Deciding the SHOQ(D)-Satisfiability with a Fully Tiered Clause Group

Huamao Gu, Jinqin Shi and Xun Wang
Zhejiang Gongshang University/Institute of Artificial Intelligence, Hangzhou, China
Email:{ghmsjq, shijinqin, wx}@mail.zjgsu.edu.cn

***Abstract*—SHOQ(D) is one of the fundamental theories In Description Logics due to its support to concrete datatypes and named individuals. At present, deciding the satisfiability of SHOQ(D)-concepts is mainly completed by enhancing Tableau algorithm with blocking. However, there is still much to be desired in performance as there are tremendous description overlaps in completion forest, thus causing great spatial waste as a result. To tackle this problem, this paper presented a new approach to check the satisfiability of acyclic SHOQ(D)-concepts——FTC(Fully Tiered Clauses) algorithm. This calculus can make a direct judgement on the satisfiability of acyclic SHOQ(D)-concept by translating its description into a fully tiered clause group whose satisfiability is directly available, and reusing clauses to block unnecessary extensions. FTC algorithm eliminates description overlaps to the largest extent as it works on concept description directly. Therefore, FTC algorithm has notably better performance than Tableau by saving a lot of spatial costs.**

***Index Terms*— Satisfiability, Concept Clause, SHOQ(D), Fully Tiered Clauses**

I. Introduction

Semantic WEB aims at giving WEB content clear semantics and making computer understand and process them automatically. Amid it, Ontology is a key component which is mainly employed to describe different WEB resources and their relations and cooperation. However, no matter the early WEB Ontology languages OIL[1] and DAML+OIL[2] or the updated OWL[3], they are all based on description logics (DLs). Besides, with the increasing requirements on intelligentizing information, description logics will play growingly important roles in other applications. And in the researches on description logics, reasoning is apparently a key issue. Then in reasoning, concept satisfiability is the most fundamental one (the other basic issue, concept subsumption, can convert into concept satisfiability in most cases).

Comparatively speaking, although the reduction based reasoning[4] can be used to decide the concept satisfiability, its advantage is embodied by query answering over large *ABoxes*. As far as the satisfiability of a specific concept is concerned, its performance is far less than Tableaux algorithm in that it hires a reduction process additionally. As for the structural subsumption, it has even no practicality since it just applies to much simple DLs such as FL_0. Under such circumstance, the research on concept satisfiability mainly goes around the classic concept satisfiability algorithm Tableaux. They either add new operators or functions to existent description logics systems and extend Tableaux to certain extent to fit new systems[5,6,7,8] or integrate description logics with other fields and still provide modified versions of Tableaux to support reasoning[9,10,11,12,13].

Actually, though Tableau proved practical in practice, it has some irresolvable drawbacks induced by its mechanism. For a specific concept, Tableau algorithm unfolds its description gradually and at last obtains all the descriptions that should be entailed. In the process, each unfolding will produce an overlapped description to the concept unfolded completely or partially. For example, let $\boldsymbol{L}(x)$ be a node labeling in a completion tree of Tableau. If $(A\cap B)\in \boldsymbol{L}(x)$, then there will be $A\in \boldsymbol{L}(x)$ and $B\in \boldsymbol{L}(x)$ according to ∩-rule, that is to say, $\{A\cap B, A, B\}\in \boldsymbol{L}(x)$. This is complete overlap. Similarly, If $A\cup B\in \boldsymbol{L}(x)$, then there will be $A\in \boldsymbol{L}(x)$ or $B\in \boldsymbol{L}(x)$ according to ∪-rule. This is partial overlap. As operators ∩, ∪ are nearly most frequently used in describing concepts, then such overlap is very serious in most cases.

The FTC (*Fully Tiered Clauses*) algorithm presented in this paper is a novel algorithm which intends to solve this problem of severe description overlaps which waste tremendous space in Tableau. Its basic idea is to translate the acyclic concept description[14] into specially organized clauses. It doesn't change model features along the reorganizations, in other words, if the latter concept produced by reorganizing is satisfiable, then the concept before the reorganizing is also satisfiable. Therefore, the satisfiability of a concept can be decided by enumerating and testing all the possible variant clauses directly. This processing mode doesn't cause description overlaps, thus saves much more space compared with Tableau, which makes FTC gain linear space costs in many cases. For cyclically defined concepts, one may translate them into general inclusion axioms thus no long appearing as defined concepts, or take advantage of the connection with propositional dynamic logic and adopt μ-calculus to realize reasoning, both being not in the scope of this paper. Therefore, all the concepts in the rest of this paper are acyclic implicitly.

Manuscript received Dec. 1, 2010; revised Jan. 5, 2011; accepted Jan. 12, 2011.
Corresponding author: wx@mail.zjgsu.edu.cn

doi:10.4304/jsw.7.2.322-331

Description logic *SHOQ*(**D**)[15] becomes a cornerstone of WEB Ontology language for its support to enumerating instances and allowing datatypes and values to construct concept which are two key properties in WEB Ontology. Not only the OIL frames could easily be mapped to equivalent axioms in the *SHOQ*(**D**)[16], but also the DAML＋OIL which is equivalent to *SHOIQ*(D) [17], and the OWL DL which is equivalent to SHOIN(**D**) [16], are all just slightly modified versions of *SHOQ*(**D**). Besides, reasoning with inverse roles is known to be difficult and/or highly intractable when combined with either *concrete datatypes* [18] or *named individuals* [19], while FTC algorithm is still on its early stage. For above reasons, we take *SHOQ*(**D**) as the carrier for FTC algorithm. Of course, the implementation of FTC algorithm in *SHOQ*(**D**) has still much significance both in theory and practice because of the important role of *SHOQ*(**D**) for OWL researches and the much higher utilization rate of concrete datatypes or named individuals against that of inverse roles [15].

This paper firstly introduces the syntax and semantics of *SHOQ*(**D**) briefly, and then discusses the process, correctness, and performance of FTC in detail. A performance comparison between Tableaux and FTC is given lastly.

II. SHOQ(D) SYNTAX AND SEMANTICS

Description logic *SHOQ*(**D**) takes the need of ontology representation into full consideration and extends the DL *SHQ* with concrete datatypes and named individuals, which makes it an ontology-oriented description logic with strong expressiveness. Following are brief introductions to *SHOQ*(**D**).

Definition 1. Let **D** be a set of *concrete datatypes*, *NC*, $NR=N_R^A \uplus N_R^D$, and *NI* be disjoint sets of *concept names*, abstract and concrete *role names*, and *individual names*. Then, the set of *SHOQ*(**D**)-concepts is the smallest set such that:

1. Each concept name $C\in NC$ is a *SHOQ*(**D**)-concept;
2. For each individual name $o\in NI$, $\{o\}$ is a *SHOQ*(**D**)-concept;
3. For C and D concepts, $R\in N_R^A$ an abstract role, $F\in N_R^D$ a concrete role, $S\in N_R^A$ a simple role[15] (abstract role that is not transitive and for each role $R'\subseteq S$, R' is not transitive), $d\in$ **D** a datatype, $n\in IN$(natural number), then $(C\cup D)$, $(C\cap D)$, $(\neg C)$, d, $(\neg d)$, $(\forall R.C)$, $(\exists R.C)$, $(\geq nS.C)$, $(\leq nS.C)$, $(\forall F.d)$, $(\exists F.d)$ are all *SHOQ*(**D**)-concepts.

Besides, $\top$ is called universal concept which subsumes any concepts in domain, while $\perp$ is called bottom concept which contains nothing and is subsumed by any concepts. If $A\in NC$ can only appear possibly at the right sides of some concept definitions, namely, there is no definition for A and thus it can not be composed of other concepts, then A is called primitive concept[14]. And we confine number restrictions (including atmost restriction and atleast restriction) to simple roles to guarantee a decidable logic[20].

Definition 2. For R and S roles, the statement of form $R\subseteq S$ is called *role inclusion axiom*. For a set of role inclusion axioms **R**, we call $\mathbf{R}^+=(\mathbf{R},\sqsubseteq^*)$ a role hierarchy where $\sqsubseteq^*$ is the transitive reflexive closure of $\subseteq$. Besides, if an abstract role R is transitive, we mark *Trans*(R). Note that concrete roles have nothing to do with transitivity, in other words, they are all not transitive.

Definition 3. *SHOQ*(**D**)-Interpretation $I=(\triangle^I,\cdot^I)$ consists of a non-empty set $\triangle^I$(*interpretation domain*) and a mapping $\cdot^I$. And set $\triangle^{\mathbf{D}}$ is the *concrete domain* for all concrete datatypes and is disjoint from $\triangle^I$. Mapping $\cdot^I$ maps different concept descriptions according to Fig. 1. Besides, for R and S roles, if $R\sqsubseteq^* S$, then $<x,y>\in R^I$ implies $<x,y>\in S^I$. And for *Trans*(R), if $<x,y>\in R^I$, $<y,z>\in R^I$, then there is $<x,z>\in R^I$.

Constructor	Syntax	Semantics
primitive concept	A	$A^I\subseteq\triangle^I$
abstract role	R	$R^I\subseteq\triangle^I\times\triangle^I$
concrete role	F	$F^I\subseteq\triangle^I\times\triangle^D$
named individual	$\{o\}$	$\{o\}^I\subseteq\triangle^I,\#\{o\}^I=1$
datatype	d	$d^D\subseteq\triangle^D$
conjunction	$(C\cap D)$	$(C\cap D)^I=C^I\cap D^I$
disjunction	$(C\cup D)$	$(C\cup D)^I=C^I\cup D^I$
negation	$\neg C$	$\neg C^I=\triangle^I\setminus C^I$
	$\neg d$	$\neg d^D=\triangle^D\setminus d^D$
exists restriction	$(\exists R.C)$	$(\exists R.C)^I=\{x\in\triangle^I \mid$ there is one $y\in\triangle^I$ satisfying $<x,y>\in R^I$ and $y\in C^I\}$
value restriction	$(\forall R.C)$	$(\forall R.C)^I=\{x\in\triangle^I \mid$ for each $y\in\triangle^I$, $<x,y>\in R^I$ implies $y\in C^I\}$
atleast restriction	$(\geqslant nS.C)$	$(\geqslant nS.C)^I=\{x\in\triangle^I \mid \#\{<x,y> \mid <x,y>\in S^I\wedge y\in C^I\}\geq n\}$
atmost restriction	$(\leqslant nS.C)$	$(\leqslant nS.C)^I=\{x\in\triangle^I \mid \#\{<x,y> \mid <x,y>\in S^I\wedge y\in C^I\}\leq n\}$
datatype exists	$(\exists F.d)$	$(\exists F.d)^I=\{x\in\triangle^I \mid$ there is one $y\in\triangle^{\mathbf{D}}$, satisfying $<x,y>\in F^I$ and $y\in d^{\mathbf{D}}\}$
datatype value	$(\forall F.d)$	$(\forall F.d)^I=\{x\in\triangle^I \mid$ for each $y\in\triangle^{\mathbf{D}}$, $<x,y>\in F^I$ implies $y\in d^{\mathbf{D}}\}$

Figure 1. Syntax and semantics of *SHOQ*(D), # denoting set cardinality.

With negation operator $\neg$ in *SHOQ*(**D**), subsumption reasoning and concept satisfiability can be reduced to each other. This is because $C\subseteq D$ holds if and only if $C\cap\neg D$ is unsatisfiable. On the other hand, C is satisfiable if and only if $C\subseteq\perp$ is untenable. Hence, we just talk about satisfiability in the following discussion.

III. THE FTC ALGORITHM FOR SHOQ(D)

The FTC adopts idea quite different from Tableau that it builds a group of fully tiered clauses on concept to decide its satisfiability directly. This section firstly provides relevant definitions and then fully describes the FTC algorithm in terms of processing, soundness, and completeness.

A. Relevant Definitions

Definition 4 (**Literal** and **Restricting Concept**). Let A be primitive concept, o a named individual, C a concept, R an abstract role, S a simple role, F a concrete role, d a datatype, n positive integer, then expressions of the forms A, $\neg A$, $\{o\}$, $\exists R.C$, $\forall R.C$, $(\geq nS.C)$, $(\leq nS.C)$, $(\forall F.d)$, $(\exists F.d)$, d, $\neg d$, $\top$ and $\bot$ are *Literals*. Besides, $\exists R.C$, $\forall R.C$ are called $\exists/\forall$ *role literals* (or briefly *role literals*) or more specifically, $\exists/\forall R$ role literals, with C being the *restricting concept* of role literals $\exists R.C$ and $\forall R.C$. In this paper, literals are denoted with capital letters such as X, Y. And the role of a role literal X is signified by ***Rol***(X).

Definition 5 (**Clause**). *Clauses* are the descriptions satisfying: A single literal is a clause; A description composed of two or more literals put together with $\cap$ operator is a clause. In this paper, clauses are denoted with lowercased letters such as x, y. The restricting concept of a $\exists$ role literal will also form a clause in FTC algorithm. This clause is called *restricting concept clause*. Furthermore, the restricting concept of a role literal in a clause is also called the *restricting concept* of this clause. Conversely, this clause is called the father clause of the restricting concept (clause), while the role is called father role of the restricting concept (clause) correspondingly. For a father clause x and its child clause y, x can be represented by y as ***Upper***(y) and the $\exists$ role literal whose restricting concept is clause y can be represented as *y.Letter*. Undoubtedly, *y.Letter* is one literal of clause x. A literal X appeared in clause x can be represented as $X \in x$.

Definition 6 (**Disjunctive Normal Form**). the description consisting of one clause, or two or more clauses concatenated with $\cup$ is *disjunctive normal form* (**DNF**).

Definition 7 (**Fully Tiered Clause**). If a concept description is in the form of clause, and it contains no $\exists$ role literal or all its restricting concepts are also fully tiered clauses, then this concept is called *Fully Tiered Clause* (FTC). Note that not only the abbreviation FTC is used as the name of algorithm but also its italic form *FTC* stands for a description with the form of fully tiered clause.

Definition 8 (**$\exists$ Labeling Set, $\exists$ Original Restricting Concept**). Each $\exists$ simple role literal X has a labeling set ***B*** to reserve concepts incurred by processing number restrictions, denoted as ***B***(X). While each $\exists$ transitive role literal Y has a data structure ***E*** to keep its original restricting concept, denoted as ***E***(Y).

Definition 9 (**Path**). For a clause sequence $x_1, x_2, \ldots, x_n$ in a *FTC*, x_i is the father clause of x_{i+1} (for $i \geq 1$ and $i \leq n-1$), we say such sequence is a *path*.

Definition 10 (**Clause reuse**). for a path $x_1, x_2, \ldots, x_n$, if there are two clauses x_i, x_j ($j>i$) satisfying:

Rol(x_i.*Letter*)= ***Rol***(x_j.*Letter*)= R, and Trans(R),
E(x_i.*Letter*)= ***E*** (x_j.*Letter*), and
Upper(x_j)/$R \sqsubseteq$ ***Upper***(x_i)/R.

Note that x/R is defined as: $x/R = \{ C \mid \forall S.C \in x$, and $R \,\underline{\circledast}\, S \}$

Then x_i is a clause reusable by x_j. Draw a reuse directed line from x_j.*Letter* to the x_i to signify that x_i can be used as the restricting concept clause of x_j.*Letter*, while we needn't do anything more to x_j.

Definition 11 (**Concept Description Group and FTC Group**). If the description of concept C contains named individuals and is satisfiable at the same time, then it usually requires these named individuals to be instances of some certain concepts. In such case, each named individual has a corresponding concept description (for a specific named individual o, it is denoted as $Des(o)$). All these concept descriptions form a *concept description group*, and the description of C is then called *principal concept description* of this group. This description group will form a group of *FTC*s (namely, *FTC* group) after applying the FTC algorithm. The same way, the *FTC* corresponding to principal concept description is *principal FTC*.

Definition 12 (**Processing Node**). In the course of turning a concept description into the form of *FTC*, we need to transform not only this concept itself but also other restricting concepts at all levels into clauses. Therefore, we term the description being turned into clause as *processing node*.

Definition 13 (**Merging Operation**). *Merging operation* is to merge two role literals into one role literal. Let B, C be two concepts, R, S roles (abstract or concrete), then:

Merging $\exists R.B \cap \forall S.C$ (with $R \,\underline{\circledast}\, S$) into $\exists R.(B \cap C \cap \forall R.C)$ or $\exists R.(B \cap C)$ depending on whether Trans(R) or not, and substitute them for the $\exists R.B$, is called $\forall$-*Merge Operation*. Such merging operation is used to act all $\forall$ role literals in a clause to all corresponding $\exists$ role literals with the same role in the same clause.

Merging $\exists R.B \cap \exists S.C$ (with $R \,\underline{\circledast}\, S$) into $\exists R.(B \cap C)$ and substitute it for both the $\exists R.B$ and $\exists S.C$, is called $\exists$-*Merge Operation*. Such merging operation is used to merge some chosen $\exists$ role literals and reduce their number in a clause to meet the semantic requirement of atmost restriction.

If there is description like $B \cap \{o_1\}$ occurring in clause x, then algorithm transfers description B to concept description of individual o_1 so as to form $Des(o_1) = Des(o_1) \cap B$, and just keeps $\{o_1\}$ unmoved, namely, $x = \{o_1\}$ after operation. If clause x is also the concept description of another named individual, say o_2, then mark $o_1 = o_2$ to stand for the identity of these two individuals. This is *o-Merge Operation*.

Definition 14 (**Clash**). For an unsatisfiable concept, its description should involve some special description fragments which can never be satisfied semantically anyway. They are called *clash*. For a clause, such cases include:

① Bottom concept: $\bot$.

② Concept name clash: $\neg A \cap A$ (A is primitive concept).

③ Datatype clash: Clause contains datatypes $d_1, \ldots d_n$, but $d_1^{D} \cap \ldots \cap d_n^{D} = \varnothing$.

④ Number clash: Clause contains literal $\leq nS.C$ (S simple role, n positive integer), but this clause also contains more than n $\exists R$ *role* literals ($R \,\underline{\circledast}\, S$) $Y_0, Y_1, \ldots Y_n$ satisfying $C \in$ ***B***(Y_i) for all $0 \leq i \leq n$.

⑤ Individual ≠ clash: Clause x contains two ∃ role literals whose restricting concepts only contain same $\{o\}$ (o, a certain named individual), or $\{o_1\}$ and $\{o_2\}$ (o_1 and o_2, two certain named individuals) respectively, while these two literals are marked unable to merge with ≠.

⑥ Individual ¬ clash: For a certain named individual o, there is $\neg\{o\} \in Des(o)$.

If there is any clash occurring in the clause, then mark this concept description with ⊥ and finish the constructing of this *FTC* group. This is because if the current clause is ⊥, then its father clause will be replaced with ⊥ according to $\exists R.\bot \equiv \bot$ and $C \cap \bot \equiv \bot$. Doing so repeatedly, the whole expression will be replaced with ⊥. So does other expressions in this group. However, this doesn't necessarily mean the end of the algorithm; after all, there may be other satisfiable *FTC* group for the input concept.

B. FTC Algorithm Process

Starting from an input concept C_0, we can decide the satisfibility of C_0 after applying the FTC process to it. If at least one clash free *FTC* group can be obtained after the application of FTC algorithm, then C_0 is satisfiable, otherwise, unsatisfiable. The following is a detailed depiction of FTC algorithm according to the process graph shown in Fig. 2:

(1) Pretreatment: Replace all the concept names in C_0 with their corresponding descriptions until there are no concept names except primitive concepts in C_0. Then turn C_0 into negation normal form (NNF), namely, push all negation operators as inward as possible so that negation only occurs in front of primitive concepts. This can be easily done by applying De Morgan's laws and other rules such as $\neg\exists R.C = \forall R.(\neg C)$ and $\neg\forall R.C = \exists R.(\neg C)$, *etc*. the negation of a number restriction can be eliminated by changing the number and direction, for example, $\neg(\leq nR.C) = \geq(n+1)R.C$, and vice versa. After done these, set C_0 as processing node **P**, and at the same time, initialize $Des(o) = \top$ for each named individual o. They are also called distinguished points[15]. At last, initialize two binary relations (i.e. = and ≠) for conserving equivalent individual pairs and ∃ role literal pairs unable to merge respectively) to be empty.

(2) Building DNF: Reorganize the literals in processing node **P** into the form of DNF with relevant laws (association, commutation, distribution, and idempotence, *etc*.) and choose one clause of it (other clauses should be removed). The chosen clause in **P** is also called *current clause*. And the chosen clause of the first processing node for a description is called *top clause*. If there are two same literals occurring at the chosen clause with ≠ relation, then they cannot be merged into one with idempotence. Then, check clashes. Note that the literals defined in definition 4 have enumerated all possible elements to form a *SHOQ*(**D**)-concept. And the only way to form a more complicated concept is to concatenate these elements by operator ∩ and ∪. Besides, both ∩ and ∪ support association, commutation and distribution law which are just needed to build a DNF. Therefore, it is always available to reorganize an acyclic concept into a disjunctive normal form.

If current clause doesn't have any clashes (clash □ and □ usually cannot appear at this step, but may appear in the later steps), then further detect if it contains literals like $\{o\}$. If so, conduct o-Merge operation.

(3) Locating reusable clause: If father role of the current clause x is transitive, then starting from father clause, search the only path to top clause for reusable clause. If find one, then there is no need to process x any more, set ***Upper***(x) as **P** and turn to relocating **P** step.

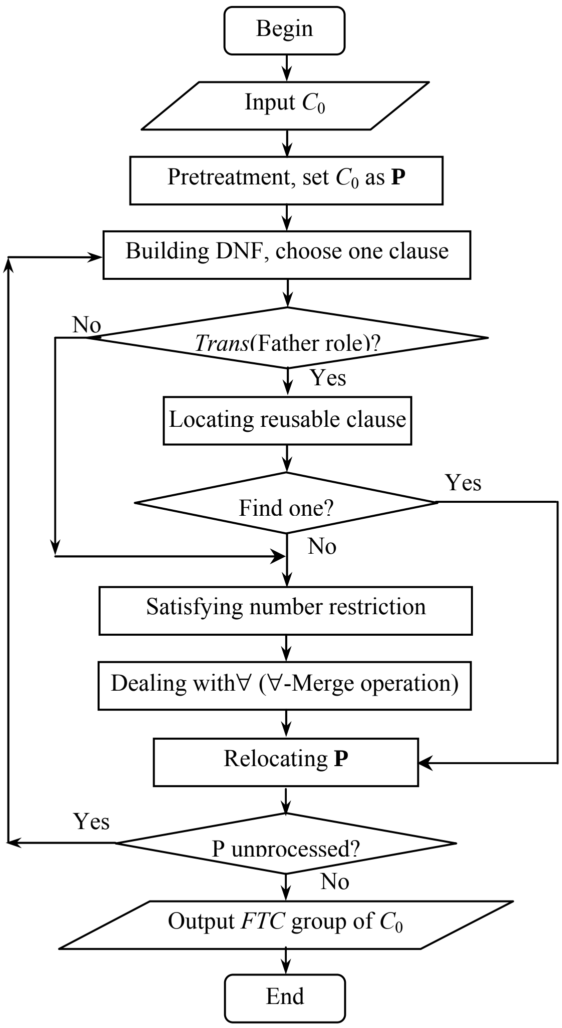

Figure 2. The basic processes of FTC algorithm.
P means processing node.

(4) Satisfying number restriction: For current clause x, if there are literals $\geq nS.C \in x$ or $\leq nS.C \in x$, then process x as follows (R stands for any roles occurring in x and satisfying $R \sqsubseteq S$). Note that if there are other number restriction literals on role S' with $S' \sqsubseteq S$, make sure to process those literals first.

1) Add one of $\{C, \neg C\}$ to the labeling set $\boldsymbol{B}$ of each $\exists R$ role literal, and integrate the added C or $\neg C$ into the restricting concept of corresponding role literal by conjoining the current restricting concept and C (or $\neg C$ respectively) with operator ∩.

2) If there is literal $\geq nS.C \in x$ and the $\#\{\exists R$ role literal $X \mid X \in x$ and $C \in \mathbf{B}(X)\} < n$, append n $(\exists S.C)$s to x and mark these newly-added literals unable to merge into each other with $\neq$. This is $\geq$ *satisfying operation*.

3) If there is literal $\leq nS.C \in x$ and the $\#\{\exists R$ role literal $X \mid X \in x$ and $C \in \mathbf{B}(X)\} > n$, conduct $\exists$-Merge operation on two randomly selected $\exists R$ role literals without $\neq$ relationship. Repeat this operation until $\#\{\exists R$ role literal $X \mid X \in x$ and $C \in \mathbf{B}(X)\} \leq n$. If there are no enough $\exists R$ role literals available for merging due to the $\neq$ relations among them, then clash □ (number clash) occurs and this expression needs to be set to $\perp$. This is $\leq$ *satisfying operation*.

(5) Dealing with $\forall$: For every $\forall S.C \in x$ (x, current clause), apply $\forall S.C$ to each $\exists R.D \in x$ (with $R \sqsubseteq S$) by $\forall$-Merge operation: obtaining newly reorganized literal: $\exists R.(C \sqcap D)$ or $\exists R.(C \sqcap D \sqcap \forall R.C)$ according to the transitivity of R.

(6) Relocating P: If there is no restricting concept unprocessed, then trace backward along father clauses one after another and check the clash □ meanwhile when coming to a new clause, until encountering a clause with restricting concepts unprocessed or the top clause. Set the relocated clause processing node **P**. If the newly located **P** has still no unprocessed restricting concept, move **P** to next distinguished point which is not in form of *FTC* and repeat steps (2), (3), (4), (5), and (6). Make clear that some distinguished points may destroy their *FTC* forms due to the *o*-Merge operation, thus need to be processed further.

If all the concept descriptions in the group are in form of *FTC* and none of them is $\perp$ or there is no possibly available *FTC* group any more, the algorithm ends. The former case means the input concept C_0 is satisfiable, while the later unsatisfiable.

Now, let's take a look at an example. Suppose that there is an online system whose user management module requires: (1) User categories (in ascending order of privilege): *User*, *Manager*, and *Administrator*; (2) *User* cannot grant rights to the ones with higher privilege, but user can grant rights to the ones in the same category and *Administrator* can grant rights to itself; (3) *Grant* is a management action, any doer of management actions should be granted by *Administrators*. Now, concept C_1 represents the users who have business relations with user "LiuGang" and with exactly two users who have business relations with *Managers*. Concept C_0 represents the users with salary higher than 10,000 dollars who are not *Manager* and have business relations with users of C_1 and users being granted by *Manager*. Upon that, C_0 can be expressed formally as $C_0 = \neg A \sqcap \exists S.(\exists S.\{o\} \sqcap \leq 2S.(\exists S.A) \sqcap \geq 2S.(\exists S.A)) \sqcap \exists S.(\exists R.A \sqcap \forall P.(\exists R.B)) \sqcap \exists F.d$; (where A: *Manager*; B: *Administrator*; R: Granted-from; P: Managed-by; S: Have-business-relations-with; F: Salary; d:≥10000; o: "LiuGang"). And now let's work out the *FTC* group of C_0 and gain the judgement of its satisfiability (strikethrough stands for clause which is not needed any more thanks to clause reuse).

For the best of visual effect for expression, we use some intermediate concepts (see $C_1 \sim C_4$) and mark the processing node with [] and underline. Besides, some operating explanations are also given briefly following the concept description (unconcerned steps are omitted). Following is the whole process on C_0.

Intermediate concepts: $C_1 = \exists S.\{o\} \sqcap \leq 2S.(\exists S.A) \sqcap \geq 2S.(\exists S.A)$; $C_2 = \exists R.A \sqcap \forall P.(\exists R.B)$; $C_3 = \exists S.A$; $C_4 = \exists R.B$.

Beginning: Pretreatment stage sets $Des(o) = \top$, and C_0 is set as processing node **P** for C_0 is already in NNF.

$[\underline{\neg A \sqcap \exists S.C_1 \sqcap \exists S.C_2 \sqcap \exists F.d}]$;—Relocating P—>

$\neg A \sqcap \exists S.[\underline{\exists S.\{o\} \sqcap \leq 2S.C_3 \sqcap \geq 2S.C_3}] \sqcap \exists S.C_2 \sqcap \exists F.d$;—Satisfying number restriction:$\leq 2S.C_3$, $\geq 2S.C_3$—>

$\neg A \sqcap \exists S.[\underline{\exists S.(\{o\} \sqcap C_3) \sqcap \leq 2S.C_3 \sqcap \geq 2S.C_3 \sqcap \exists S.C_3}] \sqcap \exists S.C_2 \sqcap \exists F.d$; —Relocating P,*o*-Merge operation—>

$\neg A \sqcap \exists S.(\exists S.[\underline{\{o\}}] \sqcap \leq 2S.C_3 \sqcap \geq 2S.C_3 \sqcap \exists S.C_3) \sqcap \exists S.C_2 \sqcap \exists F.d$; $Des(o) = C_3$; —Relocating P—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.[\underline{\exists R.A \sqcap \forall P.C_4}] \sqcap \exists F.d$; —Dealing with$\forall$:$\forall P.C_4$—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.[\underline{\exists R.(A \sqcap C_4 \sqcap \forall R.C_4) \sqcap \forall P.C_4}] \sqcap \exists F.d$; —Relocating P—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.[\underline{A \sqcap \exists R.B \sqcap \forall R.C_4}] \sqcap \forall P.C_4) \sqcap \exists F.d$; —Dealing with$\forall$: $\forall R.C_4$—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.[\underline{A \sqcap \exists R.(B \sqcap C_4 \sqcap \forall R.C_4) \sqcap \forall R.C_4}] \sqcap \forall P.C_4) \sqcap \exists F.d$; —Relocating P—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.(A \sqcap \exists R.[\underline{B \sqcap \exists R.B \sqcap \forall R.C_4}] \sqcap \forall R.C_4) \sqcap \forall P.C_4) \sqcap \exists F.d$; —Dealing with$\forall$: $\forall R.C_4$—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.(A \sqcap \exists R.[\underline{B \sqcap \exists R.(B \sqcap C_4 \sqcap \forall R.C_4) \sqcap \forall R.C_4}] \sqcap \forall R.C_4) \sqcap \forall P.C_4) \sqcap \exists F.d$; —Relocating P—>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.(A \sqcap \exists R.(B \sqcap \exists R.[\underline{B \sqcap C_4 \sqcap \forall R.C_4}] \sqcap \forall R.C_4) \sqcap \forall R.C_4) \sqcap \forall P.C_4) \sqcap \exists F.d$; —Locating reusable clause —>

$\neg A \sqcap \exists S.(C_1 \sqcap \exists S.C_3) \sqcap \exists S.(\exists R.(A \sqcap \exists R.(B \sqcap \exists R.[\cancel{B \sqcap C_4 \sqcap \forall R.C_4}] \sqcap \forall R.C_4) \sqcap \forall R.C_4) \sqcap \forall P.C_4) \sqcap \exists F.d$;

At last, we gain a *FTC* group:

$C_0' = \neg A \sqcap \exists S.(\exists S.\{o\} \sqcap \leq 2S.(\exists S.A) \sqcap \geq 2S.(\exists S.A) \sqcap \exists S.(\exists S.A)) \sqcap \exists S.(\exists R.(A \sqcap \exists R.(B \sqcap \exists R. \sqcap \forall R.(\exists R.B)) \sqcap \forall R.(\exists R.B)) \sqcap \forall P.(\exists R.B)) \sqcap \exists F.d$; $Des(o) = \exists S.A$. According to this *FTC* group, C_0 is satisfiable.

C. The Semantics of FTC Group

If a concept description can gain a satisfiable *FTC* group after applying FTC algorithm, then we can construct a model of the input concept easily.

We firstly introduce a new mode to represent $\exists$ role literal. Literal $\exists R.C$ can be reshaped into a form of $\exists R.\rightarrow C$ where $\rightarrow$ represents a directed edge connecting two parts of a $\exists$ role literal. We call such form *separated form* of $\exists$ role literal. With a satisfiable *FTC* group, we apply the separated form to all the $\exists$ role literals in the clauses at every level. Then the *FTC* group becomes a multi-tier forest naturally (Of course, when we say it a forest, we neglect the reuse directed lines. Otherwise, it is not a forest). We call such *FTC* group *forested FTC* group upon which clauses in different levels form instance nodes. Of course, the $\exists$ role literal in the clause encased in an instance node contains only the first part, as its restricting concept is scattered in other instance nodes. Note that we just take this form in the descriptions which are not embedded in any $\forall$ role literals. For example, for a description: $\forall R.(A \sqcup \exists S.C)$, the $\exists S.C$ needn't to take the separated form. Based on the forested *FTC* group, do the following work:

If the father role of the clause in an instance node is a concrete role, then mark this node with a value that satisfies all the datatypes in the clause. Otherwise, christen the node a distinct name. For a named individual *o*, *o* is just the name of the node who encases top clause of *Des*(*o*).

Now we employ $\boldsymbol{L}(a, b)=R$ to denote relations between nodes *a* and *b*: (1) *a* and *b* (*b* may be a name or a value here) are connected with $\exists R.\rightarrow$; (2) *a* and *b* are linked with a reuse directed line starting from a $\exists R$ role literal in *a*. We also employ $\boldsymbol{L}(e, o)=S$ to denote the relation between node *e* and named individual *o*, if *e* contains literal like $\exists S.\{o\}$.

Suppose that *R* and *F* are the abstract and concrete role in *FTC* group respectively, *A* is the primitive concept, *C'* is the principal *FTC*. Then the model of *C'*, $I =(\Delta^I, \cdot^I)$ can be built as follows:

Δ^I={ *x* | *x* is the name for a node in forested *FTC* group};
$\Phi(S)$ ={$<x,y>$| There is $\boldsymbol{L}(x,y)=R$ (with $R \sqsubseteq_* S$) in forested *FTC* group};
If Trans(*S*), then $S^I=\Phi(S)+$; otherwise, $S^I=\Phi(S)$;
$\Phi(F)$={$<x,d>$| There is $\boldsymbol{L}(x,d)=F$ in forested *FTC* group};
A^I={ *x* | node containing literal *A* is named *x*};
C'^I={x_0 | x_0 is the name of the node containing top clause of the principal *FTC*.}.

Fig. 3 shows the forested *FTC* group of C_0 in subsection *B*.

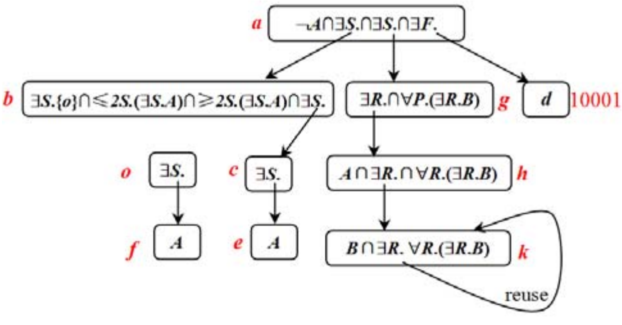

Figure 3. the forested FTC group of C_0 and names for its nodes.

According to Fig. 3, we can easily find a model of C_0' (the principal *FTC*). Of course, it is also a model of C_0:

Δ^I={ *a, b, c, e, f, g, h, k, o*}
R^I={ $<g, h>$, $<h, k>$, $<g, k>$, $<k, k>$};
S^I={ $<a, b>$,$<a, g>$,$<b, c>$,$<c, e>$,$<b, o>$,$<o, f>$};
F^I={ $<a, 10001>$}
A^I={ *f, e, h* } ; B^I={ *k* }
$C_0^I= C_0'^I$={ *a* }

Now let's see why *I* is a model of *C'*. Actually, we will prove later that *I* is also a model of the input concept.

Lemma 1: *If literal* $X \in cla(x)$, *then* $x \in X^I$ ($cla(x)$ denoting the clause designated by *x* here).

(1) If *X* is primitive concept *A*, then clearly it holds $x \in X^I$ according to the building of *I*. And then by induction, we have:

(2) If *X* is the negation of primitive concept, $\neg A$, then because there is no clash □ in $cla(x)$, so $A \notin cla(x)$. Therefore, $x \in \Delta^I \setminus A^I = X^I$.

(3) If *X* is $\top$, and $cla(x)$ contains only *X*, then it holds $x \in \Delta^I = X^I$ according to the semantics of $\top$.

(4) If *X* is $\leq nS.C$ or $\geq nS.C$, then there should be $\#\{<x,y>|<x,y>\in S^I \wedge y \in C^I\} \leq n$ or $\geq n$ according to the building processes of *FTC* group. Therefore, $x \in X^I$.

(5) If *X* is $\forall S.C$, meanwhile $cla(x)$ contains $\exists R$ role literals with $R \sqsubseteq_* S$, then because the *C* has been merged to each $\exists R$ literal, the individuals corresponding to restricting concepts of these $\exists R$ literals are necessarily the instances of *C*. Hence $x \in X^I$. However, if $cla(x)$ contains no $\exists R$ role literal with $R \sqsubseteq_* S$, then there is no individual having *S* relations with *x*. therefore, it is obvious that $x \in (\forall S.C)^I = X^I$. This is also quite similar with ($\forall F.d$).

(6) If *X* is $\exists R.C$, there is undoubtedly one individual $y \in (C)^I$ having $<x, y> \in (R)^I$. Hence, it holds $x \in X^I$. Similar inference also applies to ($\exists F.d$).

Proposition 1: *I is a model of C'*.

Proof: Actually, The description of concept *C'* is just the top clause which corresponds to an individual, say, x_0. Then x_0 is the instance of all the literals in the top clause according to lemma 1. Clearly, x_0 is an instance of this clause, namely, *C'*. Therefore, *I* is a model of *C'*.

D. Soundness, Completeness, and termination of FTC algorithm

In this subsection, we firstly present four lemmas to demonstrate the semantic features of merging operations ($\forall$-Merge,$\exists$-Merge, and *o*-Merge) and number restriction satisfying, and then prove the soundness, completeness, and termination of FTC algorithm based on them.

Lemma 2: *Providing a concept C turns into a new concept C' after being applied* $\forall$*-Merge once, then C and C' share same models, namely, if I is a model of C', then I is also a model of C and vice versa.*

Proof: Because each $\forall$-Merge operation only inferences two role literals, Let $C=\exists S.d_1 \sqcap \forall S.d_2 \sqcap C_3$ (*S* being concrete role, d_1 and d_2 being datatypes). Then, $C'=\exists S.(d_1 \sqcap d_2) \sqcap \forall S.d_2 \sqcap C_3$ after being applied with $\forall$-Merge.

Supposing that *I* is a model of *C*, there is certainly an individual $a \in \Delta^I$ with $a \in C^I$. Because of the existence of $\exists S.d_1$, there is surely a value $t \in \Delta^{\mathbf{D}}$ with $<a,t> \in S^I$ and $t \in d_1^{\mathbf{D}}$. And because of the existence of $\forall S.d_2$, *t* should be also a value within datatype d_2, namely, $t \in d_2^D$. Therefore, we have $a \in (\exists S.\ (d_1 \sqcap d_2) \sqcap \forall S.d_2 \sqcap C_3)^I = (C')^I$, In other words, *I* is also a model of *C'*.

Conversely, let *I* be a model of *C'*. Due to $(d_1 \sqcap d_2) \subseteq d_1$, It holds $\exists S.(d_1 \sqcap d_2) \sqsubseteq \exists S.(d_1)$ according to their semantics. Further, we have: $\exists S.(d_1 \sqcap d_2) \sqcap \forall S.d_2 \sqcap C_3 \subseteq \exists S.d_1 \sqcap \forall S.d_2 \sqcap C_3$, namely, $C' \sqsubseteq C$. Hence, *I* is also a model of *C*.

The way of proving also applies to ∀-Merge on abstract role.

Now, we know that a concept description won't change its models (if it has) after receiving a ∀-Merge operation. We call such operation *model-keeping operation*.

Lemma 3: *Providing a concept G turns into a new concept G' after being applied ≥ satisfying operation, then G is satisfiable if and only if G' is satisfiable.*

Proof: (1)"**if**"direction: If G' is satisfiable, then G is satisfiable.

Supposing that the current clause x contains literal $\geq nS.C$, G turns into G' after undergoing ≥ satisfying operation, with x turning into clause x' accordingly. It holds $x' \sqsubseteq x$ because x' contains all literals of x. Let I be a model of G'. Then there is surely an individual $a \in \Delta^I$ with $a \in (x')^I$ and $a \in (x)^I$ as a result. Furthermore, except x, the rests of G and G' are completely the same. Therefore, I is also a model of G.

(2)"**only if**"direction: If G is satisfiable, then G' is satisfiable.

Supposing that the current processing clause x contains literal $\geq nS.C$, x turns into $x'= x \sqcap \exists S.C_1 \sqcap \exists S.C_2 \sqcap \ldots \sqcap \exists S.C_n (C_i = C, 1 \leq i \leq n)$ after undergoing ≥ satisfying operation, with G turning into G' accordingly. $\boldsymbol{I}=(\Delta^I, \cdot^I)$ is a model of G. Therefore, there should be an individual $a \in \Delta^I$ with $a \in (x)^I$ and at least $c_1, c_2, \ldots c_n \in \Delta^I$ with $(a, c_i) \in S^I$ and $c_i \in C^I$. Prorate these n individuals (namely, c_i for $1 \leq i \leq n$) to the restricting concepts of newly created n role literals ($\exists S.C_i$, $1 \leq i \leq n$), namely, $c_i \in (C_i)^I$. This shows $a \in (x')^I$. therefore, ≥ satisfying is also model-keeping operation.

Lemma 4: *o-Merge is semantically equivalent operation.*

Proof: The *o*-Merge is just to move a description from its original place to extinguished point. Supposing the current clause $x=\{o\} \sqcap C_1$, while extinguished point $Des(o)=C_2$ currently. Because o is named individual, it holds $o \in (x)^I$, thus holds $o \in (C_1)^I$ in semantics. Furthermore, it holds $o \in (C_2)^I$ due to $Des(o)=C_2$. Therefore, individual o is actually an instance of $(C_1 \sqcap C_2)$, namely, $o \in (C_1 \sqcap C_2)^I$. And this is right the result of *o*-Merge operation: $x'=\{o\}$ and $Des(o)=C_1 \sqcap C_2$. This process is totally reversible, in other words, a concept won't change anything in semantics after receiving *o*-Merge operation. This is called *semantics equivalent operation*.

Lemma 5: *Providing a concept G can be possibly turned into new concepts* $G'_1, G'_2, \ldots,$ *or* G'_n *after being applied ≤ satisfying operation, then G is satisfiable if and only if at least one* G'_i $(1 \leq i \leq n)$ *is satisfiable.*

Proof: (1)"**if**"direction: If one certain G'_i $(1 \leq i \leq n)$ is satisfiable, then G is satisfiable.

Note that formula $\exists R.(C_i \sqcap C_j) \sqsubseteq \exists S.C_i \sqcap \exists R.C_j$ holds for any simple roles S, R with $R \underline{\sqsubseteq} S$ and any concepts C_i, C_j. And it follows that ∃-Merge is in deed an extension shrinking operation. Therefore, according to the transitivity of $\sqsubseteq$ and axiom: if $C_i \sqsubseteq C_j$ then $\exists S.C_i \sqsubseteq \exists S.C_j$, it holds $G'_i \sqsubseteq G$ for $1 \leq i \leq n$. Now, we conclude that if one certain G'_i $(1 \leq i \leq n)$ is satisfiable, then G is satisfiable.

(2)"**only if**"direction: If G is satisfiable, then at least one certain G'_i $(1 \leq i \leq n)$ is satisfiable.

Let current clause in G be $x=C_0 \sqcap \exists S.C_1 \sqcap \exists S.C_2 \sqcap \ldots \sqcap \exists S.C_m \sqcap \leq nS.C$ with $n<m$ and $C_i \underline{\sqsubseteq} C$, $\boldsymbol{I}=(\Delta^I, \cdot^I)$ be a model of G. Then there are at least one individual $a \in \Delta^I$ with $a \in x^I$ and individuals $c_1, c_2, \ldots c_m \in \Delta^I$ with $(a, c_i) \in S^I$ and $c_i \in C_i^I$. To satisfy atmost number restriction, there should be some $c_i (1 \leq i \leq m)$ identical. Based on this fact, we conduct ≤ satisfying operation on G as follows:

If c_j and c_i are identical, then apply ∃-Merge to $\exists S.C_j$ and $\exists S.C_i$. Repeat this until atmost number restriction holds.

After taking above process, we can develop a new clause x'. At the same time, G turns into G'. I is obviously also a model of G'. To such operation which is guided by certain relevant information, we called it *guided operation*.

Proposition 2 (**Soundness and Completeness**): *A concept G is satisfiable if and only if at least one of principal FTCs generated from it is satisfiable. And the models of these satisfiable principal FTCs are also models of G.*

Proof: Suppose that G can be possibly turned into principal *FTC*s $G'_1, G'_2, \ldots, G'_n$ after undergoing FTC algorithm.

(**1**) "**if**"direction, namely, soundness: From lemma 2, 3, and 5, we know ∀-Merge and ≥ satisfying are model-keeping operations, while ∃-Merge is an extension shrinking operation. The *o*-Merge and the building of DNF are both semantically equivalent transformations according to lemma 4 and the semantics features of ⊓ and ⊔. Clearly, the choosing clause from a formed DNF is an extension shrinking operation. Therefore, it holds $G'_i \sqsubseteq G$ for $1 \leq i \leq n$. If a principal *FTC* is satisfiable, say, G'_i $(1 \leq i \leq n)$. Let $\boldsymbol{I}=(\Delta^I, \cdot^I)$ be a model of G'_i. Then I is surely a model of G. Of course, there may be some primitive concepts or/and roles in G while not in G'_i. We just need to map them to ∅, which doesn't intervene the fact that I is a model of G.

(**2**) "**only if**"direction, namely, completeness: Let $\boldsymbol{I}=(\Delta^I, \cdot^I)$ be a model of G. Starting from G, we can generate a principal *FTC*, say, G'**,** such that I is a model of G'. To ensure I is a model of G', we need to do slight modification on FTC algorithm:

(1) For each processing node C, according to semantics, there should be an individual $a \in \Delta^I$ with $a \in C^I$. If the DNF of C has several clauses, say, $C'_1, C'_2, \ldots, C'_m$, then there should be an i $(1 \leq i \leq m)$ such that $a \in C_i^I$. Choose C_i and delete the rest clauses.

(2) When doing ≤ satisfying operations, take the guided mode provided in lemma 5.

This way, we can guarantee for any new descriptions generated by each step of operations, I is a model of them. Therefore, for the final G', I is obviously a model of it.

Lemma 6: *Let m be number of subconcepts of concept G, n be integer with* $n> m*2^m$, *R be role in G with*

Trans(R). And let clause sequence $x_1, x_2, \ldots ,x_n$ be a path in one FTC of FTC group of G. Then there is a clause x_i reusable by x_j for j>i.

Proof: The subconcepts of *G* have at most *m* possibilities, such that,

$$|\{ x_i.Letter| \ 2\le i\le n\}| < m, \text{ and}$$

$$|\{ \textbf{\textit{Upper}}(x_i)/R \mid 2\le i\le n \}| < 2^m.$$

Therefore, in this path, there should be two clauses x_i, x_j ($j>i$) satisfying:

$\textbf{\textit{Rol}}(x_i.Letter)=\textbf{\textit{Rol}}(x_j.Letter)=R$, Trans($R$),$\textbf{\textit{E}}(x_i.Letter)=\textbf{\textit{E}}(x_j.Letter)$, and $\textbf{\textit{Upper}}(x_j)/R \sqsubseteq \textbf{\textit{Upper}}(x_i)/R$.

This implies that x_i is the clause reusable by x_j.

Proposition 3(Termination): *FTC algorithm will terminate after having processed a certain amount of processing nodes.*

Proof: Let *m* be number of subconcepts of concept *G*. Distinctly, *m* is linear in length of *G*. The termination of FTC algorithm is decided by the following properties.

(1) The process of FTC doesn't remove processing nodes, and doesn't remove literals from clause except o-Merge operation. It seems that ∃-Merge deletes restricting node or ∃ role literal. However, to be more exact, such operations move descriptions from one node to another.

(2) New processing nodes can only be created by literals like $\exists R.C$ and $\ge nS.C$. And $\exists R.C$ can create at most one processing node, while $\ge nS.C$ may produce *n* or no processing nodes. Moreover, such literals like $\exists R.C$ or $\ge nS.C$ can only be at most *m*. Hence, the outbound degree of a clause can be *nm* at most.

(3) According to lemma 6, the length of a path in *FTC*s of *G* will be m*2^m maximum.

E. Complexity issues

The FTC-based satisfiability algorithm for SHOQ(D) presented above may need exponential time and space. Undoubtedly, the length of clauses in the instance nodes of forested *FTC* group is linear in the length of input concept description. Then, the temporal and spatial costs of FTC algorithm both focus on the number of instance nodes of forested *FTC* group, which is just the number of ∃ role literals at all levels in the final description (namely, the *FTC* group). This number is linear in the length of input concept description in many cases. However, it can reach exponential level due to the interaction between ∀ role literals and ∃ role literals. Take the following classic example [14] for instance:

$$C_1=\exists R.A\sqcap\exists R.B;$$
$$\ldots$$
$$C_n=\exists R.A\sqcap\exists R.B\sqcap\forall R.C_{n-1};$$

Obviously, the size of C_n grows linearly in n. However, the number of the instance nodes of forested *FTC* of C_n can be exponential in n.

Moreover, from lemma 6, we know that the length of the path of *FTC* may reach m*2^m (*m* is the number of subconcepts of input concept) due to the non-shrinking passing down of transitive ∀ role literals. Because different paths can be processed independently, we can keep only one path in the memory when processing. And the processing time and space on one instance node are all linear in the size of input concept. Therefore, both the temporal and spatial complexities are $O(m^2\cdot 2^m)$.

IV. The Comparison between FTC and Tableaux

In description logics, most (even slightly) complex languages (say, with negation) take Tableaux algorithms to decide the concept satisfiability. Therefore, we make a comparison between these two algorithms in temporal and spatial performances from which we can conclude that FTC algorithm is much better than Tableau spatially.

The instance nodes of the forested *FTC* group actually correspond to the nodes in Tableau completion forest because there are all linked by roles and interpreted as individuals later. And the processing in one node in Tableau and FTC needs only linear time. Therefore, both have the same temporal complexity.

For each node, Tableau algorithm unfolds the descriptions gradually with ⊓/⊔-rules until no descriptions can be applied to. At this moment, the group of descriptions which cannot be broken down further forms the right literals of the clause that can be obtained in FTC algorithm. The difference lies in the fact that Tableau reserves all the initial and intermediate descriptions in the decomposition, while the FTC just keeps the final undecomposable descriptions which really avail later. For example, for a concept $x_0=\{A\sqcap B\sqcap(C\sqcup B)\}$, Tableau will turns it into $x_0=\{A\sqcap B\sqcap(C\sqcup D), A, B\sqcap(C\sqcup D), \mathrm{B}, (C\sqcup D), C/D \}$, while FTC generates $x_0=\{A\sqcap B\sqcap C/D\}$. Actually, the initial and intermediate descriptions are just used to generate the final undecomposable descriptions (we call them literals in FTC) and become unnecessary once done with their duties. The problem is that Tableau still keeps all these discardable descriptions due to its inner mechanism thus causing great spatial losses. This is the first advantage of FTC algorithm.

Besides, on the process of ∃ operator in Tableaux, each description prefixed with ∃ (namely, the ∃ role literal in FTC) create a new node. For example, providing there is $\exists R.C$ in node *x*, then ∃-rule will be triggered to create a new node *y* labeled with *C*, together with a new edge $R(x,y)$. In such mode, description *C* will appears twice (both in nodes *x* and *y*). FTC algorithm has no such expense. This is the second advantage of FTC algorithm. Nevertheless, to stop the unnecessary extension, FTC still keeps the original restricting concepts for transitive roles, which amounts to having the same costs as Tableaux in this regard. However, the fraction of transitive roles used in concept is usually small; therefore the cost saved here is still considerable in many cases in FTC algorithm.

In general, for each node, it is labeled with a set of subconcepts in Tableau, and a clause (with the ∃ role literals' restricting concepts scattered in other nodes) in FTC algorithm respectively. Furthermore, the length of each subconcept is certainly linear in the size and also the subconcept number of input concept. Therefore, providing *m* is the number of subconcepts of input concept, the spatial cost of one node in Tableau is $O(m^2)$, while that of FTC is just $O(m)$. Besides, the numbers of nodes in Tableaux and FTC algorithm are almost the same which could be linear (or exponential) in the length of input concept. As a result, FTC algorithm can save linear (or exponential) space compared with Tableau. It is the significance of FTC.

Still taking the C_0 in subsection *B* of section III for example, let's take a look at the space cost of Tableau on C_0 (see Fig. 4).

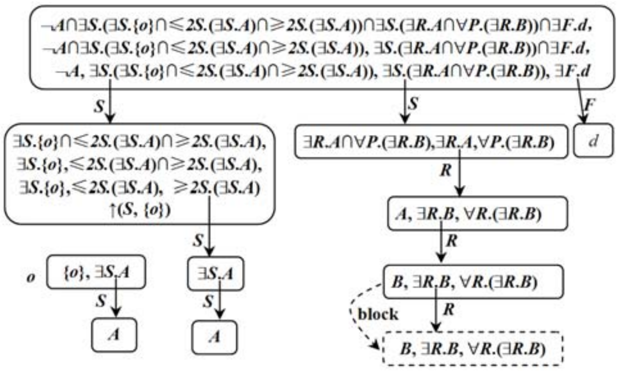

Figure 4. The Tableau's completion forest built according to concept C_0.

By comparing Fig. 3 with Fig. 4 and the statistics of length of descriptions labeled in the nodes in both figures, we can discover that the size of the description of C_0 is 61 characters, while that of Tableau forest is 360 characters and FTC 111 characters (108 for descriptions in Fig. 3, and 3 for labeling), being 5.9 and 1.8 times the length of C_0 respectively. From this statistics, we clearly know that the overlaps caused by ∩, ∪, ∃ operators in Tableaux are very serious. Furthermore, the fraction of ∩, ∪, ∃ operators in most concepts is very large. Therefore, such huge difference in space cost shown here has considerable universality. Because of this, the advantage of FTC in space looks much obvious and the popularization of it seems much urgent.

V. Conclusion and Discussion

Tableau algorithm adopts the consistence of *ABox* to decide the satisfiability of *SHOQ*(**D**)-concepts, while FTC algorithm reorganizes the input concept description and obtains the judgment of satisfibility, thus "working out" the concept satisfiability in a very real sense. In implementation, Tableau features extending around individuals, while FTC focuses on clauses. In this sense, they are interlinked, for a clause corresponds to an individual when forming interpretation.

In performance, FTC is a direct decision process on concept satisfiability, discarding those unnecessary operations. Especially on the process of ∩, ∪, ∃ operators, FTC is obviously much better than Tableau. Therefore, compared with Tableau, FTC can save linear (or exponential) space depending on the number of nodes. Besides, FTC algorithm has still room for improvement. For simple roles, the ∀ role literals can be eliminated after applying ∀-Merge to ∃ each role literals with the same roles, which can also save considerable space in many cases. However, Tableau algorithm is not only used to judge to concept satisfiability, while FTC just aims at this single function currently. Whether FTC can be used in other reasoning issues is still under research. We firmly believe, with the deepening of the research, FTC should be a new basic support for reasoning in description logics and semantic WEB.

Acknowledgment

This work is supported in part by the Natural Science Foundation of ZheJiang Province of China under Grant No. Y1090734.

References

[1] D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, "OIL: An ontology infrastructure for the semantic web," IEEE Intelligent Systems, vol. 16, no. 2, pp. 38-45, 2001.

[2] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, "DAML+OIL (March 2001) reference description," W3C Note, December 2001, Available at http://www.w3.org/TR/2001/NOTE-daml+oil-reference-20011218.

[3] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P.F. Patel-Schneider, L. A. Stein, "OWL web ontology language reference. W3C Working Draft," March 2003, Available at http://www.w3.org/TR/2003/WD-owl-ref-20030331.

[4] U. Hustadt, B. Motik, U. Sattler, "Reasoning in Description Logics by a Reduction to Disjunctive Datalog," Journal of Automated Reasoning, vol. 39, pp. 351-384, 2007.

[5] H.W. Christian, P. Bijan and S. Evren, "Description logic reasoning with syntactic updates," In: The 5th international conference on ontologies, databases, and applications of semantics, Springer Berlin/Heidelberg, Vol. 4275, pp. 722-737, 2006.

[6] I. Horrocks, U. Sattler, "A Tableau Decision Procedure for SHOIQ," Journal of Automated Reasoning, vol. 39, no. 3, pp. 249-276, 2007.

[7] T. Liebig, F. Müller, "Parallelizing Tableaux-Based Description Logic Reasoning," In On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, Vol. 4806, pp. 1135-1144. 2007.

[8] C. Lutz and M. Miličić, "A Tableau Algorithm for Description Logics with Concrete Domains and General Tboxes," Journal of Automated Reasoning, Springer Netherlands, vol. 38, no. 1-3, pp. 227-259, 2007.

[9] J. Bao, D. Caragea, and V. Honavar, "A Distributed Tableau Algorithm for Package-based Description Logics," In Proceedings of the Second International Workshop on Context Representation and Reasoning (CRR 2006), Riva del Garda, Italy, CEUR, 2006.

[10] L. Chang, F. Lin, and Z.Z. Shi, "A Dynamic Description Logic for Representation and Reasoning About Actions, " In: Zhang Z. and Siekmann J. (Eds.), KSEM 2007. LNAI 4798, pp. 115-127, 2007.

[11] L. Chang, Z.Z. Shi, L.R. Qiu, F. Lin, "A Tableau Decision Algorithm for Dynamic Description Logic," Chinese

Journal of Computers, vol.31, no.6, pp. 896-909, 2008. (In Chinese)

[12] Y.C. Jiang, Z.Z. Shi, Y. Tang, J. Wang, "Fuzzy Description Logic for Semantics Representation of the Semantic Web," Journal of Software, vol. 18, no. 6, pp. 1257−1269, 2007. (In Chinese)

[13] G. Stoilos, G. Stamou, J. Z. Pan, V. Tzouvaras, I. Horrocks, "Reasoning with Very Expressive Fuzzy Description Logics," Journal of Artificial Intelligence Research, vo. 30, pp. 273-320, 2007.

[14] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, P. F. Patel-Schneider, "The Description Logic Handbook: Theory, Implementation, and Applications," Cambridge: Cambridge University, 2003.

[15] I. Horrocks, and U. Sattler, "Ontology reasoning in the SHOQ(D) description logic," In Proc. of IJCAI 2001, pp.199-204, 2001.

[16] I. Horrocks, P. F. Pate1-Schneider, F. van Harmelen, "From SHIQ and RDF to OWL: The Making of a Web Ontology Language," Journa1 of Web Semantics, vol. 1, no.1, pp.7-26, 2003.

[17] I. Horrocks, "DAML+OIL: A description logic for the semantic Web," Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol.25, no.1, pp. 4-9, 2002.

[18] F. Baader and P. Hanschke, "A scheme for integrating concrete domains into concept languages," In: Proc. of IJCAI-91, pp. 452-457, 1991.

[19] A. Schaerf, "Reasoning with individuals in concept languages," Data and Knowledge Engineering, vol. 13, no. 2, pp. 141-176, 1994.

[20] I. Horrocks, U. Sattler, and S. Tobies, "Practical reasoning for expressive description logics," In: Ganzinger H., McAllester D., and Voronkov A.(eds.), Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99), number 1705 in Lecture Notes in Artificial Intelligence, Springer-Verlag, pp.161-180, 1999.