \Delta_0$ has a unique $\text{Label}(x) \subseteq \text{closure}_A(T')$, the set $\Delta \setminus \Delta_0$ contains only $O(1)$ elements. Hence, the size of $\Delta$ is of rank $O(n)$. If $\text{Succ}(x, E) = y$ then $E$ contains at least one concept of the form $\exists R.A$ or $\exists R, \top$, which depends only on $T$, but not $A$. By [15], for each $x \in \Delta$, the numbers of $E \in \text{EdgeLabels}$ such that $\text{Succ}(x, E)$ is defined is bounded by a constant. Consequently, it can be seen that each iteration of the main loop of Algorithm 4 runs in polynomial time in $n$.

Observe that each iteration of the main loop of Algorithm 4

- either extends the label of a node $x \in \Delta_0$,
- or adds a new node to $\Delta$,
- or creates a new “edge”,
- or combines edges to a “bigger” one,
or merges named individuals and hence reduces the number of named individuals occurring in the constructed graph.

Due to the mentioned “bounds” on labels of nodes, the number of nodes in ∆ \ ∆₀, and “edges”, it follows that the number of iterations the main loop of Algorithm 1 executes is bounded by a polynomial in n. Therefore, Algorithm 1 runs in polynomial time in n.

Lemma Appendix A.2. If Algorithm 4 returns true then the knowledge base (R, T, A) is satisfiable.

Proof. Suppose Algorithm 1 returns true for (R, T, A). We will refer to the data structures used by that run of Algorithm 1. A model for (R, T, A) will be constructed by starting from ∆₁, then “unfolding” the remaining part of the graph constructed by Algorithm 1 and then completing the interpretation of basic roles and named individuals. For that we define a counterpart ∆′ of ∆, a counterpart Next′ of Next, a set Edges ⊆ ∆′ × R × ∆′, a mapping f : ∆′ → ∆ and a queue unresolved of elements of ∆′ in Algorithm 2.

Algorithm 2: unfolding a Horn-DL graph for the proof of Lemma Appendix A.2

1. Δ′ := Δ₁, Edges := NamedEdges, unresolved := Δ′;
2. foreach x ∈ Δ₁ do f(x) := x;
3. while unresolved is not empty do
   4.   extract an element u from unresolved;
   5.   foreach E and y such that Succ(f(u), E) = y do
      6.     if f(u) /∈ Δ₀ and there exist (≤1 S, A) ∈ Label(f(u)), ⟨u′, R′, u⟩ ∈ Edges, R ∈ Roles(E) such that R ⊆ R S and R ⊇ R S then
         7.         foreach ∃R′′, C ∈ Roles(E) do Next′(u, ∃R′′, C) := u′;
      8.     else if y /∈ Δ₂ then
         9.         if y /∈ Δ′ then add y to Δ′ and unresolved and set f(y) := y;
        10.        foreach R ∈ Roles(E) do add ⟨u, R, y⟩ to Edges;
        11.        foreach ∃R, C ∈ Roles(E) do Next′(u, ∃R, C) := y;
      12.   else
         13.     if there exists (≥ n S, A) ∈ Label(f(u)) such that n ≥ 2 and ∃S, A ∈ E then
            14.       let m be the largest among such numbers n;
            15.       else let m = 1;
            16.       add new elements v₁, …, v_m to ∆′ and unresolved;
            17.       foreach 1 ≤ i ≤ m do
               18.          f(vᵢ) := y;
               19.          foreach R ∈ Roles(E) do add ⟨u, R, vᵢ⟩ to Edges;
               20.          foreach ∃R, C ∈ Roles(E) do Next′(u, ∃R, C) := vᵢ;
            21.     foreach ∃R, Self ∈ Label(f(u)) do add ⟨u, R, u⟩ to Edges;

Algorithm 2 does not have the termination property. What we want is that the “results” of Algorithm 2 are well defined although they may be infinite; and we have it.

Let I be the interpretation with Δ² = Δ′, specified by:

• for each a ∈ I, if a ∈ Δ₀ then a² = a else a² = Replacement(a);
• for each A ∈ C, A² = {u ∈ Δ′ | A ∈ Label(f(u))};
• for all R ∈ R, R² are the least relations satisfying the following conditions:
  - if ⟨u, R, v⟩ ∈ Edges then ⟨u, v⟩ ∈ R²,
  - for every word S₁…S_k accepted by A_R, S₁² o … o S_k² ⊆ R².

26
Note that the steps 6 and 7 of Algorithm 2 correspond to a case discussed earlier of the rule (Q7) when it does nothing. The steps 8–11 are related to the rule (Q9). The partial mapping Next’ has the property that: if \( \exists R.C \in Label(f(u)) \) then Next’(u, \( R.C \)) is defined to be a v such that \( \langle u, v \rangle \in R^2 \) and \( C \in Label(f(v)) \).

We show that \( I \) is a model of (\( R, T, A \)). As Procedure Merge is defined appropriately, it suffices to prove that, for every \( u \in \Delta' \) and every \( \varphi \in Label(f(u)), u \in \varphi^T \). We prove this by induction on the structure of \( \varphi \). Let \( u \in \Delta' \) and suppose \( \varphi \in Label(f(u)) \).

- Case \( \varphi = A \) is trivial.
- Case \( \varphi = \{a\} \): By the rule (O), we have that \( u \in \Delta_0 \) and \( u = a \). Hence \( u \in \varphi^T \).
- Case \( \varphi = \exists R.Self \): By the step 21 of Algorithm 2 and the definition of \( I, \langle u, u \rangle \in R^2 \). Hence \( u \in \varphi^T \).
- Case \( \varphi = (\leq 1.S.A) \): By the rules (Q4)–(Q8) and the steps 6–11 of Algorithm 2 we have that \( u \in \varphi^T \).
- Case \( \varphi = (\geq n.S.A) \): Since \( \varphi \in Label(f(u)) \) and \( Label(f(u)) \) is saturated, we also have that \( \exists S.A \in Label(f(u)) \). Hence, there exists \( E \in EdgeLabels \) such that \( \exists S.A \in E \) and \( \text{Succ}(f(u), E) \) is defined. By the steps 13–20 of Algorithm 2 it follows that \( u \in \varphi^T \).
- Case \( \varphi = \exists U.A \): For \( x = \text{Next}′(\exists U.A) \), we have that \( x \in \Delta_1 \subseteq \Delta' \) and \( A \in Label(x) \). Hence \( u \in \varphi^T \).
- Case \( \varphi = \forall U.A \): By the rule (\( \forall U \)), we have that \( u \in \varphi^T \).
- Case \( \varphi = \exists R.C \), where \( C \in C \cup \{\top\} \): Since \( \varphi \in Label(f(u)) \), with \( v = \text{Next}′(u, \exists R.C) \), we have that \( \langle u, v \rangle \in R^2 \) and \( C \in Label(f(v)) \). Hence \( u \in \varphi^T \).
- Case \( \varphi = \forall R.A \): Let \( v \) be any element of \( \Delta^T \) such that \( \langle u, v \rangle \in R^2 \). We show that \( v \in A^T \). Since \( \langle u, v \rangle \in R^2 \), there exist a word \( S_1 \ldots S_k \) accepted by \( A_R \) and elements \( u_0 = u, u_1, \ldots, u_{k-1}, u_k = v \) such that, for every \( 1 \leq i \leq k \), \( \langle u_{i-1}, u_i \rangle \in S_i^2 \) and \( \langle u_{i-1}, S_i, u_i \rangle \in Edges \) or \( \langle u_i, s, u_{i+1} \rangle \in Edges \). Let \( A = A_R \). Since \( S_1 \ldots S_k \) is accepted by \( A \), there exist states \( q_0 = q_k, q_1, \ldots, q_k \) such that \( q_k \in F_A \) and \( \langle q_{i-1}, S_i, q_i \rangle \in \delta_A \) for every \( 1 \leq i \leq k \). Since \( \varphi \in Label(f(u)) \) and \( \varphi = \forall R.A, \) by saturation, we have that \( [A]\{r\}A \in Label(f(u)) \), which means \( [A]A \in Label(f(u)) \) and \( [A_{q_0}]C \in Label(f(u_0)) \). For each \( i \) from 1 to \( k \), since \( \langle u_{i-1}, S_i, u_i \rangle \in Edges \) or \( \langle u_i, s, u_{i+1} \rangle \in Edges \), it follows that \( [A_{q_0}]A \in Label(f(u_i)) \). Since \( q_k \in F_A \) and \( u_k = v \), it follows that \( A \in Label(f(v)) \). Hence, by the inductive assumption, \( v \in A^T \).
- Case \( \varphi = (C \subseteq D) \) and \( C = C_1 \cap \ldots \cap C_k \): Suppose \( u \in C^2 \). We prove that \( u \in D^2 \). The last call CheckPremise(\( f(u), C \)) returned true because the following observations hold for every \( 1 \leq i \leq k \):
  - Case \( C_1 = A \): Since \( u \in C^2 \), we have that \( A \in Label(f(u)) \).
  - Case \( C_1 = \{a\} \): Since \( u \in C^2 \), we have that \( u = a \). Hence, \( u \in \Delta_0 \) and \( f(u) = u = a \).
  - Case \( C_1 = \exists s.Self \): Since \( u \in C^2 \), \( \langle u, u \rangle \in s^2 \). Observe that, due to the saturation operator and the definition of NamedEdges, \( \langle u, u \rangle \in s^2 \) happens only when:
    * \( u \notin \Delta_0 \) and \( \exists s.Self \in Label(f(u)) \), or
    * \( u \in \Delta_0 \) and \( \langle u, s, u \rangle \in NamedEdges \), which imply that \( f(u) = u \) and \( \langle f(u), s, f(u) \rangle \in NamedEdges \).
  - Case \( C_1 = \exists U.A \): Since \( u \in C^2 \), there exists \( v \in \Delta' \) such that \( A \in Label(f(v)) \). Clearly, \( y = f(v) \) is reachable from \( \Delta_1 \).
  - Case \( C_1 = \forall U.A \): For \( x = \text{Next}′(\exists U.\top) \), we have that \( x \in \Delta_1 \subseteq \Delta' \) and \( f(x) = x \). Since \( u \in C^2 \), we have that \( x \in A^2 \), hence \( A \in Label(f(x)) \), which means \( A \in Label(\text{Next}′(\exists U.\top)) \).
following corollary is a consequence of the last assertion of this lemma.

Lemma Appendix A.3. Let \( R, T, A \) be a clausal Horn-DL knowledge base. Suppose KB is satisfiable and \( \mathcal{I} \) is a model of KB. Consider an execution of Algorithm \( 2 \) for KB and a moment either before the main loop (i.e., after the initialization) or after an iteration of the main loop of Algorithm \( 2 \). Let \( r = \{ (x, u) \in \Delta \times \Delta^T \mid u \in \text{Label}(f(u)) \} \). Then:

1. \( \mathcal{I} \) is a model of \( \langle R', T', A' \rangle \).
2. for every \( a \in \Delta_0 \), \( (a, u) \in r \).
3. for every \( a, b \in 1 \) such that Replacement\( (a) = b \), we have \( a^T = b^T \).
4. for every \( x \in \Delta_0 \cup \Delta_2 \), there exists exactly one \( u \in \Delta^T \) such that \( (x, u) \in r \).
5. for every \( u \in \Delta^T \), \( \text{Next}_T(u, \text{A}) \) is defined. Let \( A \in \Delta^T \) and \( (a, u) \in r \).
6. for every \( A \in \text{Roles} \) and \( (A, u) \in r \).
7. for every \( x, y \in \Delta \), \( u, v \in \Delta \), \( E \in \text{EdgeLabels} \) and \( 3R.C \in E \) such that \( \text{Succ}(x, E) = y \), if \( (x, u) \in r \), \( (u, v) \in \Delta^T \), then \( (y, v) \in r \) and, for every \( S \in \text{Roles}(E) \), \( (u, v) \in \Delta^T \).
8. for every \( x \in \Delta \), there exists \( u \in \Delta^T \) such that \( (x, u) \in r \).

This lemma can be proved by induction on the number of executed steps in a straightforward way. The following corollary is a consequence of the last assertion of this lemma.

Corollary Appendix A.4. If KB is a satisfiable clausal Horn-DL knowledge base then Algorithm \( 3 \) returns true for it.