Concept Abduction in Description logics with cardinality restrictions

Viet-Hoang VU, Nhan LE-THANH
KEWI - Laboratory I3S - CNRS
Nice-Sophia Antipolis University, France

Abstract—Recently the usefulness of Concept Abduction, a novel non-monotonic inference service for Description Logics (DLs), has been argued in the context of ontology-based applications such as semantic matchmaking and resource retrieval. Based on tableau calculus, a method has been proposed to realize this reasoning task in $\mathcal{ALN}$, a description logic that supports simple cardinality restrictions as well as other basic constructors. However, in many ontology-based systems, the representation of ontology would require expressive formalisms for capturing domain-specific constraints, this language is not sufficient. In order to increase the applicability of the abductive reasoning method in such contexts, we would like to present in the scope of this paper an extension of the tableaux-based algorithm for dealing with concepts represented in $\mathcal{ALCQ}$, the description logic that extends $\mathcal{ALN}$ with full concept negation and quantified number restrictions.

Keywords—abductive reasoning, description logics, semantic matchmaking, non-monotonic inference, tableaux-based method.

I. INTRODUCTION

Description Logics (DLs) [1] are a family of knowledge representation formalisms that are used widely for building ontologies, especially in the context of Semantic Web. Descendants from earlier semantic networks and frame-based languages, but equipped with a logic-based semantics, these languages allow to represent the knowledge of an application domain in a well-structured and formal way. Besides, to better support the ontology engineering, description logics systems provides as well tools to reason about its knowledge bases efficiently. There are two kinds: standard reasoning services, mainly monotonic, like concept satisfiability or concept subsumption; and non-standard reasoning services, usually non-monotonic, such as least-common subsumer or concept approximation.

Abduction is a popular technique of common sense reasoning[12] to find back explanations from observations. Despite it was well studied in the context of classical logics, just recently this inference method have been begun to gain the attention of the description logics community because of its usefulness to deal with inferences that new ontology-based applications would require [7], [6], [11]. In one of such contexts [11], concept abduction has been proposed as a novel inference service for $\mathcal{ALN}$, the description logic that, in addition to basic constructors like conjunction of concepts ($\wedge$) and value restriction on role ($\forall$), supports unqualified number restrictions. Derived from propositional abduction, the non-monotonic reasoning method allows to, given two non-disjoint concepts $C$, $D$ and a TBox $\mathcal{T}$, find a hypothesis $H$ such that $C \sqcap H \equiv_{\mathcal{T}} \bot$ and $C \sqcap H \sqsubseteq_{\mathcal{T}} D$.

There are many applications that this abductive inference service would be useful, ranging from resource retrieval [5] to semantic matchmaking [10] as well as many others [7]. In order to realize it, an uniform tableaux-based method has been proposed in [4] for the description logic $\mathcal{ALCQ}$. However, this language is rather inexpressive, in many domains the representation of ontology would require more expressive formalisms. Taking a tourism ontology, for example, according to the classification of hotels in France [9], a three stars hotel is an accommodation that must have:

1) at least 6 chambers of all categories,
2) 1 salon and elevators if the building has more than 3 floors,
3) 1 table or desk, 1 telephone and 1 color television in every room,
4) private bathrooms with shower or bathtub, etc.

Clearly, for capturing fully semantics of these constraints, at least the quantified cardinality and the disjointness should be used as shown below:

```
Accommodation ⊑ ¬1hasRoom.
Hotel ⊑ ¬6hasRoom.3StarsRoom
3StarsHotel ⊑ ¬6hasRoom.3StarsRoom
3StarsRoom ⊑ ¬6hasRoom.

PrivateBathRoom ⊑ 1hasEquipment.
Shower ⊑ 1hasEquipment.

Evelator ⊑ 1hasFloor.
```

As a consequence, in order to increase the applicability of abductive reasoning methods in such situations, there is a need to extend the tableaux-based technique for more expressive description logics. In the scope of this paper, we would like to present an extension of the approach for solving the concept abduction problem in $\mathcal{ALCQ}$, the language that extends $\mathcal{ALN}$ with full concept negation and quantified number restrictions.

The rest of this paper is organized as follows. Firstly, in the section 2, we recall some preliminaries on the description logic $\mathcal{ALCQ}$. Next, we give an overview of abductive reasoning tasks in description logics in the section 3. Then, the problem of concept abduction in $\mathcal{ALCQ}$ as well as the tableaux-based
algorithm for computing its solutions are presented in the section 4. An example to illustrate how the algorithm works is presented in the section 5. And finally, the section 6 is reserved for some conclusions.

II. DESCRIPTION LOGIC $\mathcal{ALCQ}$

Like every other description logics, in $\mathcal{ALCQ}$, knowledge of domains is represented by means of concept descriptions which denotes a group of individuals sharing some common characteristics. Starting with a set $NC$ of concept names and a set $NR$ of role names, these formulas are inductively constructed by using a set of concept constructors.

For $\mathcal{ALCQ}$, concepts are formed as follows:

$$
C, D ::= T \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall r.C \mid \exists r.C \mid \geq nr.C \mid \leq nr.C
$$

As usual, the semantics of $\mathcal{ALCQ}$ concept descriptions is defined in a model-theoretic way. Formally, an interpretation $I$ satisfies a logical axiom $C \sqsubseteq D$ (or $C \sqsubseteq D$) if $C \subseteq D$. $\mathcal{I}$ is a model of the TBox $\mathcal{T}$ if it satisfies all of its axioms. $\mathcal{I}$ satisfies an assertion $A$ if $A \in \mathcal{I}$ and it satisfies $R(b, c)$ if $\{a^x \mid b^y, \{a^x \mid b^y \in R^Z \}$.

The interpretation $\mathcal{I}$ satisfies a terminological axiom $C \sqsubseteq D$ (or $C \sqsubseteq D$) if $C \subseteq D$. $\mathcal{I}$ is a model of the ABox $\mathcal{A}$ if it satisfies all its assertions. Finally, $\mathcal{I}$ satisfies an assertion $\alpha$ or an ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$ if in addition to being a model of $\alpha$ or of $\mathcal{A}$, it’s also a model of $\mathcal{T}$.

In addition to the means for representing and storing knowledge about concepts, roles and individuals in the application domain, DLs systems provide a set of standard inference services to reason about them:

1) Concept satisfiability: Given a TBox $\mathcal{T}$, a concept $C$ is satisfiable w.r.t $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C \subseteq \emptyset$.

2) Concept subsumption: Given a TBox $\mathcal{T}$, a concept $C$ is subsumed by a concept $D$ w.r.t $\mathcal{T}$, denoted as $C \subseteq D$ if $C \subseteq D$ for every model $\mathcal{I}$ of $\mathcal{T}$.

3) Entailment: Given an ABox $\mathcal{A}$, an assertion $\alpha$ is entailed by $\mathcal{A}$, denoted as $\mathcal{A} \models \alpha$, if every model of $\mathcal{A}$ satisfies $\alpha$.

4) Consistency: Given an ABox $\mathcal{A}$ and a TBox $\mathcal{T}$, $\mathcal{A}$ is consistent w.r.t $\mathcal{T}$, if there is an interpretation $\mathcal{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$. $\mathcal{A}$ is called simply consistent if it is consistent w.r.t the empty TBox.

Most popular reasoning techniques in DLs are tableau-based methods [2]. Accordingly, in order to verify the satisfiability of a concept $C$ w.r.t a TBox $\mathcal{T}$, the algorithms try to construct a consistent model $\mathcal{I}$ for $C$ and $\mathcal{C}_\mathcal{T}$, where $\mathcal{C}_\mathcal{T}$ is the internalized concept of $\mathcal{T}$ which is defined as follows

$$
\mathcal{C}_\mathcal{T} := \cap_{(C_i \sqsubseteq D_i) \in \mathcal{T}} (\neg C_i \sqcup D_i)
$$

To do that, tableaux or completion graphs are used for representing candidates for $\mathcal{I}$. These graph structures contains nodes and edges that represent respectively individuals and role instances in the domain of models. Each node and edge is labeled with a set $L()$ of concepts and roles of which they are instances. If the tableau does not contain any clash (obvious contradiction) in all its labels, then it’s open, otherwise it’s closed. As any clash-free tableau $T$ corresponds to a consistent canonical interpretation $I_T$, if the algorithms can find an open tableau for $C$ and $\mathcal{T}$ then $C$ is satisfiable w.r.t $\mathcal{T}$, on the contrary, $C$ is unsatisfiable.

III. ABDUCTIVE REASONING IN DESCRIPTION LOGICS

Abduction is a well-known form of commonsense reasoning that gives the ability to infer from observations to explanatory
hypotheses. This technique was introduced firstly by Charles Sander Peirce in [12], since then it has been intensively studied in classical logics. Recently, abduction has begun to gain the attention of the DLs community because of its usefulness to deal with inferences that new applications using DLs would require [7], [6], [11].

In [7], abductive reasoning in DLs are divided into four tasks:

1) **Concept abduction**: Given two concepts C and D represented in some description logic L, Γ is a knowledge-base in L and supposing that C, D are satisfiable w.r.t Γ. A solution to the (conditionalized) concept abduction problem for (Γ, C, D) is any concept H in L′ such that:

   \[ C \cap H \neq \bot \text{ and } C \cap H \subseteq \Gamma \]

The set of all such solutions H is denoted by \( S_{\text{CCA}}(\Gamma, C, D) \).

2) **ABox abduction**: Let Γ be a knowledge-base in some description logic L and α be an ABox assertion in L such that Γ ∪ α is consistent. A solution to the ABox abduction problem for given (Γ, α) is any finite set \( S_A = \{ \beta \} \) of ABox assertions in L′ such that:

   \[ \Gamma \cup S_A \models \alpha \]

The set of all such solutions is denoted by \( S_A(\Gamma, C, D) \).

3) **TBox abduction**: Let Γ be a knowledge-base in some description logic L, C and D be two concepts that are satisfiable w.r.t Γ and supposing that Γ ∪ \{ C ∩ D \} is consistent. A solution to the TBox abduction problem for give (Γ, C, D) is any set \( S_T = \{ E_i \subseteq F_i \} \) of terminological axioms in L′ such that:

   \[ C \cap \Gamma \cup S_T \subseteq \Gamma \]

The set of all such solutions is denoted by \( S_T(\Gamma, C, D) \).

4) **Knowledge-base abduction**: Let L be some description logic, Γ be a knowledge-base in L and γ be a TBox axiom or ABox assertion such that Γ ∪ {γ} is consistent. A solution to the knowledge-base abduction problem for (Γ, γ) is any finite set \( S_K = \{ \delta \} \) of TBox axioms or ABox assertions such that:

   \[ \Gamma \cup S_K \models \gamma \]

The set of all such solutions is denoted by \( S_K(\Gamma, C, D) \).

In the definitions, L is the source logic in which concept descriptions are represented and L′ is the target logic in which the solutions are searched. Without any restriction, solutions to abductive reasoning problems can be very general, trivial and even inconsistent. To prevent undesirable solutions, several conditions have to be imposed in order to obtain worth solutions [7].

There are many applications of abductive reasoning methods in ontology-based systems using description logics, ranging from resource retrieval [5], semantic matchmaking [10] to medical diagnosis as well as many others [7]. In the scope of this paper, we concentrate only on the concept reasoning task.

Concept abduction is proposed as a novel inference service for description logics firstly in the context of semantic matchmaking [11], [10], an ontology-based process aimed to find best matches for a request or a demand among available supplies in the market. Formalizing this operation in the context of description logics, matches between the request D and a supply C with respect to a common ontology T can be categorized into five classes which are listed below in the order of the most preferable to the less:

1) **Exact match** corresponding to \( D \equiv_T C \) implies that the match is perfect because D and C are equivalent concepts.
2) **Full match** corresponding to \( C \subseteq_T D \) suggests that all features required by D can be entirely fulfilled by C .
3) **Plug-in match** corresponding to \( D \subseteq_T C \) indicates that what is demanded by D is more specific than C .
4) **Potential match** corresponds to \( C \cap D \not\subseteq_T \bot \) and thus C and D share something in common.
5) **Partial match** corresponding to \( C \cap D \subseteq_T \bot \) implies the conflict between C and D.

From the traditional point of view, only the first three classes are really interesting [8], [13], as a consequence the last two ones are less considered. There is no method to rank potential matches and partial matches are just ignored.

It turns out that abductive reasoning methods can bring new measurements for potential matching. In fact, if there is a potential match between C and D w.r.t T , the maximal solution H to the abduction problem \( P = (T, C, D) \) can be viewed as the semantic distance between C and D w.r.t T. If there are several potential supplies \( C \), then H can be used to rank them for finding most promising ones[10].

For solving the concept abduction problem \( P = (T, C, D) \) when the logics are \( ALCN \), an uniform tableaux-based algorithm has been proposed [4]. However, this language is quite inexpressive, in many domains the representation of ontology would require more expressive languages. That is the reason why we would like to extend this method for \( ALCQ \), an extension of \( ALCN \) with the full concept negation and quantified cardinality. It will be the subject of discussion in the next section.

IV. CONCEPT ABDUCTION IN ALCQ.

We consider the problem of concept abduction \( P \) for given \( (T, C, D) \) where concepts C, D and the TBox T are expressed in \( ALCQ \), solutions H are computed in the target logic \( ALCN \).

To recall, in the uniform tableaux-based method [4], instead of using one labeling function \( L() \) as normally, two different functions are employed, \( T() \) and \( F() \), to denote respectively concepts and roles that individuals and pairs of individuals belong and do not belong to.

Hence, given a tableau \( \tau \) and let \( \nabla \tau \) be its canonical interpretation, for each node \( v \), each edge \( e = (v, w) \) of \( \tau \), we have :

- if \( C \in T(v) \) and \( D \in F(v) \) then \( v^{\tau_C} \in C^{\nabla \tau} \) and \( v^{\tau_D} \notin D^{\nabla \tau} \).
- if \( R \in T(e) \) and \( S \in F(e) \) then \( \langle v^{\tau_R}, w^{\tau_S} \rangle \in R^{\nabla \tau} \) and \( \langle v^{\tau_R}, w^{\tau_S} \rangle \notin R^{\nabla \tau} \).

From these semantics, two kinds of clashes can be then distinguished in \( \tau \):
1) **Homogeneous clashes** that occur inside T’s or F’s labels and

2) **Heterogeneous clashes** that happen due to the intersection of two different T’s and F’s labels.

If the tableau contains a clash, \( I_2 \) will not be consistent and thus the knowledge-base can not have a model with respect to the tableau’s constraints. But more than that, separating two types of clashes allow to determine if the unsatisfiability is caused either by single concepts (homogeneous clashes) or by both of them (heterogeneous clashes), which will be translated to either by the self-contradiction of concepts or by the subsumption between them.

Consequently, in order to build the solution \( H \), we only need to find concept descriptions that, when added to the root of the original tableau, will generate clashes in every of its derivations, in which at least one clash is heterogeneous. For discovering such formulas, we perform abduction on the labels of the tableau, in which at least one clash is heterogeneous. For the transitive closure of predecessors and descendant is the transitive closure of successors.

Principally, from defined semantics of \( T() \) and \( F() \), let \( I_1 \) be the canonical interpretation of a prefixed tableau \( \tau \), the followings hold:

1) If \( (C_2 \sqcap C_1) \in T(v) \) then \( v^{T'} \in (C_1)^{T'} \land v^{T'} \in (C_2)^{T'} \).
2) If \( (C_1 \sqcap C_2) \in F(v) \) then \( v^{T'} \in (\neg C_1)^{T'} \lor v^{T'} \in (\neg C_2)^{T'} \).
3) If \( \exists R.C \in T(v) \) and there is \( R\)-successor \( w \) of \( v \) then \( w^{T} \in (C)^{T'} \).
4) If \( \forall R.C \in F(v) \) then there must be a \( R\)-successor \( w \) of \( v \) such that \( w^{T'} \in (\neg C)^{T'} \).

Fig. 1. A tableau for \( \exists r.A \sqcap \forall r.B \sqsubseteq \exists r.C \).

Since the tableau is open, \( C \nsubseteq D \). To engender a heterogeneous clash in it, obviously \( H := \forall r.C \) must be the finding solution.

### A. An uniform tableaux-based method for ALCQ

We start by extending uniform tableaux calculus for the description logic ALCQ. To do that, we define a prefixed tableau as a tree-like structure \( \tau = (V, E, v^r, T, F) \), where:

- \( V \) is a finite set of nodes, \( v^r \in V \) is the root of the tableau.
- \( E \) is a finite set of edges.
- \( T() \) and \( F() \) are two labeling functions.

\( T() \) and \( F() \) are called respectively T-labels and F-labels. Sometimes, we will use \( w_i \neq w_j \) when necessary for denoting that \( w_i \) and \( w_j \) are two distinct individuals in \( \tau \).

For any \( e = (v, w) \in E \), if either \( R \in T((v, w)) \) or \( \neg R \in F((v, w)) \), then \( w \) is called a \( R\)-successor of \( v \), and \( v \) is called the predecessor of \( w \) in \( \tau \). Then, as usual, ancestor is the transitive closure of predecessors and descendant is the transitive closure of successors.

1) **\( \sqcap \)-rules:**
   - \( \sqcap \) : If \( (C_1 \cap C_2) \in T(v) \), \( v \) is not blocked and \( \{C_1, C_2\} \nsubseteq T(v) \) then \( T(v) = T(v) \cup \{C_1, C_2\} \).
∀ if \( \tau \) (\( R(\neg \tau) \) - no \( C \) \( \cup \) \( V,E,v,T,F \) where \( T(v) = T(v) \cup \{ C \} \)

- \( \tau_1 = \{ V,E,v',T,F \} \) where \( T(v) = T(v) \cup \{ C \} \)
- \( \tau_2 = \{ V,E,v',T,F \} \) where \( T(v) = T(v) \cup \{ C \} \)

2) \( \sqcup \)-rule:
- if \( (C_1 \cup C_2) \in \{ v \} , v \) is not blocked and \( \{ C_1, C_2 \} \not\subseteq T(v) \) then \( F(v) = F(v) \cup \{ C_1, C_2 \} \).

3) \( \exists \)-rule:
- if \( (\exists R.C) \in T(v) \), \( v \) is not blocked and there is no \( R \)-successor \( w \) of \( v \) with either \( C \in T(w) \) or \( \neg C \in F(w) \) then:
  - \( V = V \cup \{ w \}, E = E \cup \{ (v,w) \} \), \( T((v,w)) = \{ R \} \), and \( T(w) = \{ C \} \)
- if \( (\forall R.C) \in F(v) \), \( v \) is not blocked and there is no \( R \)-successor of \( v \) with \( C \in T(w) \) or \( \neg C \in F(w) \) then:
  - \( V = V \cup \{ w \}, E = E \cup \{ (v,w) \} \), \( F((v,w)) = \{ \neg R \} \), and \( F(w) = \{ C \} \).

4) \( \forall \)-rules:
- if \( (\forall R.C) \in T(v) \), \( v \) is not blocked and there is a \( R \)-successor \( w \) of \( v \) such that \( C \not\subseteq T(w) \) then:
  - \( T(w) = T(w) \cup \{ C \} \)
- if \( (\exists R.C) \in F(v) \), \( v \) is not blocked and there is a \( R \)-successor of \( v \) such that \( C \not\subseteq F(w) \) then:
  - \( F(w) = F(w) \cup \{ C \} \)

5) choose-rule:
- if \( (\leq n R.C) \in T(v) \), \( v \) is not blocked and there is a \( R \)-successor \( w \) of \( v \) such that \( \{ C, \neg C \} \cap T(w) = \emptyset \) then:
  - \( \tau_1 = \{ V,E,v',T,F \} \) where \( T(w) = T(w) \cup \{ C \} \)
  - \( \tau_2 = \{ V,E,v',T,F \} \) where \( T(w) = T(w) \cup \{ \neg C \} \)
- if \( (\geq n R.C) \in F(v) \), \( v \) is not blocked and there is a \( R \)-successor of \( v \) such that \( \{ C, \neg C \} \cap F(w) = \emptyset \) then:
  - \( \tau_1 = \{ V,E,v',T,F \} \) where \( F(w) = F(w) \cup \{ C \} \)
  - \( \tau_2 = \{ V,E,v',T,F \} \) where \( F(w) = F(w) \cup \{ \neg C \} \)

6) \( \geq \)-rules:
- if \( (\geq n R.C) \in T(v) \), \( v \) is not blocked and there are not \( n \) \( R \)-successors \( w_1, \ldots, w_n \) of \( v \) such that \( C \in T(w_i) \) for \( 1 \leq k \leq n \) and \( w_i \neq w_j \) for \( 1 \leq i < j \leq n \) then:
  - \( V = V \cup \{ w_i \}, E = E \cup \{ (v,w_i) \} \), \( T((v,w_i)) = \{ R \} \), let \( T(w_i) = \{ C \} \) if \( C \not\subseteq \top \) and add \( w_i \neq w_j \) for \( 1 \leq i < j \leq n \).
- if \( (\leq n R.C) \in F(v) \), \( v \) is not blocked and there are not \( n \) \( R \)-successors \( w_1, \ldots, w_n \) of \( v \) such that \( \neg C \in F(w_i) \) for \( 1 \leq k \leq n + 1 \) and \( w_i \neq w_j \) for \( 1 \leq i < j \leq n + 1 \) then:
  - \( V = V \cup \{ w_i \}, E = E \cup \{ (v,w_i) \} \), \( F((v,w_i)) = \{ \neg R \} \), let \( F(w_i) = \{ \neg C \} \) if \( C \not\subseteq \top \) and add \( w_i \neq w_j \) for \( 1 \leq i \leq n + 1 \).

7) \( \leq \)-rules:
- if \( (\leq n R.C) \in T(v) \), \( v \) is not blocked and there are \( n + 1 \) \( R \)-successors \( w_1, \ldots, w_{n+1} \) of \( v \) such that \( C \in T(w) \) or \( \neg C \in F(w) \) for \( 1 \leq k \leq n + 1 \) and \( w_i \neq w_j \) for \( 1 \leq i < j \leq n + 1 \) then:
  - for each pair \( w_i, w_j \) such that \( i < j \), either \( C \in T(w) \) or \( \neg C \in F(w) \)
  - for each pair \( w_i, w_j \) such that \( i < j \), either \( C \in T(w) \) or \( \neg C \in F(w) \)
  - for each pair \( w_i, w_j \) such that \( i < j \), either \( C \in T(w) \) or \( \neg C \in F(w) \)

In \( \leq \)-rules, we have used the operator Merge() in order to merge a node with another:
- Merge(\( v, v' \)) where \( v \) and \( v' \) are two nodes such that there is not \( v \neq v' \) in \( \tau \):
  - let \( T(v) = T(v) \cup T(v') \), \( F(v) = F(v) \cup F(v') \)
  - let \( T((v,w)) = T((v',w)) \), and \( F((v,w)) = F((v',w)) \)
  - let \( V = \{ v, v' \} \), \( E = E \cup \{ (u,v'), (v',v) \} \)
  - add \( v \neq v' \) for each \( v, v' \neq v' \) in \( \tau \)

We assign to each rule a priority in such a way that if there is more than one can be applied, those on \( T \)-labels are always executed first. To ensure the termination of the transformation, a blocking strategy is required as usual: for any node \( v \in V \), \( v \) is blocked by \( u \) if:
- \( u \) comes before \( v \) in some enumeration and
- either \( T(v) \subseteq T(u) \) or \( F(v) \subseteq F(u) \).

Some rules are non-deterministic, as a result a tableau \( \tau \) can be transformed into new tableaux \( \tau_1, \ldots, \tau_m \). Because the rules preserve the consistency, the canonical interpretation \( I_{\tau} \) of \( \tau \) is consistent if one of \( I_{\tau_1}, \ldots, I_{\tau_m} \) is so.

Definition 1: (Clashes) A prefixed tableau \( \tau = \langle V,E,v',T,F \rangle \) contains a homogeneous clash if one of the followings holds for some node \( v \in V \):
- either \( \langle A, \neg A \rangle \subseteq T(v) \) or \( \langle A, \neg A \rangle \subseteq F(v) \)
- \( (\leq n R.C) \in T(v) \), \( v \) has \( n \)-successors \( w_1, \ldots, w_m \)

such that:
- \( m > n \).
\(- C \in T(w_i) \) for \(1 \leq i \leq m \) and
\(- w_i \neq w_j \) for \(1 \leq i < j \leq m \);

- \( \{ \geq nR.C \} \in F(v) \), \( v \) has \( m \) \( R \)-successors \( w_1, \ldots, w_m \) such that:
  - \( m \geq n \),
  - \( \neq C \in T(w_i) \) for \(1 \leq i \leq m \) and
  - \( w_i \neq w_j \) for \(1 \leq i < j \leq m \);

The tableau contains a heterogeneous clash if \( T(v) \cap F(v) \neq \emptyset \).

Clashes are defined very similarly as in [4]. To distinguish, we call homogeneous clashes happening in \( T \)-labels and \( F \)-labels respectively \( T \)-homogeneous and \( F \)-homogeneous clashes. Then, the tableau \( \tau \) is complete if no rule can be applied to it. It’s closed if it contains a clash, otherwise it’s open. Finally, the following theorem establishes the soundness and completeness of the algorithm:

**Theorem 1:** Let \( C, D \) be two \( ALCQ \) concepts, \( T \) be an acyclic general TBox in \( ALCQ \). Then \( C \sqsubseteq D \) iff all prefixed tableau constructed by the uniform algorithm starting with \( \tau_0 = (V, E, v^1, T, F) \), where \( V = \{ v^1 \}, E = \emptyset, T(v) = C \) and \( F(v) = \{ D \} \) are closed. Particularly, if every tableau contains at least one \( T \)-homogeneous clash then \( C \sqsubseteq D \).

The complexity of the algorithm is derived directly from the hardness of the normal tableaux-based algorithm for \( ALCQ \). Indeed, if we look at dual rules in detail, each of them applied to \( T(v) = \{ C \} \) and \( F(v) = \{ D \} \) is equivalent to a normal rule applied to \( L(v) = \{ C \cup \neg D \} \). Thus, the uniform algorithm performs neither worse nor better than the ordinary one.

**Theorem 2:** Let \( C \) and \( D \) be two \( ALCQ \) concepts, \( T \) be an acyclic general TBox in \( ALCQ \). Deciding if \( C \sqsubseteq D \) using the uniform tableaux algorithm is \( \text{EXPTIME}\)-complete.

\( B. \) Computing abductive solutions.

As mentioned previously, in order to construct a solution \( H \) to the concept abduction problem \( \mathcal{P} \) for given \( \langle T, C, D \rangle \), we try to find concept descriptions that would generate clashes in every tableau for \( C \sqsubseteq D \) such that among them at least one clash is heterogeneous. To do that, let \( \theta = \{ \tau_1, \ldots, \tau_n \} \) be the set of all open tableaux for verifying \( C \sqsubseteq D \), if \( \theta \) is not empty, to compute \( H \), we start by building for each \( \tau_i \in \theta \), two closing sets \( clT(\tau_i) \) and \( clH(\tau_i) \), which contains formulas that will eventually close it:

**Definition 2:** (Closing sets) Let \( \tau = (V, E, v^1, T, F) \) be any open and complete tableau. We define \( clT(\tau) \) and \( clH(\tau) \) as two disjoint sets of concept descriptions built from labels of \( \tau \) such that, if we let \( \tau_T \) and \( \tau_H \) be respectively complete tableaux derived from \( \tau \) by adding any element of \( clT(\tau) \) and \( clH(\tau) \) into \( T(v^1) \), then \( \tau_T \) and \( \tau_H \) are closed. Moreover,

- every clash in \( \tau_T \) is \( T \)-homogeneous and,
- every clash in \( \tau_H \) is heterogeneous.

It can be seen that any conjunctive expression \( H \) built from elements of \( clT(\tau) \) and \( clH(\tau) \) such that, for each open tableau at least one of its closing elements is appeared in \( H \), will be the finding solution if \( C \sqcap H \not\sqsubseteq T \). We characterize these conditions in the following theorem:

**Theorem 3:** Let \( \mathcal{P} = \langle T, C, D \rangle \) be a concept abduction problem in the description logic \( ALCQ \). Let \( \theta = \{ \tau_1, \ldots, \tau_n \} \) be the set of open and complete prefixed tableaux for \( C \sqsubseteq T \).

If \( \theta \) is not empty, for each \( \tau_i \in \theta \), let \( cl(\tau_i) = clT(\tau_i) \cup clH(\tau_i) \) and choice(.) be some choice function.

For any set \( S = (H_1 = \text{choice}(cl(\tau_1)), \ldots, H_n = \text{choice}(cl(\tau_n))) \), let \( H \) be a conjunctive expression built from elements of \( S \), \( H := H_1 \sqcap \cdots \sqcap H_n \). If:

1) for every \( \tau_i \in \theta \), \( S \sqcap cl(\tau_i) \neq \emptyset \) and
2) \( \exists \tau_j \in \theta \) such that \( S \sqcap cl(T(\tau_j)) = \emptyset \).

then \( H \) is a solution to the problem \( \mathcal{P} \).

For building the closing sets, from the definition of clashes, it can be seen that there are two ways to generate a clash in some node \( v \) of the tableau, the first one is the contradiction of concept names. Thus,

- if \( A \) is a (maybe negated) concept name and \( A \in T(v) \), then \( \neg A \in clT(v) \);
- if \( B \) is a (maybe negated) concept name, \( B \in F(v) \) and \( \neg B \notin T(v) \), then \( B \in clH(v) \);
- if \( v \) has a \( R \)-successor \( w \) and there is not another \( R \)-successor \( w' \) of \( v \) such that \( w' \neq w \) is included in \( \tau \) then :
  - if \( R \in T(\langle v, w \rangle) \) then \( \forall R.E \in clT(v) \), for each \( E \in clT(w) \),
  - otherwise \( \forall R.F \in clH(v) \), for each \( F \in clH(w) \) if \( \forall R.F \notin clT(v) \).

The other way comes from the incompatibility of qualified number restrictions. So,

- if \( v \) has \( n \) \( R \)-successors \( w_1, \ldots, w_n \) such that \( n > 1 \) and \( w_i \neq w_j \) for \(1 \leq i < j \leq n \) then :
  - if \( R \in T(\langle v, w_i \rangle) \) for every \(1 \leq i \leq n \) then :
    - \( \star \leq (k - 1)R.T \in clT(v) \);
  - otherwise :
    - \( \star \leq (k - 1)R.T \in clH(v) \).

From the semantics of tableaux rules and the definition of clashes, it can be verified that if one of these expressions is added into \( T(v) \), a clash will be eventually produced in some label of derived tableaux. Based on these descriptions, we design a recursive algorithms for computing \( clT(v) \) as follow.
Algorithm 1 compute $cl^T(v)$

**Input:** $v$, a node in a prefixed tableau $\tau$.

**Output:** The closing set $cl^T(v)$.

1. $cl^T(v) \leftarrow \emptyset$.
2. if $T(v) \neq \emptyset$ then
   3. for each $A \in T(v)$, $A$ is a (maybe negated) concept name do
      4. $cl^T(v) \leftarrow cl^T(v) \cup \{\neg A\}$.
   5. end for
3. end if
4. $cl^T(v) \leftarrow cl^T(v) \cup \{\forall R.F\}$.
5. return $cl^T(v)$.

In the same manner, $cl^H(v)$ is built by the following procedure.

Algorithm 2 compute $cl^H(v)$

**Input:** $v$, a node in an open and complete prefixed tableau $\tau$.

**Output:** The closing set $cl^H(v)$.

1. $cl^H(v) \leftarrow \emptyset$.
2. if $F(v) \neq \emptyset$ then
   3. for each $B \in F(v)$, $B$ is a (maybe negated) concept name do
      4. $cl^H(v) \leftarrow cl^H(v) \cup \{B\}$.
   5. end for
3. end if
4. if $\tau$ has some $i$ such that $w \neq w'$ is present in $\tau$ do
   5. for each $w \in ch(v)$ and there is not $w'$ such that $w \neq w'$ is present in $\tau$ do
      6. if $\tau$ has $w$ such that $w \neq w'$ is present in $\tau$ do
         7. for each $F \in cl^H(w) do$
            8. $cl^H(v) \leftarrow cl^H(v) \cup \{\forall R.F\}$.
         9. end for
      10. end if
   11. end for
   12. return $cl^H(v)$.

Theorem 4: Given $\tau$ a prefixed tableaux for the description logic $\mathcal{ALCQ}$. The sets $cl^T(v')$ and $cl^H(v')$ computed respectively by the algorithm 1 and 2 are the closing sets of $\tau$. Let $c_T := \bigcup_{\tau \in \Theta} cl^T(\tau)$ and $c_H := \bigcup_{\tau \in \Theta} cl^H(\tau)$. For each closing element $H_i \in c_T \cup c_H$, we define $\tau - set(H_i) = \{\tau \ | \ H_i \in cl(\tau)\}$. Let $S$ be a minimal set of $c_T \cup c_H$ such that:
   - $\emptyset \subseteq \bigcup_{H_i \in S} \tau - set(H_i)$,
   - $\{H_i, \neg H_i\} \subseteq S$.
Then, $H := \bigcap_{H_i \in S} H_i$ is a solution to the abduction problem $P = (T, C, D)$. We implement this process in the algorithm 3 below:

Algorithm 3 conceptAbduction

**Input:** $C$ and $D$, two $\mathcal{ALCQ}$ concepts.

**Input:** $T$, a general acyclic TBox in $\mathcal{ALCQ}$.

**Output:** $H$, a solution to the concept abduction problem $(T, C, D)$.

1. $S := \emptyset$ and $H := \emptyset$.
2. Build $\theta = \{\tau_1, \ldots, \tau_n\}$, the set of open and complete prefixed tableaux for $C \sqsubseteq_T D$.
3. if $\theta = \emptyset$ then
   4. $\forall C \sqsubseteq_T D$ and thus no abductive solution is needed$\ast$.
else
   5. $c_T := \emptyset$ and $c_H := \emptyset$.
   6. for each $\tau_j \in \theta$ do
      7. $c_T := c_T \cup cl^T(\tau_j)$.
      8. $c_H := c_H \cup cl^H(\tau_j)$.
   9. end for
10. Build $S$, the minimal subset of $c_T \cup c_H$ such that:
    - $\emptyset \subseteq \bigcup_{H_i \in S} \tau - set(H_i)$,
    - $\{H_i, \neg H_i\} \subseteq S$ and
    - $S \cap c_H \neq \emptyset$.
11. for each $H_i \in S$ do
   12. $H := H \cap H_i$
   13. end for
14. end if
15. return $H$.

Theorem 5: The concept description $H$ returned by the algorithm 3 is a solution to the concept abduction problem $P$ for given $(T, C, D)$.

In [11], the complexity of any abductive algorithm is proved to be bound by the complexity of the subsumption problem in the corresponding description logics. For the description $\mathcal{ALCQ}$, it’s EXPTIME-complete.

It turns out that our algorithm do not perform worse than this bound. Indeed, we observe that the complexity of algorithms 1 and 2 is equal to the complexity of verifying if a tableau specifies a consistent model, which is known to be exponential in [1]. Then, for finding of the minimal set $S$ which contains sub-descriptions of the solution $H$, at the worst, we have to investigate through all subsets of $c_T \cup c_H$. As this process does not exceed the exponential bound neither, the complexity of the algorithm 3 can not be worse than EXPTIME.

V. Example

To illustrate how the algorithms 1, 2 and 3 work, let’s consider the following abduction problem $P = (T, C, D)$.
where:

\[
T = \{ \text{Room1} \subseteq \exists \text{hasEquipment}. (\text{Desk} \sqcap \text{Table}) \} \\
\{ \text{Room2} \subseteq \exists \text{hasInternet}. (\text{WiFi} \sqcap \text{ADSL}) \}
\]

\[
C := \text{Room1} \sqcup \text{Room2}
\]

\[
D := \exists \text{hasEquipment}. \text{Desk} \sqcap \exists \text{hasInternet}. \text{WiFi}
\]

For computing the solution to \( \mathcal{P} \), the algorithm starts by building tableaux for \( C \supseteq T D \). The whole transformation of tableaux is illustrated in the appendix. Accordingly, the set of open and complete tableaux is

\[
\theta = \{ t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}, t_{21}, t_{22}, t_{23}, t_{24}, t_{25}, t_{26}, t_{27}, t_{28}, t_{29}, t_{30}, t_{31}, t_{32}, t_{33}, t_{34}, t_{35}, t_{36}, t_{37} \}
\]

As \( \theta \) is not empty, \( C \supseteq T D \). We compute then closing sets for these tableaux. The result is listed and then summarized in two tables I and II:

<table>
<thead>
<tr>
<th>Table I</th>
<th>CLOSING SETS OF OPEN TABLEAUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_8 )</td>
<td>{ \text{Room1, Room2, hasEquipment, Desk} }</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>{ \text{Room1, Room2, hasEquipment, Table} }</td>
</tr>
<tr>
<td>( t_{10} )</td>
<td>{ \text{Room1, Room2, hasEquipment, Table} }</td>
</tr>
<tr>
<td>( t_{11} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{12} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{13} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{14} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{15} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{16} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{17} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{18} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{19} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{20} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{21} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{22} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{23} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{24} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{25} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{26} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{27} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{28} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{29} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{30} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{31} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{32} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{33} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{34} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{35} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{36} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
<tr>
<td>( t_{37} )</td>
<td>{ \text{Room1, hasEquipment, WiFi} }</td>
</tr>
</tbody>
</table>

From these elements, we build a concept description \( H = \text{Room1} \sqcap \text{Room2} \sqcap \exists \text{hasEquipment}. \text{Desk} \sqcap \exists \text{hasInternet}. \text{WiFi} \). It can be verified that \( H \) is a solution to the abduction problem \( \mathcal{P} \).

VI. CONCLUSIONS

We have studied in the scope of this paper the problem of concept abduction in the context of \( \mathcal{ALCQ} \), the description logic that is enough expressive for many ontology-based applications with the support of full concept negation and qualified number restrictions. Based on the already developed tableau-based method for solving the problem in \( \mathcal{ACN} \), we have developed an extension of this technique for dealing with concepts represented in \( \mathcal{ALCQ} \).

To do that, we have introduced firstly new expansion rules to handle expressive formulas under the scope of full concept negations and quantified cardinality. Then, using complements of constraints in open tableaux, we compute concept descriptions that will eventually lead to their closure. Finally, a solution to the problem of concept abduction is built as a consistent conjunction of such formulas.

We would like to develop the result of this paper in two directions. Firstly, we would like to improve algorithms 1 and 2 to obtain abductive solutions in more expressive languages, such as \( \mathcal{ALCQ} \). In the other direction, we want to extend the concept abductive reasoning task to the distributed modular formalisms, such as Package-based Description logics (3). Thereby, semantic matchmaking can be performed in more general contexts where requests and offers are specified in different, distributed but related ontologies, which are quite common in domains such as eTourism.

REFERENCES


APPENDIX

The initial tableau is :
\( \tau_0 = \{ V_0, E_0, v^r, T, F \} \) where \( V_0 = \{ v^r \}, E_0 = \emptyset \) with :

- \( T(v^r) = \{ \) Room1 ∪ Room2,
  - ¬Room1
  - ∪ hasEquipment. (Desk ∪ Table)
  - ¬Room2
  - ∪ hasInternet. (ADSL ∪ WiFi)
- \( F(v^r) = \{ \) hasEquipment.Desk ∪ hasInternet.WiFi\)

Tableaux developed on the first branch are :

1) \( \tau_1 = \{ V, E, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \) Room1\};
   - \( F(v^r) = \{ \ldots \} \).

2) \( \tau_2 = \{ V, E, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \) Room2\};
   - \( F(v^r) = \{ \ldots \} \).

3) \( \tau_3 = \{ V, E, v^r, T, F \} \) (CLASH) with :
   - \( T(v^r) = \{ \ldots \) Room1,¬Room1\};
   - \( F(v^r) = \{ \ldots \} \).

4) \( \tau_4 = \{ V, E, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \) Room1\} (Desk ∪ Table)\};
   - \( F(v^r) = \{ \ldots \} \).

5) \( \tau_5 = \{ V, E, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \) Room2\} (Desk ∪ Table)\};
   - \( F(v^r) = \{ \ldots \} \).

6) \( \tau_6 = \{ V, E, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \) Room2\} (Desk ∪ Table)\};
   - \( F(v^r) = \{ \ldots \} \).

7) \( \tau_7 = \{ V_1, E_1, v^r, T, F \} \) (CLASH) with :
   - \( T(v^r) = \{ \ldots \} \);
   - \( F(v^r) = \{ \ldots \) hasEquipment.Desk\};
   - \( T(v_1) = \{ Desk ∪ Table, Desk\} \);
   - \( F(v_1) = \{ Desk\} \);
   - \( T((v^r, v_1)) = \{ hasEquipment\} \);
   - \( F((v^r, v_1)) = \emptyset \).

8) \( \tau_8 = \{ V_1, E_1, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \} \);
   - \( F(v^r) = \{ \ldots \) hasInternet.WiFi\};
   - \( T(v_1) = \{ Desk ∪ Table, Desk\} \);
   - \( F(v_1) = \{ Desk\} \);
   - \( T((v^r, v_1)) = \{ hasEquipment\} \);
   - \( F((v^r, v_1)) = \emptyset \).

9) \( \tau_9 = \{ V_1, E_1, v^r, T, F \} \) with :
   - \( T(v^r) = \{ \ldots \} \);
   - \( F(v^r) = \{ \ldots \) hasEquipment.Desk\};
   - \( T(v_1) = \{ Desk ∪ Table, Table\} \);
   - \( F(v_1) = \{ Desk\} \);
   - \( T((v^r, v_1)) = \{ hasEquipment\} \);
   - \( F((v^r, v_1)) = \emptyset \).

10) \( \tau_{10} = \{ V_1, E_1, v^r, T, F \} \) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \) hasInternet.WiFi\};
    - \( T(v_1) = \{ Desk ∪ Table, Table\} \);
    - \( F(v_1) = \emptyset \);
    - \( T((v^r, v_1)) = \{ hasEquipment\} \);
    - \( F((v^r, v_1)) = \emptyset \).

11) \( \tau_{11} = \{ V_2, E_2, v^r, T, F \} \) where \( V_2 = \{ v^r, v_1, v_2 \}, E_2 = \{ (v^r, v_1), (v^r, v_2) \} \) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \} \);
    - \( T(v_2) = \{ Desk\} \);
    - \( F(v_2) = \emptyset \);
    - \( T((v^r, v_1)) = \{ hasEquipment\} \);
    - \( F((v^r, v_1)) = \emptyset \);
    - \( T((v^r, v_2)) = \{ hasInternet\} \);
    - \( F((v^r, v_2)) = \emptyset \).

12) \( \tau_{12} = \{ V_2, E_2, v^r, T, F \} \) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \} \);
    - \( T(v_1) = \{ Table\} \);
    - \( F(v_1) = \emptyset \);
    - \( T((v^r, v_1)) = \{ hasEquipment\} \);
    - \( F((v^r, v_1)) = \emptyset \);
    - \( T((v^r, v_2)) = \{ hasInternet\} \);
    - \( F((v^r, v_2)) = \emptyset \).

13) \( \tau_{13} = \{ V_2, E_2, v^r, T, F \} \) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \} \);
    - \( T(v_1) = \{ Desk\} \);
    - \( F(v_1) = \emptyset \);
    - \( T((v^r, v_1)) = \{ hasEquipment\} \);
    - \( F((v^r, v_1)) = \emptyset \);
    - \( T((v^r, v_2)) = \{ hasInternet\} \);
    - \( F((v^r, v_2)) = \emptyset \).

14) \( \tau_{14} = \{ V_2, E_2, v^r, T, F \} \) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \} \);
    - \( T(v_1) = \{ Table\} \);
    - \( F(v_1) = \emptyset \);
    - \( T((v^r, v_1)) = \{ hasEquipment\} \);
    - \( F((v^r, v_1)) = \emptyset \);
    - \( T((v^r, v_2)) = \{ hasInternet\} \);
    - \( F((v^r, v_2)) = \emptyset \).

15) \( \tau_{15} = \{ V_2, E_2, v^r, T, F \} \) (CLASH) with :
    - \( T(v^r) = \{ \ldots \} \);
    - \( F(v^r) = \{ \ldots \) hasEquipment.Desk\}. 
Fig. 2. Transformation of tableaux

16) $\tau_{16} = \{V_2, E_2, v', T, F\}$ (CLASH) with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasInternet.WiFi\}$
- $T(v_1) = \{Desk\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{WiFi\}$
- $F(v_2) = \{WiFi\}$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$
- $F((v', v_2)) = \emptyset$

17) $\tau_{17} = \{V_2, E_2, v', T, F\}$ (CLASH) with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasInternet.WiFi\}$
- $T(v_1) = \{Table\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{WiFi\}$
- $F(v_2) = \{WiFi\}$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$
- $F((v', v_2)) = \emptyset$

18) $\tau_{18} = \{V_2, E_2, v', T, F\}$ with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasEquipment.Desk\}$
- $T(v_1) = \{Table\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{WiFi\}$
- $F(v_2) = \emptyset$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$

19) $\tau_{19} = \{V_2, E_2, v', T, F\}$ with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasInternet.WiFi\}$
- $T(v_1) = \{Desk\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{ADSL\}$
- $F(v_2) = \{WiFi\}$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$
- $F((v', v_2)) = \emptyset$

20) $\tau_{20} = \{V_2, E_2, v', T, F\}$ (CLASH) with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasEquipment.Desk\}$
- $T(v_1) = \{Desk\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{ADSL\}$
- $F(v_2) = \emptyset$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$
- $F((v', v_2)) = \emptyset$

21) $\tau_{21} = \{V_2, E_2, v', T, F\}$ with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasEquipment.Desk\}$
- $T(v_1) = \{Table\}$
- $F(v_1) = \emptyset$
- $T(v_2) = \{ADSL\}$
- $F(v_2) = \emptyset$
- $T((v', v_1)) = \{hasEquipment\}$
- $F((v', v_1)) = \emptyset$
- $T((v', v_2)) = \{hasInternet\}$
- $F((v', v_2)) = \emptyset$

22) $\tau_{22} = \{V_2, E_2, v', T, F\}$ with :
- $T(v') = \{\ldots\}$
- $F(v') = \{\ldots, \exists hasInternet.WiFi\}$
- $T(v_1) = \{Table\}$
- $F(v_1) = \emptyset$
\( T(v_2) = \{ \text{ADSL} \} \);
\( F(v_2) = \{ \text{WiFi} \} \);
\( T(\langle v', v_1 \rangle) = \{ \text{hasEquipment} \} \);
\( F(\langle v', v_1 \rangle) = \emptyset \);
\( T(\langle v', v_2 \rangle) = \{ \text{hasInternet} \} \);
\( F(\langle v', v_2 \rangle) = \emptyset \).

In the very similar manner, \( \tau_2, \tau_{23}, ..., \tau_{42} \) are developed on the second branch.