An Optimized Tableau Structure for Explicit Representation of Disjunction

François de Bertrand de Beuvron, Martina Kullmann
and François Rousselot
Laboratoire d'Informatique et d'Intelligence Artificielle,
ENSAIS, FRANCE
beuvron@liia.u-strasbg.fr

Abstract

Recent DL reasoners (Ciclop [1], DLP [8],iFact [5], RACER [2]) have proven their practical efficiency for concept expression satisfiability checking (TBox Reasonning) within expressive DLs. Typically, such reasoners explore one branch of disjunction (alternatives introduced by OR construct) at a time, backtracking when necessary. We present in this paper a new tableau structure that efficiently keep track of proof states for more than one alternative. We claim that such structure called *Alt*-Tableau is well suited for reasoning within ABoxes and so-called Concrete Domains [7]. We also present a decision procedure for the satisfiability of $\mathcal{ALC}$-concept within *Alt*-Tableau .

1 Informal Presentation and Motivations

The idea presented in this paper is orthogonal to expressivity, and can be applied to any logic that allow for general disjunction. However, if allowing cyclic TBoxes and/or high expressivity as SHIQ [5] we cannot rely on the finite tree model property, and tableau algorithm for these logics have to include some kind of so-called blocking mechanism. Intuitively, blocking technics try to find if an already existing individual can be used instead of creating a new individual while processing *individual-introducing* construction such as $\exists R.C$. We think that we can take advantage of the structure, designed for disjunction representation, presented in this paper to deal also with this problem by designing a rule like *choose an existing individual from a set of candidate blockers, and try to associate it with the individual to be created. If none such association leads to satisfiability, create a new individual.* The choice within candidates can be

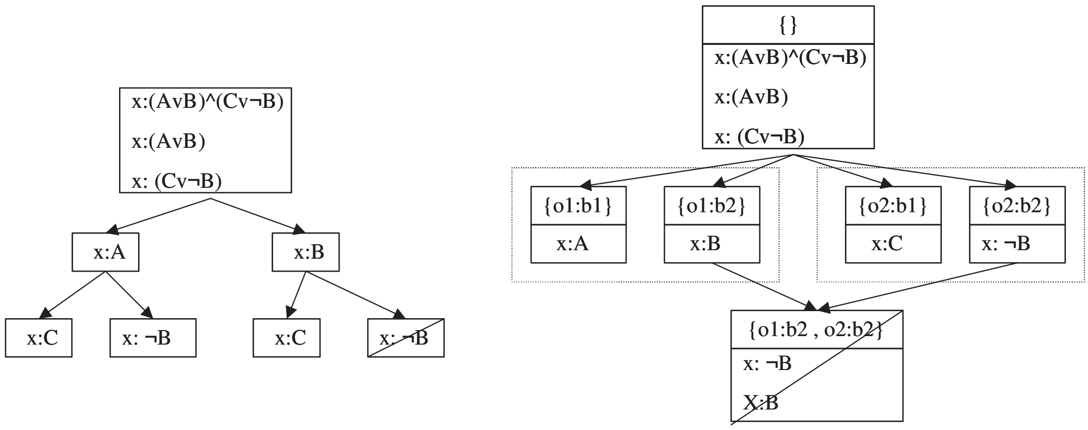

Figure 1: OR-tree and *Alt*-DAG for $\mathcal{ALC}$-concept $(A \sqcup B) \sqcap (C \sqcup \neg B)$

seen as a specific disjunction. Unfortunately, the formalization of this idea is still under study, and we will therefore restrict ourself to $\mathcal{ALC}$ with acyclic TBox (where satisfiablility can be reduced to $\mathcal{ALC}$-concept satisfiability by unfolding) in this paper.

Now consider the simple $\mathcal{ALC}$-concept $(A \sqcup B) \sqcap (C \sqcup \neg B)$. Three distinct tableaux can be obtained depending of the branches of ORs we choose. The fourth possibility lead to a clash. These tableaux can be represented by two OR-Trees (the first beeing given in figure 1 depending of the OR we choose to expand first. Note that this OR-Tree structure is effectively used in many reasoners although implicitely : only one branch of the OR-Tree is expanded at a given time, some backtracking being applied if the current branch lead to a clash. Nevertheless, it is easy to see that this is not optimal : in the first tree, two nodes correspond to the same expression C. Suppose that C is now some complex sub-expression, it will be expanded twice. Note that the $x : C$ constraint cannot be straightforwardly expanded once and used for both branches since this expansion may interact with the constraints $x : A$ and/or $x : B$. To avoid such redundancy in the structure proposed here, we create a single node (called BranchNode) for each branch of all disjunctions at a given level. If and only if there is some kind of interaction between branches introduced by distinct ORs, we create a new node (called MergeNode) representing the situation where both branches have been chosen. The resulting structure presented in figure 1, is no more a tree but an Acyclic Directed Graph we call *Alt*-DAG . This representation has two main advantages:

- each constraint is represented only once and will be expanded only once eventually producing new MergeNodes if it interact with other constraints in distinct alternatives.

- more than one alternative can be efficiently stored in the structure. Furthermore, it can be efficiently used in subsequent proofs allowing enhanced caching strategies. Also, this structure is well suited for designing parallel solvers.

We claim therefore that *Alt*-DAG representation is particularly well suited for the following tasks (see [6] for a detailed presentation):

- using heuristics to guide the proof : Some expansion rules are computationally more expansive than others. It may be effective to define heuristics to guide the proof. This is possible only if more than one alternative can be represented within the tableau, effectively allowing to switch from an alternative to another while keeping track of the proof state in each alternative. Although, please note that such improvement can only be obtained if more than one alternative is satisfiable. In particular, if the expression we are checking is inconsistent, all the alternatives have to be checked, and the computational overhead of the structure presented here may be an handicap in such situation. However, the advantages of the single expansion of constraints, and of potential for parallelization, still hold.
- ABox consistency checking :
  - many disjunction in distinct individuals are expected to be unrelated : suppose a library database with persons and books. As usual, persons can be men or women, and books may be real books or articles. Wether a person is man or woman do not interfere with it's hiring of books or articles. In this case, allowing for some kind of *local* disjunction representation, while ensuring that interactions will be correctly taken into account when needed should prove very effective.
  - without specific caching strategy, checking ABox consistency will become more and more computationally intensive as the number of individuals increase. DL systems incorporating ABoxes generally provide some caching strategy to deal with this problem. However, without explicit representation of disjunction, these caches are often limited to some deterministic subset of individual's descriptions.
  - inconsistency is uncommon and often correspond to user's misconception.
- reasoning with concrete domains : the use of heuristics and the single expansion of constraints may become more important if one want to deal with so-called concrete domains : various concrete domains (numeric constraints, time constraint, string constraint...) will have various complexity,

both theoretically and practically. Choosing a proof that minimize the calls to the most demanding concrete reasoners may greatly reduce the overall proof time.

2 $\mathcal{ALC}$ tableau algorithm

$\mathcal{ALC}$-concept are defined as usual from a set $\mathcal{CN}$ of concept names and a set $\mathcal{RN}$ of role names. We denote by $sub(C)$ the set of subconcept in C and by $\|C\|$ the size of an $\mathcal{ALC}$-concept C (see fig. 2).

syntax	semantics	subconcepts	size
$A \in \mathcal{CN}$	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\{\mathrm{A}\}$	1
$\top$	$\Delta^{\mathcal{I}}$	$\{\top\}$	1
$\neg D$	$\Delta^{\mathcal{I}} - D^{\mathcal{I}}$	$\{\neg D\} \cup sub(D)$	$\|D\| + 1$
$D \sqcap E$	$D^{\mathcal{I}} \cap E^{\mathcal{I}}$	$\{D \sqcap E\} \cup sub(D) \cup sub(E)$	$\|D\| + \|E\|$
$D \sqcup E$	$D^{\mathcal{I}} \cup E^{\mathcal{I}}$	$\{D \sqcup E\} \cup sub(D) \cup sub(E)$	$\|D\| + \|E\|$
$\exists R.D$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \cap D^{\mathcal{I}} \neq \emptyset\}$	$\{\exists R.D\} \cup sub(D)$	$\|D\| + 1$
$\forall R.D$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \subseteq D^{\mathcal{I}}\}$	$\{\forall R.D\} \cup sub(D)$	$\|D\| + 1$

Figure 2: $\mathcal{ALC}$-concept

In the rest of this presentation, we will assume that $\mathcal{ALC}$-concept are finite and are in Negation Normal Form (NNF) in which negation apply only to concept names. The transformation of a $\mathcal{ALC}$-concept to a equivalent $\mathcal{ALC}$-concept in NNF is quite straightforward, and will be skipped in this paper.

2.1 *Alt*-Tableau

As foreseen in fig. 1, wee need to identify each expansion of disjunction. We will call OrIds these idendificators, and denote by $\mathcal{O}$ the set of OrIds. An alternative is a partial function from $\mathcal{O}$ to $\{0, 1\}$ identifying the branches of the disjunctions that have been expanded. We denote by $\mathcal{A} \subset 2^{\mathcal{O}\times\{0,1\}}$ the set of alternatives. An alternative a is said to be complete *iff* $domain(a) = \mathcal{O}$. Two alternatives a_1 and a_2 are compatibles noted $a_1 \odot a_2$ *iff* $\forall o \in domain(a_1) \cap domain(a_2), a_1(o) = a_2(o)$. Note that if two alternatives a_1 and a_2 are compatibles, then the relation $a_1 \cup a_2$ is also a partial function from $\mathcal{O}$ to $\{0,1\}$ and so an alternative.

A pre-*Alt*-Tableau for an $\mathcal{ALC}$-concept C_0 is a quintuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}]$ where S is the set of individuals, $\mathcal{O}$ is the set of OrIds, $\mathcal{C} \subseteq \mathcal{A}$ is the set of clashed alternatives, $\mathcal{LP} \subseteq \mathcal{A} \times S \times sub(C_0)$ keep track of the constraints associated with individuals, and $\mathcal{E} \subseteq \mathcal{A} \times S \times S \times \mathcal{RN}$ keep track of the relations between individuals. An *Alt*-Tableau is a sextuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}, a_*]$, where $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}]$ is a pre-*Alt*-Tableau and $a_* \in \mathcal{A}$ is a complete alternative. A pre-*Alt*-Tableau must verify properties A1-A7 of figure 3. An *Alt*-Tableau must also verify property A8.

Id	name	property definition
A1	C_0 in model	$\exists[\emptyset, s, C_0] \in \mathcal{LP}$
A¬	¬ coherence	if $[a_1, s, D] \in \mathcal{L}$ and $[a_2, s, \neg D] \in \mathcal{L}$ and $a_1 \odot a_2$ then $a_1 \cup a_2 \in \mathcal{C}$
A⊓	⊓ coherence	if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$
A⊔	⊔ coherence	if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\forall a_2 \in \mathcal{A}$ with a_2 complete and $a_1 \subseteq a_2$ $\exists a_3 \subseteq a_2$ such that $[a_3, s, D] \in \mathcal{L}$ or $[a_3, s, E] \in \mathcal{L}$
A∀	∀ coherence	if $[a_1, s_1, \forall R.D] \in \mathcal{L}$ and $[a_2, s_1, s_2, R] \in \mathcal{D}$ and $a_1 \odot a_2$ and $a_1 \notin \mathcal{C}$ and $a_2 \notin \mathcal{C}$ then $\exists a_3 \in \mathcal{A}$ with $a_3 \subseteq a_1 \cup a_2$ such that $[a_3, s_2, D] \in \mathcal{L}$
A∃	∃ coherence	if $[a_1, s_1, \exists R.D] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s_2, D] \in \mathcal{L}$ and $[a_3, s_1, s_2, R] \in \mathcal{E}$
A7	clashes	$\forall a_1, a_2 \in \mathcal{A}, a_1 \in \mathcal{C} \wedge a_1 \subseteq a_2 \Rightarrow a_2 \in \mathcal{C}$
A8	satisfiable	$a_* \notin \mathcal{C}$

Figure 3: *Alt*-Tableau properties

Id	name	property definition
B1	C_0 in model	$\exists s \in S$ such that $C \in \mathcal{L}_b(s)$
B¬	¬ coherence	if $D \in \mathcal{L}_b(s)$ then $\neg D \notin \mathcal{L}_b(s)$
B⊓	⊓ coherence	if $D \sqcap E \in \mathcal{L}_b(s)$ then $D \in \mathcal{L}_b(s)$ and $E \in \mathcal{L}_b(s)$
B⊔	⊓ coherence	if $D \sqcup E \in \mathcal{L}_b(s)$ then $D \in \mathcal{L}_b(s)$ or $E \in \mathcal{L}_b(s)$
B∀	∀ coherence	if $\forall R.D \in \mathcal{L}_b(s_1)$ and $[s_1, s_2] \in \mathcal{E}_b$ then $D \in \mathcal{L}_b(s_2)$
B∃	∃ coherence	if $\exists R.D \in \mathcal{L}_b(s_1)$ then $\exists s_2 \in S$ such that $D \in \mathcal{L}_b(s_2)$ and $[s_1, s_2] \in \mathcal{E}_b$

Figure 4: *Basic*-Tableau properties

Lemma 2.1 *An $\mathcal{ALC}$-concept C_0 is satisfiable iff there exists an Alt-Tableau for C_0*

A tableau structure was introduced in [3] for ALC_{R^+}. Such a tableau called here *Basic*-Tableau to avoid confusion with *Alt*-Tableau is a triple $[S_b, \mathcal{L}_b, \mathcal{E}_b]$ where S_b is the set of individuals, $\mathcal{L}_b : S_b \rightarrow 2^{sub(C_0)}$ keep track of the constraints associated with individuals, and $\mathcal{E}_b : \mathcal{RN} \rightarrow 2^{S_b \times S_b}$ keep track of the relations between individuals. A *Basic*-Tableau must verify the properties given in figure 4. It is proven in [3] that an ALC_{R^+}-concept C_0 is satisfiable *iff* there is a *Basic*-Tableau for C_0. For proving lemma 2.1, we can prove that an *Alt*-Tableau exists for C_0 *iff* a *Basic*-Tableau exists for C_0.

Given an *Alt*-Tableau $\mathcal{T} = [S, \mathcal{O}, \mathcal{C}, \mathcal{L}, \mathcal{E}, a_*]$, we create a *Basic*-Tableau $\mathcal{T}_b = [S_b, \mathcal{L}_b, \mathcal{E}_b]$ with $S_b =$ S, $\mathcal{L}_b(s) = \{D \in sub(C_0)$ such that $\exists [a, s, C] \in \mathcal{L}$ for some $a \subseteq a_*\}$ and $\mathcal{E}_b(R) = \{[s_1, s_2] \in S \times S$ such that $\exists [a, s_1, s_2, R] \in \mathcal{E}$ for some $a \subseteq a_*\}$. Assuming that $\mathcal{T}$ verifies properties A1-A8 (fig 3) of *Alt*-Tableau , we prove below that $\mathcal{T}_b$ verifies properties B1-B$\exists$ (fig 4) of *Basic*-Tableau :

B1: obvious from A1

B$\neg$: $D \in \mathcal{L}_b(s) \Rightarrow \exists a_1 \subseteq a_*$ such that $[a_1, s, D] \in \mathcal{L}$
$\neg D \in \mathcal{L}_b(s) \Rightarrow \exists a_2 \subseteq a_*$ such that $[a_2, s, \neg D] \in \mathcal{L}$
since $a_1 \cup a_2 \subseteq a_*$ then (A$\neg$) $a_1 \cup a_2 \in \mathcal{C}$ so (A7) $a_* \in \mathcal{C}$
impossible by (A8). Hence $D \in \mathcal{L}_b(s) \Rightarrow \neg D \notin \mathcal{L}_b(s)$

B$\sqcap$: if $D \sqcap E \in \mathcal{L}_b(s)$ then ($\mathcal{L}_b$ def.) $\exists a_1 \subseteq a_*$ such that $[a_1, s, D \sqcap E] \in \mathcal{L}$ and (A7,A8) $a_1 \notin \mathcal{C}$
so (A$\sqcap$) $\exists a_2 \subseteq a_1 \subseteq a_*$ and $a_3 \subseteq a_1 \subseteq a_*$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$. Hence ($\mathcal{L}_b$ def.) $D \in \mathcal{L}_b(s)$ and $E \in \mathcal{L}_b(s)$

B$\sqcup$: if $D \sqcup E \in \mathcal{L}_b(s)$ then ($\mathcal{L}_b$ def.) $\exists a_1 \subseteq a_*$ such that $[a_1, s, D \sqcup E] \in \mathcal{L}$ and (A7,A8) $a_1 \notin \mathcal{C}$
so (A$\sqcup$ + a_* complete) $\exists a_3 \subseteq a_*$ such that $[a_3, s, D] \in \mathcal{L}$ or $[a_3, s, E] \in \mathcal{L}$. Hence ($\mathcal{L}_b$ def.) $D \in \mathcal{L}_b(s)$ or $E \in \mathcal{L}_b(s)$

B$\forall$,B$\exists$: same kind of proof as B$\sqcap$

For the converse, we construct an *Alt*-Tableau from a *Basic*-Tableau with $S = S_b$, $\mathcal{O} = \emptyset$, $\mathcal{C} = \emptyset$, $\mathcal{L} = \{[\emptyset, s, D] \in \mathcal{A} \times S \times sub(C_0);\ D \in \mathcal{L}_b(s)\}$, $\mathcal{E} = \{[\emptyset, s_1, s_2, R] \in \mathcal{A} \times S \times S \times \mathcal{RN};\ [s_1, s_2] \in \mathcal{E}_b(R)\}$, and $a_* = \emptyset$. This tableau verifies properties A1-A8 (fig 3) :

A1: obvious from (B1) and $\emptyset \subseteq a_*$

A$\neg$: obvious from (B$\neg$) : the condition is never met

A$\sqcap$,A$\forall$,A$\exists$: obvious from (B$\sqcap$,B$\forall$,B$\exists$) : simply rewrite with $a_i = \emptyset$

A$\sqcup$: if $[a_1, s, D \sqcup E] \in \mathcal{L}$ then $a_1 = \emptyset$ and $D \sqcup E \in \mathcal{L}_b(s)$ so ($D \in \mathcal{L}_b(s)$ and $[\emptyset, s, D] \in \mathcal{L}$) or ($E \in \mathcal{L}_b(s)$ and $[\emptyset, s, E] \in \mathcal{L}$). since $\forall a, \emptyset \subseteq a$ A$\sqcup$ is trivially verified.

A8,A7: trivial since $\mathcal{C} = \emptyset$

2.2 *Alt*-Tableau Algorithm

The tableau algorithm presented below work on ProofContext . A ProofContext is a sextuple $[S, \mathcal{O}, \mathcal{C}, \mathcal{LP}, \mathcal{E}, \mathcal{P}]$ where S, $\mathcal{O}$, $\mathcal{C}$, $\mathcal{LP}$, and $\mathcal{E}$ are the same as for a pre-*Alt*-Tableau but without properties A1-A7, and $\mathcal{P} \subseteq \mathcal{A} \times S \times sub(C_0)$ will contains the set of already expanded Or constraint (see rule R$\sqcup$). To check the satisfiability of an $\mathcal{ALC}$-concept C_0, we apply the expansion rules of figure 5 to the initial ProofContext defined by : $S = \{s_0\}$, $\mathcal{O} = \emptyset$, $\mathcal{C} = \emptyset$, $\mathcal{L} = \{[\emptyset, s_0, C_0]\}$ (meaning individual s_0 must satisfy C_0 for the root alternative, and so for all alternatives), $\mathcal{E} = \emptyset$, $\mathcal{P} = \emptyset$.

A ProofContext is complete when none of the rules is applicable. Then, the algorithm return "C is satisfiable" if $\emptyset \in \mathcal{C}$, and "C is unsatisfiable" otherwise.

Lemma 2.2 (termination) *for every $\mathcal{ALC}$-concept C_0, the Alt-Tableau algorithm terminate*

This is ensured by the following properties of the *Alt*-Tableau algorithm :

T1 each constraint in $\mathcal{L}$ can trigger at most one rule expansion for rules R$\sqcap$, R$\sqcup$, and R$\exists$. Each pair of constraints in $\mathcal{L} \times \mathcal{E}$ can trigger at most one rule expansion for rule R$\forall$. This is ensured by conditions (2) on each rule.

T2 each rule expansion add at least one new element in $\mathcal{L}$, $\mathcal{E}$, or $\mathcal{C}$.

T3 S is given a tree structure by $\mathcal{E}$, since for all $s \in \mathcal{S}, s \neq s_0$ there is an unique $s_p \in S$ such that $[a, s_p, s, R] \in \mathcal{E}$ (see R$\exists$)

We denote by S^n the subset of S of depth n within the tree induced by $\mathcal{E}$. S^n is recursively defined by : $S^0 = \{s_0\}, S^{i+1} = \{s \in S; \exists [a, s_p, s, R] \in \mathcal{E}$ with $s_p \in S^i\}$. By analogy, we also define $\mathcal{L}^i = \{[a, s, C] \in \mathcal{L}; s \in S^i\}$, $\mathcal{E}^i = \{[a, s_1, s_2, R] \in \mathcal{E}; s_1 \in S^i\}$, $\mathcal{A}^i = \{a \in \mathcal{A}; \exists [a, s, C] \in \mathcal{L}^i\}$, $\mathcal{O}^i = \{o \in \mathcal{O}; \exists a \in \mathcal{A}^i$ such that $[o, 0] \in a$ or $[o, 1] \in a\}$.

T4 expansion of constraints $[a_1, s_1, C_1]$ by rules R$\sqcap$, R$\sqcup$, R$\forall$, and R$\exists$ can only produce new constraints $[a_2, s_2, C_2]$ with $\|C_1\| \leq \|C_2\| - 1$.

T5 Constraints in $\mathcal{L}^i$ can only be introduced by constraints in $\mathcal{L}^i$ by rules R$\sqcap$ and R$\sqcup$, constraints in $\mathcal{L}^{i-1}$ by rule R$\exists$, and constraints in $\mathcal{L}^{i-1}$ and $\mathcal{E}^{i-1}$ by rule R$\forall$. Constraints in $\mathcal{E}^i$ can only be introduced by constraints in $\mathcal{L}^{i-1}$ by rule R$\exists$.

T6 from (T4,T5), it is easy to see that

$$\max_{[a_1, s_1, C] \in \mathcal{L}^{i+1}} \|C\| \leq \max_{[a_2, s_2, C] \in \mathcal{L}^i} \|C\| \text{ and } \max_{[a_1, s_1, C] \in \mathcal{L}^0} \|C\| \leq \|C_0\|$$

Since only $\mathcal{ALC}$-concept of size at least two ($\exists R.C$) can produce new individuals, the depth of the tree is at most $\|C_0\| - 1$.

Id	name	rule definition
R$\neg$	clash creation	1- if $[a_1, s, D] \in \mathcal{L}$ and $[a_2, s, \neg D] \in \mathcal{L}$ and $a_1 \odot a_2$ 2- if $a_1 \cup a_2 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_1 \cup a_2\}$
R$\sqcap$	$\sqcap$-Rule	1- if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ 2- if not $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ and $[a_3, s, E] \in \mathcal{L}$ then $\mathcal{L} \rightarrowtail \mathcal{L} \cup \{[a_1, s, D], [a_1, s, E]\}$
R$\sqcup$	$\sqcup$-Rule	1- if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ 2a- if not ($[a_1, s, D \sqcup E] \in \mathcal{P}$ or $\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ such that $[a_2, s, D] \in \mathcal{L}$ or $\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ such that $[a_3, s, E] \in \mathcal{L}$) then $\mathcal{P} \rightarrowtail \mathcal{P} \cup \{[a_1, s, D \sqcup E]\}$ create a new OrId o_n, $\mathcal{O} \rightarrowtail \mathcal{O} \cup \{o_n\}$ $\mathcal{L} \rightarrowtail \mathcal{L} \cup \{[a_1 \cup \{[o_n, 0]\}, s, D]\} \cup \{[a_1 \cup \{[o_n, 0]\}, s, E]\}$
R$\forall$	$\forall$-Rule	1- if $[a_1, s_1, u] \in \mathcal{LP}$ with $C/u = \forall R.D$ and $[a_2, s_1, s_2, R] \in \mathcal{E}$ and $a_1 \odot a_2$ 2- if not $\exists a_3 \in \mathcal{A}$ with $a_3 \subseteq a_1 \cup a_2$ such that $[a_3, s_2, D] \in \mathcal{L}$ then $\mathcal{LP} \rightarrowtail \mathcal{LP} \cup \{[a_1 \cup a_2, s_2, u \oplus [1]]\}$
R$\exists$	$\exists$-Rule	1- if $[a_1, s_1, u] \in \mathcal{LP}$ with $C/u = \exists R.D$ 2- if not $\exists a_2, a_3 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $a_3 \subseteq a_1$ such that $[a_2, s_1, s_2, R] \in \mathcal{E}$ and $[a_3, s_2, D] \in \mathcal{L}$ then create a new individual ns, $S \rightarrowtail S \cup \{ns\}$, $\mathcal{LP} \rightarrowtail \mathcal{LP} \cup \{[a_1, ns, u \oplus [1]]\}$, $\mathcal{E} \rightarrowtail \mathcal{E} \cup \{[a_1, s_1, ns, R]\}$
R6	clash propagation CP1	1- if $\exists a_1, a_2 \in \mathcal{C}$ such that $\exists a_3 \in \mathcal{A}, \exists o \in \mathcal{O}$ such that $a_1 = a_3 \cup \{[o, 0]\}$ and $a_2 = a_3 \cup \{[o, 1]\}$ 2- if $a_3 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_3\}$
R7	CP2	1- if $\exists a_1 \in \mathcal{C}$ such that $\exists a_2 \in \mathcal{A}$ such that $a_1 \subseteq a_2$ 2- if $a_2 \notin \mathcal{C}$ then $\mathcal{C} \rightarrowtail \mathcal{C} \cup \{a_2\}$

Figure 5: Expansion Rules for *Alt*-Tableau

T7 We denote by $nbrI(c)$ the maximum number of individuals that can be created in S^{i+1} by a constraint $c = [a, s, C] \in \mathcal{L}^i$. By recurrence on C we can prove that $nbrI([a, s, C]) \leq \|C\|$:

$$\begin{array}{rcl} A \in \mathcal{CN} & \rightarrow & nbrI(c) = 0 \\ D \sqcap E & \rightarrow & nbrI(c) = nbrI([a, s, D]) + nbrI([a, s, E]) \\ & & \text{(note that } \|D\| + \|E\| < \|C\|) \\ D \sqcup E & \rightarrow & nbrI(c) = nbrI([a, s, D]) + nbrI([a, s, E]) \\ \forall R.D & \rightarrow & nbrI(c) = 0 \text{ since R}\forall \text{ only create new constraints in } \mathcal{L}^{i+1} \\ \exists R.D & \rightarrow & nbrI(c) \leq 1 \text{ since constraints on D are in } \mathcal{L}^{i+1} \end{array}$$

Defining $nbrO(c)$ as the maximum number of OrIds that can be created by a constraint c, one can also prove by a similar argument that $nbrO([a, s, C]) \leq \|C\|$.

T8 From (T5), constraints created in $\mathcal{L}^{i+1}$ from $\mathcal{L}^i$ have the form $[a, s, D] \in \mathcal{A}^i \times S^{i+1} \times sub(C_0)$. There is at most $|\mathcal{A}^i|.|S^{i+1}|.\|C_0\|$ such constraints. From (T7) the expansion of each such constraints can create at most $\|C_0\|$ new individuals in S^{i+1} and also at most $\|C_0\|$ new OrIds in $\mathcal{O}^{i+1}$. Hence, $|S^{i+2}| \leq |\mathcal{A}^i|.|S^{i+1}|.\|C_0\|^2$ and $|\mathcal{O}^{i+1}| \leq |\mathcal{O}^i| + |\mathcal{A}^i|.|S^{i+1}|.\|C_0\|^2$. From the initialization, and (T7) it is easy to see that $|S^0| = 1$, $|\mathcal{O}^0| \leq \|C_0\|$, and $|S^1| \leq \|C_0\|$. Since the depth of the tree is finite (by T6), the sets $\mathcal{O}$ and S are finite, hence the sets $\mathcal{A}$, $\mathcal{C}$, $\mathcal{L}$, $\mathcal{E}$ and $\mathcal{P}$.

The maximum number of elements in a ProofContext is finite (T8). Each rule application adds at least one element in the ProofContext (T2). Hence the algorithm terminate.

Lemma 2.3 *when the algorithm terminates, the ProofContext is a pre-Alt-Tableau*

This is quite obvious since properties A$\neg$,A$\sqcap$,A$\forall$,A$\exists$ can be proven from the corresponding expansion rules. Property A1 is ensured by the initialisation of the ProofContext . Finally, property A7 is straightforwardly deduced from R$\neg$. Property A$\sqcup$ is deduced from rule R$\sqcap$ as follow :

A$\sqcup$: if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \notin \mathcal{C}$ then either

- $[a_1, s, D \sqcup E] \in \mathcal{P}$, so the rule has been expanded, and $\exists o \in \mathcal{O}$ such that $[a_1 \cup \{[o, 0]\}, s, D] \in \mathcal{L}$ and $[a_1 \cup \{[o, 1]\}, s, D] \in \mathcal{L}$. Then for a complete a_2 such that $a_1 \subseteq a_2$, either $a_1 \cup \{[o, 0]\} \subseteq a_2$ or $a_1 \cup \{[o, 1]\} \subseteq a_2$
- $[a_1, s, D \sqcup E] \notin \mathcal{P}$, then from rule R$\sqcup$ ($\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $[a_2, s, D] \in \mathcal{L}$) or ($\exists a_2 \in \mathcal{A}$ with $a_2 \subseteq a_1$ and $[a_2, s, E] \in \mathcal{L}$). For all complete $a_3 \in \mathcal{A}$, $a_1 \subseteq a3 \Rightarrow a_2 \subseteq a3$.

Lemma 2.4 *In a complete ProofContext , if $\emptyset \notin \mathcal{C}$ then there is a complete alternative a_* such that $a_* \notin \mathcal{C}$*

from rule R6, it is easy to prove that if $\forall a_1 \in \mathcal{A}$ with $card(a_1) = n$, $a_1 \in \mathcal{C}$ then $\forall a_2 \in \mathcal{A}$ with $card(a_2) = n-1$, $a_2 \in \mathcal{C}$: given a with $card(a) = n-1 < card(\mathcal{O})$, suppose $o \in \mathcal{O}$ such that $[o, 0] \notin a$ and $[o, 1] \notin a$ then
$\exists a_1, a_2 \in \mathcal{A}$ with $card(a_1) = card(a_2) = n$ such that $a_1 = a \cup \{[o, 0]\}$ and $a_1 = a \cup \{[o, 1]\}$. Since $a_1, a_2 \in \mathcal{C}$ by hypothesis, $a \in \mathcal{C}$ by rule R6. So (by recurrence on n), if all complete alternatives are in $\mathcal{C}$, the empty alternative is also in $\mathcal{C}$.

Lemma 2.5 (Soundness) *When the algorithm terminate, if $\emptyset \notin \mathcal{C}$ then the ProofContext is an Alt-Tableau .*

Direct consequence of lemma 2.3 and 2.4

Lemma 2.6 (Completeness) *If the $\mathcal{ALC}$-concept C_0 has an Basic-Tableau $\mathcal{T}_b$, the ProofContext $\mathcal{P}$ constructed by the algorithm is an Alt-Tableau .*

By lemma 2.3 $\mathcal{P}$ is in fact a pre-*Alt*-Tableau . In order to find a not clashing complete alternative a_* in $\mathcal{P}$, we can map the individuals in $\mathcal{P}$ to individuals in $\mathcal{T}$ in such a way (following the method given in [4]) that the constraints on individuals in $\mathcal{P}$ are a subset of the constraints on the mapped individuals in $\mathcal{T}$. We construct an alternative a_+ and a function $\pi : S \to S_b$ such that :

$$(*) \left\{ \begin{array}{l} \forall [a_1, s, C] \in \mathcal{L} \text{ with } a_1 \odot a_+ : D \in \mathcal{L}_b(\pi(s)) \\ \forall [a_1, s_1, s_2, R] \in \mathcal{E} \text{ with } a_1 \odot a_+ : [\pi(s_1), \pi(s_2)] \in \mathcal{E}_b \end{array} \right|$$

The (*) property can be preserved when applying expansion rules :

R$\forall$ if $[a_1, s, D \sqcap E] \in \mathcal{L}$ and $a_1 \odot a_+$, then $D \sqcap E \in \mathcal{L}_b(\pi(s))$ then (B$\sqcap$) $D \in \mathcal{L}_b(\pi(s))$ and $E \in \mathcal{L}_b(\pi(s))$. Applying rule R$\sqcap$ produce $[a_1, s, D]$ and $[a_2, s, E]$, so property (*) is not violated.

R$\sqcup$ if $[a_1, s, D \sqcup E] \in \mathcal{L}$ and $a_1 \odot a_+$, then $D \sqcup E \in \mathcal{L}_b(\pi(s))$ then (B$\sqcup$) $D \in \mathcal{L}_b(\pi(s))(1)$ or $E \in \mathcal{L}_b(\pi(s))(2)$. Suppose R$\sqcap$ is applied, producing $[a_1 \cup \{[o_n, 0]\}, s, D]$ and $[a_1 \cup \{[o_n, 1]\}, s, E]$.
Suppose (1) hold, then by setting $a_+ \leftarrow a_+ \cup \{[o_n, 0]\}$ property (*) is preserved since $a_1 \cup \{[o_n, 1]\}$ is not compatible with a_+, hence only D is required in $\mathcal{L}_b(\pi(s))$. Note also that all constraints with alternative a compatible with the old value of a_+ are still compatible with the new value since o_n is a fresh OrId. Also, all constraints with alternative a not compatible with the old value of a_+ are obviously not compatible with the new value. So modifying a_+ in the way above do not affect property (*) for previously expanded constraints.
Suppose (2) old, setting $a_+ \leftarrow a_+ \cup \{[o_n, 1]\}$ preserves property (*) by a similar argument.

R∃ if $[a_1, s, \exists R.D] \in \mathcal{L}$ and $a_1 \odot a_+$, then $\exists R.D \in \mathcal{L}_b(\pi(s))$ then (B∃) $\exists y \in S_b$ such that $[\pi(s), y] \in \mathcal{E}_b$ and $D \in \mathcal{L}_b(y)$.
Applying R∃ generates a new variable $t \in S$ and the constraints $[a_1, t, D] \in \mathcal{L}$ and $[a_1, s, t, R] \in \mathcal{E}$. Setting $\pi \leftarrow \pi \cup [t, y]$ yields a function satisfying property (*)

R∀ if $[a_1, s, \forall R.D] \in \mathcal{L}$ and $a_1 \odot a_+$, then $\forall R.D \in \mathcal{L}_b(\pi(s))$. If $[a_2, s, t, R] \in \mathcal{E}$ and $a_2 \odot a_+$, then $\pi(s), \pi(t)] \in \mathcal{E}_b$. Hence, by rule (B∀), $D \in \mathcal{L}_b(\pi(t))$. Hence applying R∀, producing $[a_1 \cup a_2, t, D]$, do not violate property (*).

Initialy, we set $a_+ = \emptyset$, and $\pi = \{[s_0, x_0]\}$ for some $x_0 \in S_b$ with $C_0 \in \mathcal{L}_b(x_0)$ (x_0 exists since $\mathcal{T}_b$ is a *Basic*-Tableau). Property (*) is verified.

When the algorithm terminate, suppose $a_+ \in \mathcal{C}$ then, since an alternative can only produce clashes (by rule R¬, R6, R7) in compatible alternatives :

$$\begin{aligned} a_+ \in \mathcal{C} \quad &\Rightarrow \quad \exists a_1, a_2 \text{ such that } a_1 \odot a_2 \text{ and } a_1 \odot a_+ \text{ and } a_2 \odot a_+ \text{ and} \\ &\qquad [a_1, s, A] \in \mathcal{L} \text{ and } [a_2, s, \neg A] \in \mathcal{L} \\ &\Rightarrow \quad (\text{since } a_1 \odot a_+ \text{ and } a_2 \odot a_+) A \in \mathcal{L}_b(\pi(s)) \text{ and } \neg A \in \mathcal{L}_b(\pi(s)) \end{aligned}$$

This is impossible since $\mathcal{T}_b$ is a *Basic*-Tableau . So $a_+ \notin \mathcal{C}$. By R6,R7 (see lemma 2.4) $\exists a_*$ such that a_* is complete and $a_+ \subseteq a_*$ and $a_* \notin \mathcal{C}$

Theorem 2.1 *The algorithm on Alt-Tableau presented in this paper is a decision procedure for the satisfiability of $\mathcal{ALC}$-concept*

This theorem is a direct consequence of lemma 2.1, 2.2, 2.5, and 2.6.

3 Conclusion

We have presented a new tableau structure and proved its applicability to DL by providing a sound and complete algorithm for $\mathcal{ALC}$-concept satisfiability checking relying on such structure. Obviously, many extensions are still to be defined to demonstrate the claims that it will be well suited for proof optimization, in particular with ABoxes and Concrete Domains. Nevertheless, this *Alt*-Tableau algorithm has been implemented in the CICLOP reasoner, and preliminary tests with toy examples are promising. An in-depth study with both constructed and real TBoxes and ABoxes have still to be carried out.

References

[1] F. de Beuvron, M. Kullmann, D. Rudloff, M. Schlick, and F. Rousselot. The description logic reasoner ciclop (version 2.0). In *Proceedings of the 7th Workshop on Automated Reasoning*, London, Great Britain, 2000.

[2] Volker Haarslev and Ralf Mller. Description of the racer system and its applications. In *Proceedubgs International Workshop on Description Logics (DL-2001)*, Stanford, USA, 2001.

[3] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[4] I. Horrocks and U. Sattler. A description logic with transitive and converse roles and role hierarchies. In *Proceedings of the International Workshop on Description Logics*, Povo - Trento, Italy, 1998. IRST.

[5] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reasoning for expressive description logics. In Harald Ganzinger, David McAllester, and Andrei Voronkov, editors, *Proceedings of the 6th International Conference on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer-Verlag, September 1999.

[6] Martina Kullmann. *Description Logic Models for Supporting Decision Making.* PhD thesis, Université Louis Pasteur, France, 2001.

[7] C. Lutz. Nexptime-complete description logics with concrete domains. In *Proceedings of the International Joint Conference on Automated Reasoning*, LNAI, Siena, Italy, 2001. Springer Verlag.

[8] P. F. Patel-Schneider. Dlp system description. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 87–89, 1998.