The Open Cybernetics & Systemics Journal, 2015, *9*, 2545-2553

Concept Implicate Tree for Description Logics

Tingting Zou* and Ansheng Deng

Information Science and Technology College, Dalian Maritime University, Dalian, 116026, China

Abstract: Description logics is a class of knowledge representation languages with high expressive power, and the computational complexities of the queries of these expressive description logics are defined as PSPACE-complete. Moreover, knowledge compilation can be regarded as a new direction of research for dealing with the computational intractable reasoning problems. In fact, knowledge compilation based on description logic has been investigated in recent years. However, when the compiled knowledge base is exponential as compared to original knowledge base, the queries are not. Therefore, we proposed a new knowledge compilation method for description logic to solve the queries in linear time depending on the size of the query. In this paper, we first introduced the concept implicate tree for the ALC concept. Then, we present an algorithm, which can transform an ALC concept into an equivalent concept implicate tree, and proved that each branch of the tree is an implicate of this concept. Finally, we proved that the queries are computable in linear time. The proposed method has an important property that no matter how large the concept implicate tree is, any query can be resolved in linear time depending on the size of the query.

Keywords: ALC, Description logic, Knowledge compilation, PSAPCE-complete, Algorithm Build CIT, Tractable querying.

1. INTRODUCTION

Description logics (DL) is a class of knowledge representation languages, which can model an application domain of interest by a structured and formally well-understood method[1]. In fact, DLs can be used in various areas, for example, Semantic Web [2, 3], Ontologies [4], and software engineering [5]. Schmidt-Schauß and Smolka proposed description logic ALC, and proved that the queries of ALC concepts were PSPACE-complete [6]. Subsequently, Donini *et al.* stated that the queries of ALCN concepts were also PSPACE-complete [7]. With the rapid development of DLs, abundant DL systems have been presented, such as SHIN [8], SHIQ [9], SHOIQ [10, 11], SROIQ [12] and so on. However, the computational complexities of the queries of these expressive description logics are intracte.

Knowledge compilation has emerged as a new direction of research for dealing with the computational intractability of general propositional reasoning [13]. In this approach, reasoning process is split into two phases: off-line compilation and on-line query-answering [14]. In the first phase, the propositional knowledge base is compiled into some target language, which is typically tractable. In the latter phase, the query is actually answered by using the compiled knowledge base of the first phase. The key of this approach is that knowledge compilation needs to be done only once to be accessible for different queries. Hence, the compiling time can be amortized by many queries concerning the compiled knowledge base [15]. There are many target languages for knowledge compilation, such as prime implicate [16], DNNF [17], and so on. In fact, the queries for these target languages are based on polynomial time or linear time dependent on the size of the compiled knowledge base. Moreover, Murray and Rosenthal introduced the reduced implicate tree that is a target language for knowledge compilation, and proved that a query can be done in linear time considering the size of the query [18-20].

As mentioned above, knowledge compilation is an efficient method to deal with intractable problems. Therefore, many researchers have conducted their studies on knowledge compilation for description logics in recent years. Selman and Kautz compiled a concept of DL FL into two approximate concepts of DL FL^{-}, being the first knowledge compilation method for DL [21]. Subsequently, Furbach and Obermajer introduced the linkless concept description for ALC concepts, which can be regarded as a target language for knowledge compilation, by presenting an algorithm that transformed ALC concept to equivalent linkless concept description, and proved that queries for such descriptions were resolved in linear time based on the size of the descriptions [22]. Later, they used this technique for precompiled ALC concepts and TBoxes so that queries can be addressed in linear time [23, 24]. Moreover, Bienvenu proposed the prime implicate normal form for ALC concepts, and concluded that the queries of such forms are based on polynomial time [25]. Tingting Zou *et al.*, proposed a novel knowledge compilation method for description logic based on the concept extension rule [26].

In fact, the queries of these methods were also based on the polynomial time or linear time depending on the size of the compiled knowledge base. However, when the compiled knowledge base was exponential in terms of the size of the original knowledge base, the queries were not addressed rapidly. This paper aims to further improve the reduced implicate tree for propositional logic, to make it a much more

---

*Address correspondence to this author at the Information Science and Technology College, Dalian Maritime University, Dalian, Liaoning, 116026, China; Tel/Fax: 041184723122; E-mail: zoutt@dlmu.edu.cn

1874-110X/15

efficient description logic for target language. Therefore, we proposed a new knowledge compilation method for the description logic based on the concept implicate tree, for which the queries can be addressed in linear time based on the size of the query regardless of the size of the compiled knowledge base.

In this paper, we first introduced the concept implicate tree for ALC concept, which is a target language for knowledge compilation, and defined the concept represented by concept implicate tree. Then, an algorithm was presented, which can transform ALC concepts into the concept implicate trees. Moreover, we proved that the concept represented by this concept implicate tree was equivalent to the original ALC concept, and each branch of the tree was an implicate of the original concept. Furthermore, we explained that the satisfiability-testing and tautology-testing were carried out in linear time with respect to the concept implicate tree. Finally, we presented an algorithm determining the subsumption of two concepts, and proved that subsumption-testing was computable in linear time based on the size of the query. In a word, this method has an important property that no matter how large the concept implicate tree is, any query can be assessed in linear time depending on the size of the query.

The rest of this paper is organized as follows. In section 2, the concept implicate tree is defined. Section 3 presents the process of transforming an ALC concept into an equivalent concept implicate tree. In Section 4, it is proved that the queries are computable in linear time. Section 5 summarizes the main results.

2. CONCEPT IMPLICATE TREE

Let C_A, R_A and I_A be the pairwise disjointing sets of atomic concepts, abstract role names, and abstract individuals, respectively, and $\sqcup$ operation be the concept disjunction, with $\sqcap$ operation being the concept conjunction.

Definition 1. Literal L, ALC concept C, and clausal concept cl, are defined as follows:

$$L := \ |\bot| A \mid \neg A \mid \exists R.L \mid \forall R.L \ ,$$

$$C := L \mid C \sqcup C \mid C \sqcap C \ ,$$

$$cl := L \mid cl \sqcup cl \ ,$$

where $A \in C_A$, $R \in R_A$.

Definition 2. In literal L, A *or* $\neg A$ is called the concept literal, and A is known as the atomic concept variable, with the form $\exists R.L$ or $\forall R.L$ known as the role concept literal and also as the role concept variable.

For any concept C, $V_{Con}(C)$ denotes the set of all atomic concept variables of C, and $V_{Rol}(C)$ denotes the set of all role concept variables of C. Moreover, $depth(QR.L)$ denotes the number of the form QR in $QR.L$, $Q \in \{\forall, \exists\}$. For example, if

$$C = (A_1 \sqcup \neg A_2 \sqcup \exists R_1.\neg A_3) \sqcap (A_1 \sqcup \forall R_1.\exists R_2.A_2) \ ,$$

then,

$$C_A = \{A_1, A_2, A_3\}, \ R_A = \{R_1, R_2\} \ ,$$

$$V_{Con}(C) = \{A_1, A_2\}, \ V_{Rol}(C) = \{\exists R_1.\neg A_3, \forall R_1.\exists R_2.A_2\} \ ,$$

$$depth(\exists R_1.\neg A_3) = 1, \ depth(\forall R_1.\exists R_2.A_2) = 2 \ .$$

Let C_1, and C_2 be the ALC concepts, and B is the subconcept of C_1. C_1 $[C_2/ B]$ is used to refer to the new concept, which is produced by substituting C_2 for every occurrence of B that is not in the scope of role restriction in C_1. Especially, if C_2 is $\top$ or $\bot$, B is an atomic concept variable A or role concept variable $QR.L$, $Q \in \{\forall, \exists\}$, then $C_1[\top/B]$ denotes that $\top$ is substituted for B, and $\bot$ for $\neg B$, but $\top$ or $\bot$ is not substituted for $QR.B$ or $QR.\neg B$.

Definition 3. Reduction rules are defined as follows:

$C[B / B \sqcup \bot]$, $C[\bot / B \sqcap \bot]$,

$C[B / B \sqcap \top]$, $C[\top / B \sqcup \top]$,

$C[\bot / B \sqcap \neg B]$, $C[\top / B \sqcup \neg B]$,

$C[\bot / \exists R.\bot]$, $C[\top / \forall R.\top]$.

Definition 4. Let $V_{Con}(C) = \{A_1, A_2, \ldots, A_n\}$ be the set of atomic concept variables of ALC concept C, and

$$V_{Rol}(C) = \{QR_i.L_j \mid Q \in \{\exists, \forall\}, 1 \le i \le p, 1 \le j \le q\}$$

be the set of role concept variables of ALC concept C. A partial ordering relation $\prec$ on sets $V_{Con}(C)$ and $V_{Rol}(C)$ is defined as follows:

(1) $A \prec QR.L$ iff $A \in V_{Con}(C), QR.L \in V_{Rol}(C)$;

(2) $A_i \prec A_j$ iff i<j;

(3) $QR_i.L_j \prec QR_r.L_s$ iff $depth(QR_i.L_j) < depth(QR_r.L_s)$;

(4) $QR_i.L \prec QR_r.L$ iff i<r;

(5) $QR.L_j \prec QR.L_s$ iff j<s;

(6) $\exists R.L \prec \forall R.L$.

In this paper, we assumed that $V_{Con}(C)$ and $V_{Role}(C)$ satisfy this partial ordering relation, that is to say, $V_{Con}(C)$ and $V_{Role}(C)$ are the ordered sets. For simplicity, we write $V_{Con}(C)$ as V_{Con}, and $V_{Role}(C)$ as V_{Role}.

Definition 5. Let C be an ALC concept, and cl be a clausal concept. Then cl is an implicate of C if and only if $C \sqsubseteq cl$. Moreover, cl is a prime implicate of C if and only if $C \sqsubseteq cl$, and there does not exist an implicate cl' of C such that $C \sqsubseteq cl' \sqsubseteq cl$ and $cl \not\sqsubseteq cl'$.

Definition 6. Concept implicate tree (CIT) T for ALC concept C is a tree defined as follows:

(1) If C is tautology, then T contains only one root node labelled as $\top$;

(2) If *C* is unsatisfiable, then *T* contains only one root node labeled as | ;

(3) Otherwise, root node of *T* is labelled as | , and for any implicate $cl = L_1 \sqcup L_2 \sqcup \cdots \sqcup L_m$ of C, root node has a child node labelled as L_1, which is the root of a subtree containing a branch with labels corresponding to $L_2 \sqcup \cdots \sqcup L_m$.

(4) *T* is reduced by using the rules in Definition 3, until no rule can be applied.

According to definition 6, it can be observed that each branch of CIT *T* is an implicate of *C*.

Definition 7. Let *T* be the concept implicate tree for ALC concept *C*. Then concept C_T that is represented by the tree *T* is defined as follows:

(1) If *T* has only one node, then C_T is the label of this node.

(2) Otherwise, C_T is the concept disjunction of two concepts; one concept is the label of the root, and the other is the concept conjunction of labels of all branches of this root.

Example 1: ALC concept

$$C = (A_1 \sqcup A_2 \sqcup \forall R_2.\neg A_4) \sqcap (A_1 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap (\neg A_1 \sqcup \neg A_2 \sqcup \forall R_1.\exists R_2.A_5) .$$

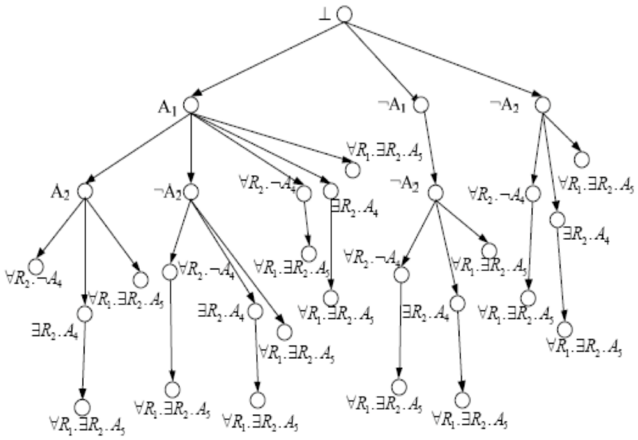
where, $V_C(C) = \{A_1, A_2\}$, $V_R(C) = \{\forall R_2.\neg A_4, \forall R_1.\exists R_2.A_5\}$.

Then, the concept implicate tree *T* of *C* is shown as follows, and each branch of *T* is an implicate of *C*. For example,

$A_1 \sqcup A_2 \sqcup \forall R_2.\neg A_4$, $A_1 \sqcup A_2 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

and $\neg A_1 \sqcup \neg A_2 \sqcup \forall R_1.\exists R_2.A_5$,

are all implicates of *C*.



Moreover, the concept C_T is

$$C_T = (A_1 \sqcup ((A_2 \sqcup (\forall R_2.\neg A_4 \sqcap (\exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap \forall R_1.\exists R_2.A_5)) \sqcap (\neg A_2 \sqcup ((\forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap (\exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5) \sqcap \forall R_1.\exists R_2.A_5)) \\ \sqcap (\forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5) \sqcap (\exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap \forall R_1.\exists R_2.A_5)) \sqcap (\neg A_1 \sqcup (\neg A_2 \sqcup ((\forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap (\exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5) \sqcap \forall R_1.\exists R_2.A_5))) \\ \sqcap (\neg A_2 \sqcup (\neg A_2 \sqcup ((\forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5) \\ \sqcap (\exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5) \sqcap \forall R_1.\exists R_2.A_5))$$

3. TRANSFORMATION

In this section, we introduced a method to transform an ALC concept into an equivalent concept implicate tree, and proved that each branch of the tree is an implicate of this concept. Let Cimp(*C*) be the sets of implicates of concept *C*.

Theorem 1. Let C be the ALC concept, V_{Con} be the atomic concept variables of C, V_{Rol} be a role concept variables of C, and clausal concept cl be an implicate of C. If there exists an atomic concept variable A (or a role concept variable $QR.L$, $Q \in \{\forall, \exists\}$), such that $A \in V_{Con}(C)$ and $A \notin V_{Con}(cl)$ (or $QR.L \in V_{Rol}(C)$, $QR.L \notin V_{Rol}(cl)$), then

$cl \in \mathrm{Cimp}(C[\bot / A]) \cap \mathrm{Cimp}(C[\top / A])$

(or $cl \in \mathrm{Cimp}(C[\bot / QR.L]) \cap \mathrm{Cimp}(C[\top / QR.L])$).

Proof. (1) We first proved that $cl \in \mathrm{Cimp}(C[\top / A])$. Let $I = < \Delta^I, \bullet^I >$ be a model of concept $C[\top / A]$, therefore, $(C[\top / A])^I \neq \varnothing$ .Following this , we extended I to $I' = < \Delta^{I'}, \bullet^{I'} >$ by setting $\Delta^I = \Delta^{I'}$, $A^{I'} = \Delta^I$, then $C^{I'} = (C[\top / A])^I \neq \varnothing$. Therefore, I' became the model of *C*. Because clausal concept *cl* was an implicate of *C*, therefore, $C^{I'} \mid \ cl^{I'}$. Since $A \notin V_{Con}(cl)$, then $cl^{I'} = cl^I$. Hence, $(C[\top / A])^I = C^{I'} \subseteq cl^{I'} = cl^I$. Thus, $C[\top / A] \sqsubseteq cl$. According to the Definition 5, $cl \in \mathrm{Cimp}(C[\top / A])$. The proof for $C[\top / QR.L]$ is similar.

(2)Following this , we proved that $cl \in \mathrm{CImp}(C[\bot / A])$. Let $I = < \Delta^I, \langle^I >$ be the model of concept $C[\bot / A]$, therefore , $(C[\bot / A])^I \neq \varnothing$. Now, we extended I to $I' = < \Delta^{I'}, \bullet^{I'} >$ by setting $\Delta^I = \Delta^{I'}$, $A^{I'} = \rightarrow$, then $C^{I'} = (C[\bot / A])^I \neq \varnothing$. Therefore, $I' = < \Delta^I, \bullet^{I'} >$ is the model of concept *C*. Because clausal concept *cl* was an implicate of concept *C*, therefore, $C^{I'} \subseteq cl^{I'}$. Since $A \notin V_{Con}(cl)$, then $cl^{I'} = cl^I$. Hence, $(C[\bot / A])^I = C^{I'} \subseteq cl^{I'} = cl^I$. Thus, $C[\bot / A] \sqsubseteq cl$. According to the Definition 5, $cl \lfloor \ \mathrm{CImp}(C[\bot / A])$. The proof for $C[\bot / QR.L]$ is similar.

Above all, $cl \in \mathrm{Cimp}(C[\bot/A]) \cap \mathrm{Cimp}(C[\top/A])$ or $cl \in \mathrm{Cimp}(C[\bot/QR.L]) \cap \mathrm{Cimp}(C[\top/QR.L])$.

Theorem 2. Let C_1 and C_2 be the ALC concepts. Then, $\mathrm{Cimp}(C_1 \sqcup C_2) = \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$.

Proof. (⇒) Assuming that a clausal concept $cl \in \mathrm{Cimp}(C_1 \sqcup C_2)$, we proved that $cl \in \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$. (1) To see that $cl \in \mathrm{Cimp}(C_1)$. Let $I = <\Delta^I, \bullet^I>$ be the model of C_1, then I is also a model of $C_1 \sqcup C_2$. Since $cl \in \mathrm{Cimp}(C_1 \sqcup C_2)$, then $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I \subseteq cl^I$. Therefore, $C_1^I \subseteq cl^I$, $C_1 \sqsubseteq cl$. Thus $cl \in \mathrm{Cimp}(C_1)$. (2) To see that $cl \in \mathrm{CImp}(C_2)$. Let $I = <\Delta^I, \bullet^I>$ be the model of concept C_2, then I is also a model of $C_1 \sqcup C_2$. Since $cl \in \mathrm{Cimp}(C_1 \sqcup C_2)$, then $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I \subseteq cl^I$. therefore, $C_2^I \subseteq cl^I$, $C_2 \gneqq cl$. Thus $cl \in \mathrm{Cimp}(C_2)$. Therefore, $cl \in \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$, and $\mathrm{Cimp}(C_1 \sqcup C_2) \subseteq \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$.

(⇐) Assuming that $cl \in \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$, we proved that $cl \in \mathrm{Cimp}(C_1 \sqcup C_2)$. Let $I = <\Delta^I, \bullet^I>$ be the model of $C_1 \sqcup C_2$, then I is a model of C_1 or C_2. There are three cases:

(1) I is a model of C_1, but not a model of C_2. Since $cl \in \mathrm{Cimp}(C_1)$, then $C_1^I \subseteq cl^I$. Therefore, $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I = C_1^I \cup \varnothing = C_1^I \subseteq cl^I$.

(2) I is a model of C_2, but not a model of C_1. Since $cl \in \mathrm{Cimp}(C_2)$, then $C_2^I \mid \ cl^I$ .Therefore,

$$(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I = \varnothing \cup C_2^I = C_2^I \subseteq cl^I.$$

(3) I is a model of C_1, and also is a model of C_2. Since $cl \in \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$, then $C_1^I \subseteq cl^I$, $C_2^I \subseteq cl^I$. Therefore, $(C_1 \sqcup C_2)^I = C_1^I \cup C_2^I \subseteq cl^I$. Therefore,

$\mathrm{cl} \in \mathrm{Cimp}(C_1 \sqcup C_2)$, and

$\mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2) \subseteq \mathrm{Cimp}(C_1 \sqcup C_2)$.

Above all, $\mathrm{Cimp}(C_1 \sqcup C_2) = \mathrm{Cimp}(C_1) \cap \mathrm{Cimp}(C_2)$.

Theorem 3. Let cl be a clausal concept without the concept variable E (E=A or E=$QR.L$), and C be any ALC concept. Then cl is an implicate of C if cl is an implicate of $C[\bot/E] \sqcup C[\top/E]$.

Proof. According to Theorem 1 and Theorem 2, it is obvious that this conclusion is correct.

According to Theorem 3, the set of implicates of C consists of three parts: (1) The first part is the set of implicates concept variables E, E=A or E=$QR.L$; (2) The second part is the set of implicate concept variable $\neg E$, E=A or E=$QR.L$ $QR.L$; (3) the third part is the set of implicates of $C[\bot/E] \sqcup C[\top/E]$, which neither contains concept variable E nor the concept variable $\neg E$, E=A or E=$QR.L$.

Therefore, a concept implicate tree T can be regarded as a ternary tree, with each node having three subtrees except the leaf node. The first subtree is T_1, the second subtree is T_2, and the third subtree is T_3. Let N be a node labelling E_i, and T_1, T_2, T_3 be the three subtrees of node N. Then the root node of T_1 is labelled as E_{i+1}, and T_1 contains the sets of implicates occurring E_{i+1}. Moreover, the root node of T_2 is labelled as $\neg E_{i+1}$, and T_2 contains the sets of implicates occurring $\neg E_{i+1}$. Furthermore, the root node of T_3 is labelled as $\bot$, and T_3 contains the set of implicates not occurring E_{i+1} and $\neg E_{i+1}$, which are the intersection of T_1 and T_2 irrespective of E_{i+1} and $\neg E_{i+1}$.

Therefore, a method was proposed to build a concept implicate tree of a given concept. First, the structure of a node of the tree was defined as shown in Fig. (**1**), then the algorithms Simplify and BuildCIT were presented as shown in Figs. (**2**) and (**3**). Algorithm BuildCIT has four input parameters,

```
Structure CITnode(label:    string,
                  leaf:     boolean,
                  first:    ↑ CITnode,
                  second:   ↑ CITnode,
                  third:    ↑ CITnode);
```

Fig. (1). Structure CITnode.

```
Algorithm Simplify
Input: concept C
Output: simplified concept of C.
1.  Applying the following rules until no rule can be applied:
        $C = C[\bot / F \sqcap \bot]$, $C = C[F / F \sqcup \bot]$,
        $C = C[F / F \sqcap \top]$, $C = C[\top / F \sqcup \top]$,
        $C = C[\bot / F \sqcap \neg F]$, $C = C[\top / F \sqcup \neg F]$,
        $C = C[\bot / \exists R.\bot]$, $C = C[\top / \forall R.\top]$.
2.  Return C.
```

Fig. (2). Algorithm simplify.

Where, ALC is a concept C, with the set of atomic concept variables V_{Con}, the set of role concept variables V_{Rol}, node N, one output parameter, and the concept implicate tree T. Initially, V_{Con}=$V_{Con}(C)$, V_{Rol}=$V_{Rol}(C)$, N=nil. For every node, the algorithm BuildCIT first built the first subtree and the second subtree, followed by the third subtree based on computing the intersection of the first two subtrees which is illustrated in Fig. (**4**).

Example 2. For the concept C in example 1, the algorithm BuildCIT(C, V_{Con}, V_{Rol}, nil) built a tree T as follows:

First, a new CITnode N' was built, which is root node of tree T, and $N'.label = \bot$, returning to BuildCIT(C, V_{Con}, V_{Rol}, N').

For BuildCIT(C, V_{Con}, V_{Rol}, N'):

1) Let $N = N'$;

2) An atomic concept variable A *was selected*$_1$;

3) $C_1 = \text{Simplify}(C[\bot/E]) = (A_2 \sqcup \forall R_2.\neg A_4) \sqcap \forall R_1.\exists R_2.A_5$
$C_2 = \text{Simplify}(C[\top/E]) = \neg A_2 \sqcup \forall R_1.\exists R_2.A_5$;

4) A new CIT node N_1 of tree T *was built with* N_1.label= A_1, N.first= N_1, call BuildCIT(C_1,{A_2},V_{Rol},N_1);

```
Algorithm BuildCIT
Input: concept $C$, $V_{Con}$, $V_{Rol}$, node $N$;
Output: concept implicate tree $T$;
1.  If $C = \bot$ or $C = \top$, then
        build a new CITnode $N'$ of tree $T$, and $N'.label = C$,
        returen $T$.
2.  If $N$=nil, then
        build a new CITnode $N'$, which is root node of tree $T$,
        and $N'.label = \bot$, return BuildCIT($C$, $V_{Con}$, $V_{Rol}$, $N'$).
3.  If $V_{Con} = \varnothing$, then
        select the first role concept variable
        $E = QR_j.L_i \in V_{Rol}$;
    Else, select the first atomic concept variable $E = A \in V_{Con}$.
4.  Let $C_1 = \text{Simplify}(C[\bot/E])$,
        $C_2 = \text{Simplify}(C[\top/E])$.
5.  If $C_1 = \bot$, then
        build a new CITnode $N_1$ of tree $T$, and $N_1$.label=$E$,
        $N_1$.leaf=True, $N$.first= $N_1$.
6.  If $C_2 = \bot$, then
        build a new CITnode $N_2$ of tree $T$, and $N_2$.label= $\neg E$,
        $N_2$.leaf=True,
        $N$.second= $N_2$.
7.  If $C_1 = \top$, then $N$.first=nil, $N$.third=nil;
    Else
        build a new CITnode $N_1$ of tree $T$, and $N_1$.label=$E$,
        $N$.first= $N_1$,
        if $V_{Con} \neq \varnothing$, then
            call BuildCIT($C_1$,$V_{Con}$-{$E$},$V_{Rol}$,$N_1$);
        else
            call BuildCIT($C_1$,$V_{Con}$,$V_{Rol}$-{$E$},$N_1$).
8.  If $C_2 = \top$, then $N$.second=nil, $N$.third=nil;
    Else
        build a new CITnode $N_2$ of tree $T$, and $N_2$.label= $\neg E$,
        $N$.second= $N_2$,
        if $V_{Con} \neq \varnothing$, then
            call BuildCIT($C_2$,$V_{Con}$-{$E$},$V_{Rol}$,$N_2$);
        else
            call BuildCIT($C_2$,$V_{Con}$,$V_{Rol}$-{$E$},$N_2$).
9.  If($N_1$.leaf and $N_2$.leaf), then
        delete node $N_1$, $N_2$, and $N$.leaf=True, return $T$.
10. If ($N$.first≠nil and $N$.second≠nil), then
        build a new CITnode $N_3$ of tree $T$, and $N$.third=$N_3$,
        call BuildThird($N$.first, $N$.second, $N_3$),
        and $N_3.label = \bot$.
11. Return $T$.
```

Fig. (3). Algorithm BuildCIT.

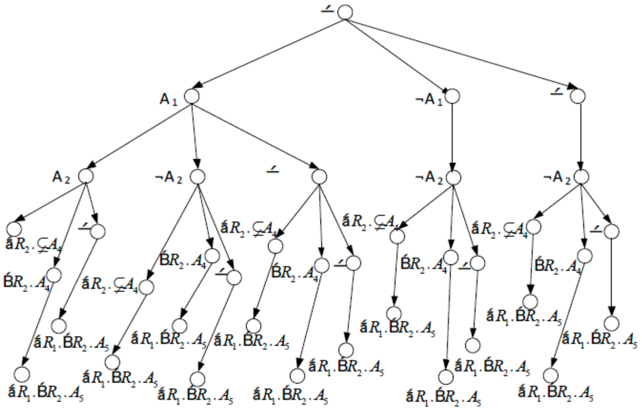
```
Algorithm BuildThird
Input: CIT nodes $N_1$,$N_2$,$N_3$;
Output: tree $T$.
1.  $N_3$.label= $N_1$.label.
2.  If $N_1$.leaf==true and $N_2$.leaf==true,
        then $N_3$.leaf=true, return $T$.
3.  If $N_1$.leaf==true,
        then $N_3$.first= $N_2$.first, $N_3$.second= $N_2$.second,
        $N_3$.third= $N_2$.third, return $T$.
4.  If $N_2$.leaf==true,
        then $N_3$.first= $N_1$.first, $N_3$.second= $N_1$.second,
        $N_3$.third= $N_1$.third, return $T$.
5.  If $N_1$.first=nil or $N_2$.first=nil, then $N_3$.first=nil;
    Else
        build a new CITnode $N_{31}$ of tree $T$,
        $N_3$.first= $N_{31}$,
        call BuildThird($N_1$.first, $N_2$.first, $N_{31}$).
6.  If $N_1$.second=nil or $N_2$.second =nil,
        then $N_3$.second =nil;
    Else
        build a new CITnode $N_{32}$ of tree $T$,
        $N_3$.second = $N_{32}$,
        call BuildThird($N_1$.second, $N_2$.second, $\mathbf{N}_{32}$).
7.  If $N_1$.third =nil or $N_2$.third =nil,
        then $N_3$.third =nil;
    Else
        build a new CITnode $N_{33}$ of tree $T$,
        $N_3$.third = $N_{33}$,
        call BuildThird($N_1$.third, $N_2$.third, $N_{33}$).
8.  Return $T$.
```

Fig. (4). Algorithm BuildThird.

5) A new CIT node N_2 of tree *T was built* with N_2.label= $\neg A_1$, *N*.second= N_2, call BuildCIT(C_2, $\{A_2\}$,V_{Rol},N_2);

6) A new CITnode N_3 of tree *T was built* with *N*.third=N_3, call BuildThird(N_1, N_2, N_3), $N_3.label = \perp$;

7) Returning to T.



In this algorithm, three algorithms are addressed, BuildCIT(C_1,$\{A_2\}$,V_{Rol},N_1), BuildCIT(C_2, $\{A_2\}$,V_{Rol},N_2), and BuildThird(N_1, N_2, N_3). The algorithms BuildCIT(C_1, $\{A_2\}$,V_{Rol},N_1) and BuildCIT(C_2, $\{A_2\}$,V_{Rol},N_2) iterate the process of algorithm BuildCIT(C, V_{Con}, V_{Rol}, N'), and build the first and second subtrees of node N'. Algorithm BuildThird(N_1, N_2, N_3) builds the third subtree of node N'. Finally, algorithm BuildCIT(C, V_{Con}, V_{Rol}, N') returns the concept implicate tree *T* of *C as shown below*.

Theorem 4. Let *C* be an ALC concept that contains only one concept variable *E*, *T* be a tree of *C* built by the algorithm BuildCIT, and C_T be a concept represented by the *T*, then C_T is logically equivalent to *C*, and is one of the concepts among $\perp$, $\top$, *E*, or $\neg E$.

Proof. The concept *C* must be one of the following four concepts: $\perp$, $\top$, *E*, or $\neg E$. If $C = \perp$ or $C = \top$, then the algorithm BuildCIT builds tree *T*, which contains only one node labelling $\perp$ or $\top$. Thus, $C_T = \perp$ or $C_T = \top$, and C_T is logically equivalent to *C*. If *C*=*E*, then the algorithm BuildCIT builds tree *T*, which contains a root node labelling $\perp$ and the first sub-node labelling *E*. Thus, $C_T = \perp \sqcup E = E$, and C_T is logically equivalent to *C*. If *C*= $\neg E$, then the algorithm BuildCIT builds tree *T*, which contains a root node labelling $\perp$ and the second sub-node labelling $\neg E$. Thus, $C_T = \perp \sqcup \neg E = \neg E$, and C_T is logically equivalent to *C*. Therefore, C_T is logically equivalent to *C*, and is one of the concepts among $\perp$, $\top$, *E*, or $\neg E$.

Theorem 5. Let *C* be an ALC concept, V_{Con} be an atomic concept variable set of *C*, V_{Rol} be a role concept variable set of *C*, *T* be a tree of *C* built by the algorithm BuildCIT, and C_T be a concept represented by the *T,* then C_T is logically equivalent to *C*, and each branch of *T* is an implicate of *C*.

Proof. (1) First the logic equivalence was verified by induction on the number *m* of concept variables in *C*, let $V = V_{Con} \cup V_{Rol} = \{E_1, \ldots, E_m\}$, $E_k = A_k \in V_{Con}$ or

$$E_l = QR_j.L_i \in V_{Rol}\ ,\ 1 \le k < l \le m\ ,\ Q \in \{\forall, \exists\}.$$

<1> Base case: Let *m*=1, according to theorem 4, then C_T is logically equivalent to *C*.

<2> Inductive hypothesis: It was assuming that the theorem was true for all concepts with almost all m concept variables.

<3> Induction: It was assumed that *C* had *m*+1 concept variables. Let E_i be any concept variable of *V*, $E_i \in V_{Con}$ or $E_i \in V_{Rol}$,$1 \le i \le m$, now assuming that E_1 is an atomic concept variable form V_{Con}, then, it must be proved that:

$$\begin{aligned} C \equiv C_T = {} & (E_1 \sqcup C_{\mathrm{BuildCIT}(C[\perp/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1)}) \\ & \sqcap (\neg E_1 \sqcup C_{\mathrm{BuildCIT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_2)}) \\ & \sqcap (C_{\mathrm{BuildCIT}(C[\perp/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1) \cap \mathrm{BuildCIT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_2)}) \end{aligned}$$

According to the inductive hypothesis, we obtain,

$$C[\perp / E_1] \equiv C_{\mathrm{BuildCIT}(C[\perp/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1)}\ ,$$

$$C[\top / E_1] \equiv C_{\mathrm{BuildCIT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1)}\ ,$$

$$\begin{aligned} & C[\perp / E_1] \sqcup C[\top / E_1] \\ & \equiv C_{\mathrm{BuildCIT}(C[\perp/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1) \cap \mathrm{BuildCIT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_2)} \end{aligned} .$$

So,

$$\begin{aligned} C_T = {} & (E_1 \sqcup C[\perp / E_1]) \sqcap (\neg E_1 \sqcup C[\top / E_1]) \\ & \sqcap (C[\perp / E_1] \sqcup C[\top / E_1]) \end{aligned} .$$

Let $I = < \Delta^I, \bullet^I >$ be any model of concept *C*, so $C^I \neq \varnothing$, and there exists an individual *a* such that $a \in C^I$ .Following are the two cases of the individual *a*.

Case 1, supposing $a \in E_1^I$, hence, $a \in (C[\top / E_1])^I$, and $a \in (C[\perp / E_1] \sqcup C[\top / E_1])^I$ .Therefore, $a \in (C_T)^I$. Thus, $C^I \subseteq (C_T)^I$, and $C \sqsubseteq C_T$.

Case 2, supposing $a \notin E_1^I$, hence $a \in (\neg E_1)^I$, $a \in (C[\perp / E_1])^I$, and $a \in (C[\perp / E_1] \sqcup C[\top / E_1])^I$ .therefore, $a \in (C_T)^I$. Thus, $C^I \subseteq (C_T)^I$, and $C \sqsubseteq C_T$.

Therefore, $C \sqsubseteq C_T$ suggesting that $C_T \sqsubseteq C$ is similar. Hence, $C \equiv C_T$, that is to say, C_T is logically equivalent to *C*.

(2) It was shown that each branch of *T* is an implicate of *C*. According to the distributive laws of description logic that is similar to the distributive laws of proposition logic, C_T is logically equivalent to the concept conjunction of the labels of its branches. Moreover, due to the interpretation of concept conjunction, each branch of *T* is an implicate of *C*.

Theorem 6. Let *C* be the ALC concept, V_{Con} be an atomic concept variable set of C, and V_{Rol} be a role concept variable set of C. Then algorithm BuildCIT is valid and complete.

Proof. (1) First, the validity of the algorithm was proved. The algorithm BuildCIT built a tree *T*, and according to theorem 5, each branch of *T* was an implicate of *C*, thereby making T a concept implicate tree of *C*. Thus, the algorithm BuildCIT was proved to be valid.

(2) Now, the complete algorithm is explained below. For any concept *C*, the algorithm BuildCIT can build a corresponding tree *T*, and there does not exist a concept without the corresponding tree. Thus, the algorithm BuildCIT is complete.

4. TRACTABLE QUERYING

In this section, for any concept represented by a concept implicate tree, the queries are computable in the linear time depending on the size of the query.

Let *C* be any ALC concept and *T* be a concept implicate tree of *C*. There are three queries for ALC concepts, satisfiability-testing, tautology-testing, and subsumption-testing.

Considering the satisfiability-testing, if *T* contains only one node that is labelled as $\perp$, then *C* is characterized with unsatisfiability, otherwise with satisfiability. With regard to the tautology-testing, if *T* contains only one node that is labelled as $\top$, then *C* has tautology, otherwise *C* has no tautology. Obviously, these two queries can be addressed in the linear time.

In order to test subsumption between the two concepts, the paper provides some theorems as follows.

Let *cl* be a clausal concept with a concept literal *L* or a role concept literal *L*. Then, *cl*/{*L*} denotes a new clausal concept that deletes the literal *L* from *cl*. The prefix of a clausal concept $cl = L_1 \sqcup L_2 \sqcup \cdots \sqcup L_s$ is a clausal concept of the form $cl' = L_1 \sqcup L_2 \sqcup \cdots \sqcup L_t$, $0 \le t \le s$. If *t*=0, then the prefix is $\perp$.

Theorem 7. Let *C* be an ALC concept, and *cl* be an implicate of *C* with a literal *L*. Then $(cl / \{L\}) \in \mathrm{Cimp}(C[\perp / L])$.

Proof. Let $I = <\Delta^I, \bullet^I>$ be any model of concept $C[\perp / L]$, then $(C[\perp / L])^I \neq \varnothing$. Now, *I* is extended to $I' = <\Delta^{I'}, \bullet^{I'}>$ by setting $\Delta^I = \Delta^{I'}$, $L^{I'} = \varnothing$, then I ' is a model of concept *C*. Therefore,, $(C[\perp / L])^I = C^{I'}$. Since *cl* be an implicate of concept *C*, hence, $C^{I'} \subseteq cl^{I'}$. Moreover, $cl^{I'} = (cl / \{L\})^I$. Thus, $(C[\perp / L])^I \subseteq (cl / \{L\})^I$, and $C[\perp / L] \gneq (cl / \{L\})$. Therefore,

$$(cl / \{L\}) \in \mathrm{Cimp}(C[\perp / L]).$$

Theorem 8. Let *C* be an ALC concept, V_{Con} be an atomic concept variable set of *C*, V_{Rol} be a role concept variable set of *C*, *T* be a tree of *C* built by the algorithm BuildCIT, and clausal concept *cl* be an implicate of *C*. Then there is a unique prefix of *cl* that is a branch of *T*.

Proof. It was proved that there is a unique prefix of *cl*, which is a branch of *T*. By induction on the number *m* of concept variables in *C*, let $V = V_{Con} \cup V_{Rol} = \{E_1, \ldots, E_m\}$, $E_k = A_k \in V_{Con}$ or $E_l = QR_j.L_i \in V_{Rol}$, $1 \le k < l \le m$.

1) Base case: Theorem 4 considers the case *m*=1.

2) Inductive hypothesis: It was assumed that the theorem was true for all concepts with almost all *m* concept variables.

3) Induction: Assuming that *C* has *m*+1 concept variables. Let

$$cl = L_{d_1} \sqcup L_{d_2} \sqcup \cdots \sqcup L_{d_s}$$

be an implicate of *C*, where

L_{d_i} is either E_{d_i} or $\neg E_{d_i}$, and $d_1 < d_2 < \cdots < d_s$.

Therefore, it must be proved that there is a unique prefix of *cl* that is a branch of *T*. Let E_i be any concept variable of *V*, $E_i \in V_{Con}$ or $E_i \in V_{Rol}$, $1 \le i \le m$, now assuming that E_1 is an atomic concept variable of the form V_C, then it must be proved that there is a unique prefix of *cl* that is a branch of C_T, which is the concept

$$\begin{aligned} C_T = & (E_1 \sqcup C_{\mathrm{BuildCIT}(C[\perp / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_1)}) \\ & \sqcap (\neg E_1 \sqcup C_{\mathrm{BuildCIT}(C[\top / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_2)}) \\ & \sqcap (C_{\mathrm{BuildCIT}(C[\perp / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_1) \cap \mathrm{BuildCIT}(C[\top / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_2)}) \end{aligned} .$$

By the inductive hypothesis, there is a unique prefix of *cl* that is a branch of the intersection of two subtrees $\mathrm{BuildCIT}(C[\perp / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_1)$ and $\mathrm{BuildCIT}(C[\top / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_2)$. In this case, the theorem is true for the third branch of C_T. Moreover, if $d_1 > 1$, then nothing is needed to prove, therefore, assuming $d_1 = 1$. L_1 is either E_1 or ¬E_1; these are the two cases.

Case 1: Assuming that L_1=E_1, then according to Theorem 7, *cl*/{E_1} is an implicate of $C[\perp / E_1]$. Moreover, by the inductive hypothesis, there is a unique prefix *G* of *cl*/{E_1} that is a branch of $\mathrm{BuildCTT}(C[\perp / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_1)$. Therefore, $H = E_1 \sqcup G$ is a prefix of *cl* that is a branch of *T*. Now, it is to be proved that *H* is a unique prefix of *cl* by contradiction. Assuming that H' is another prefix of *cl* that is a branch of *T*. Let $H' = E_1 \sqcup G'$, then G' is a prefix of *cl*/{E_1} that is a branch of $\mathrm{BuildCTT}(C[\perp / E_1], V_{Con} - \{E_1\}, V_{Rol}, N_1)$. However, *G* is a unique prefix of *cl*/{E_1} by the inductive hypothesis, therefore, $G' = G$, $H' = H$. Therefore, *H* is a unique prefix of *cl* that is a branch of *T*.

Case 2: Assuming that L_1=¬E_1, then according to Theorem 7, *cl*/{¬E_1} is an implicate of $C[\perp / \neg E_1]$ that is equivalent to $C[\top/E_1]$. Moreover, by the inductive hypothesis, there is a unique prefix G of $cl/\{\neg E_1\}$ that is a branch of $\mathrm{BuildCTT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1)$. Therefore, $H = \neg E_1 \sqcup G$ is a prefix of *cl* that is a branch of *T*. Now, it is to be proved that *H* is a unique prefix of *cl* by contradiction. Assuming that H' is another prefix of *cl* that is a branch of *T,* let $H' = \neg E_1 \sqcup G'$, then G' is a prefix of $cl/\{\neg E_1\}$ that is a branch of $\mathrm{BuildCTT}(C[\top/E_1], V_{Con}-\{E_1\}, V_{Rol}, N_1)$. However, *G* is a unique prefix of $cl/\{\neg E_1\}$ by the inductive hypothesis, so $G' = G$, $H' = H$. Therefore, *H* is a unique prefix of *cl* that is a branch of *T*.

Above all, there is a unique prefix of *cl* that is a branch of *T*.

Theorem 9. Let *C* be an ALC concept, V_{Con} be an atomic concept variable set of *C*, V_{Rol} be a role concept variable set of *C*, and *T* be a concept implicate tree of *C*. Then every prime implicate of *C* is a branch of *T*.

Proof. According to Theorem 8, the prefix of an implicate is unique. Thus, it is obvious that the conclusion holds true.

In example 2, all prime implicates of concept C are

$A_1 \sqcup A_2 \sqcup \forall R_2.\neg A_4$, $A_1 \lneqq \forall R_1.\exists R_2.A_5$,

$\neg A_2 \sqcup \forall R_1.\exists R_2.A_5$.

Considering the concept implicate tree T, they both are the branches of T.

Theorem 10. Let *C* be an ALC concept, V_{Con} be an atomic concept variable set of *C*, V_{Rol} be a role concept variable set of *C*, and *T* be a concept implicate tree of *C*. Then every subsuming implicate (including any prime implicate) of a branch of T contains the literal, labelling the leaf of that branch.

Proof. According to Theorem 8, it is obvious that this conclusion holds true.

In example 2, an implicate $A_1 \sqcup A_2 \sqcup \forall R_2.\neg A_4$ contains the literal $\forall R_2.\neg A_4$, which is a label of the leaf of a branch of T. Moreover, all other implicates of C in T, for example are,

$A_1 \sqcup A_2 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup A_2 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \neg A_2 \bigcup \forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \neg A_2 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \neg A_2 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$A_1 \sqcup \forall R_1.\exists R_2.A_5$,

$\neg A_1 \sqcup \neg A_2 \sqcup \forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$\neg A_1 \sqcup \neg A_2 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$\neg A_1 \sqcup \neg A_2 \sqcup \forall R_1.\exists R_2.A_5$,

$\neg A_2 \sqcup \forall R_2.\neg A_4 \sqcup \forall R_1.\exists R_2.A_5$,

$\neg A_2 \sqcup \exists R_2.A_4 \sqcup \forall R_1.\exists R_2.A_5$,

and $\neg A_2 \sqcup \forall R_1.\exists R_2.A_5$, contain the literal $\forall R_1.\exists R_2.A_5$, which is a label of the leaf of a branch of *T*.

Based on the above theorems, if clausal concept $cl = L_1 \sqcup L_2 \sqcup \cdots \sqcup L_d$ is an implicate of concept *C*, and *T* is a concept implicate tree of *C*, then there is a unique prefix $cl' = L_1 \sqcup \cdots \sqcup L_t$ of *cl* that is a branch of *T*, $1 \le t \le d$, and each literal L_i is a label of that branch, $1 \le i \le t$, and L_t is a label of the leaf node. Therefore, the study presents the algorithm Subsume as shown in Fig. (**5**). The main idea is to determine whether *cl* is an implicate of *C* if there exists a branch that labelled the prefix of *cl*.

```
Algorithm Subsume
Input: concept implicate tree $T$ of concept $C$, clausal concept $cl$;
Output: Yes, if $C \sqsubseteq cl$; No, if $C \neq cl$.
1.  If $cl = \top$, then return Yes.
2.  If tree $T$ has only one node labelled $\perp$, then return Yes.
3.  If tree $T$ has only one onde labelled $\top$, then return No.
4.  If $cl = \perp$, then return No.
5.  For each literal $L_i$ of $cl$
        If there exists a branch $w$ of $T$, such that $L_1 \sqcup \cdots \sqcup L_i$ is a
          label of $w$ and $L_i$ is a label of leaf node of $w$,
        then return Yes.
6.  Return No.
```

Fig. (5). Algorithm Subsume.

According to the algorithm, it is obvious that the subsumption-testing can be done by traversing a single branch. Therefore, the time complexity is linear depending on the size of the query, but not on the size of *T*. This is an important property of the proposed method.

Theorem 11. Let *C* be an ALC concept, *T* be a concept implicate tree of *C*, and *cl* be a clausal concept. Then it can be decided in the linear time in |*cl*| whether $C \sqsubseteq cl$, |*cl*| denotes the number of all literals in *cl*.

Proof. Considering the algorithm Subsume, it is obvious that the first four steps of the algorithm can be done in linear time. For the fifth step, the algorithm detects all the literals in *cl* to decide whether $C \sqsubseteq cl$. Therefore, determining whether $C \sqsubseteq cl$ can be done in liner time in |*cl*|.

Above all, three queries for ALC concepts, satisfiability-testing, tautology-testing, and subsumption-testing, can be done in linear time depending on the size of the query.

CONCLUSION

In this paper, knowledge compilation for description logic was presented based on the concept implicate tree. Firstly, the concept implicate tree was defined for the ALC concept. Moreover, the study also provided an algorithm to translate the arbitrary ALC concept into an equivalent concept implicate tree. Finally, it was proved that satisfiability-testing, autology-testing and subsumption-testing were computable in linear time with respect to the concept implicate tree. It was concluded that any query can be done in linear time based on the size of the query, regardless of the size of the concept implicate tree. In other words, the proposed method is an effective method to deal with knowledge compilation for description logic.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work was partially supported by Liaoning Province Natural Science Fund Project of China (2015020034), and National Natural Science Foundation of China (61272171), and the Fundamental Research Funds for the Central Universities of China (3132015044).

REFERENCES

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

[2] Y. Jiang, Y. Tang, J. Wang, and S. Zhou, "A semantic web oriented description logic", *Pattern Recognition and Artificial Intelligence*, vol. 20, pp. 48-54, 2007.

[3] F. Baader, I. Horrocks, and U. Sattler, "Description Logics for the Semantic Web", *KI - Künstliche Intelligenz*, vol. 16, pp. 57-59, 2002.

[4] F. Baader, I. Horrocks, and U. Sattler, *Description Logics*. In: S. Staab, and R. Studer, Eds., Handbook on Ontologies, International Handbooks on Information Systems, Springer, 2004, pp. 3-28.

[5] D. Berardi, D. Calvanese, and G. D. Giacomo, "Reasoning on UML class diagrams", *Artificial Intelligence*, vol. 168, pp. 70-118, 2005.

[6] M. Schmidt-Schauß, and G. Smolka, "Attributive concept descriptions with complements", *Artificial Intelligence*, vol. 48, pp. 1-26, 1991.

[7] F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt, "The complexity of concept languges", *Information and Computation*, vol. 134, pp. 1-58, 1997.

[8] I. Horrocks, and U. Sattler U, "A description logic with transitive and inverse roles and role hierarchies", *Journal of Logic and Computation*, vol. 9, pp. 385-410, 1999.

[9] I. Horrocks, U. Sattler, and S. Tobies, "Practical reasoning for very expressive description logics", *Logic Journal of the IGPL*, vol. 8, pp. 239-264, 2000.

[10] I. Horrocks, and U. Sattler, "A Tableau Decision Procedure for SHOIQ", In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, Morgan Kaufmann, Los Altos, 2005, pp. 448-453.

[11] I. Horrocks, and U. Sattler, "A tableau decision procedure for SHOIQ", *Journal of Automated Reasoning*, vol. 39, pp. 249-276, 2007.

[12] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ", In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, Menlo Park, California: AAAI Press, 2006, pp. 57-67.

[13] A. Darwiche, and P. Marquis, "A knowledge compilation map", *Journal of Artificial Intelligence Research*, vol. 17, pp. 229-264, 2002.

[14] A. Darwiche, and P. Marquis, "A perspective on knowledge compilation", In: *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2001, pp. 175-182.

[15] M. Cadoli, and M. Donini, "A survey on knowledge compilation", *AI Communications*, vol. 10, pp. 137-150, 1997.

[16] P. Marquis, "Knowledge compilation using theory prime implicates", In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995, pp. 837- 843.

[17] A. Darwiche, "Decomposable negation normal form", *Journal of the ACM*, vol. 48, pp. 608-647, 2001.

[18] N. V. Murray, and E. Rosenthal, "Efficient query processing with reduced implicate tries", *Journal of Automation Reasoning*, vol. 38, pp. 155-172, 2007.

[19] N. V. Murray, and E. Rosenthal, "Linear response time for implicate and implicant queries", Knowledge and Information System, vol. 22, pp. 287-317, 2010.

[20] N. V. Murray, and Erik Rosenthal, "Reduced implicate tries with updates". *Journal of Logic and Computation*, vol. 20, pp. 261-281, 2010.

[21] B. Selman, and H. Kautz, "Knowledge compilation and theory approximation", *Journal of the ACM*, vol. 43, pp. 193-224, 1996.

[22] U. Furbach, and C. Obermaier, "Knowledge compilation for description logics", In: *Proceedings of the 3rd Workshop on Knowledge Engineering and Software Engineering (KESE),* 2007.

[23] U. Furbach, and C. Obermaier, "Precompiling ALC tboxes and query answering", In: *Proceedings of the 4th Workshop on Contexts and Ontologies (C&O-2008) at the 18th European Conference on Artificial Intelligence*, Patras, Greece, 2008, pp. 11-15.

[24] U. Furbach, H. Günther, and C. Obermaier, "A knowledge compilation technique for ALC TBoxes", In: *Proceedings of the 22nd International Florida Artificial Intelligence Research Society Conference*, Sanibel Island, Florida, USA, 2009, pp. 39-44.

[25] M. Bienvenu, "Prime implicate normal form for ALC concepts", In: *Proceedings of the 23rd National Conference on Artificial Intelligence*, 2008, pp. 412-417.

[26] T. Zou, L. Liu, and S. Lv, "Knowledge compilation for description logic based on concept extension rule", *Journal of Computational Information Systems*, vol. 8, pp. 2409-2416, 2012.

Received: June 10, 2015 Revised: July 29, 2015 Accepted: August 15, 2015