A Connection Method for the Description Logic $\mathcal{ALC}$

Fred Freitas

Informatics Center, Federal University of Pernambuco (CIn - UFPE), Brazil
fred@cin.ufpe.br

1 Introduction

The problem of reasoning over ontologies written in Description Logic (DL) [1] has received strong interest, particularly after the Semantic Web inception. The necessity of creating reasoners to deal with OWL (Ontology Web Language) [2], in its various versions, posed interesting research questions for inference systems, making the problem earn the reputation of being complex, whose users see solutions as black boxes. Consequently, not many inference systems became well known to the public; indeed, tableaux methods [1] took over the field. Apart from the reasoning algorithms, many other practical issues have been brought about, stemming from the natural features of the Web, that ask for certain requirements for inferencing. Among them, the use of memory is certainly an important asset for a good reasoning performance. Therefore, methods that do not require a huge amount of memory may be promising.

This paper tries to tackle this issue by adapting a reasoning method that makes a parsimonious usage of memory. The inference system proposed here is based on the connection calculus [3], which is a clear and effective inference method applied successfully over first order logic (FOL). Its main features meet with the demands: it keeps only one copy of each logical sentence in memory and it does not derive new sentences from the stored ones. The formal system, the $\mathcal{ALC}$ Connection Method ($\mathcal{ALC}$ CM), displays some desirable features for a DL reasoner,: it is able to perform all DL inference services, namely, subsumption, unsatisfiability (or inconsistency), equivalence and disjointness. As the most widespread DL tableaux systems, $\mathcal{ALC}$ CM reduces them to subsumption or validity - the dual of unsatisfiability, used in most DL systems (interested readers should check that information at [7]).

The rest of the article is organized as follows. Section 2 defines one of the most basic description logic languages, $\mathcal{ALC}$ (Attributive Concept Language with Complements) [1], together with the disjunctive normal formal used by the method. $\mathcal{ALC}$ was chosen as the starting of the work, because it constitutes the foundation of many other Description Logics. Section 3 brings a proposed $\mathcal{ALC}$ ontologies' matrix normal form that fits to the connection method. Section 4 describes the $\mathcal{ALC}$ adapted connection method and states the key logical properties for inference systems (soundness, completeness and termination). Section 5 brings discussion in the light of representation and reasoning, and section 6 presents future work and conclusions.

2 The Description Logic $\mathcal{ALC}$

Description Logics (DLs for short) are a family of knowledge representation formalisms that have been gaining growing interest in the last two decades, particularly after OWL (Ontology Web Language) [2], was approved as the W3C standard for representing the most expressive layer of the Semantic Web.

An ontology or knowledge base in $\mathcal{ALC}$ is a set of axioms a_i defined over the triple (N_C, N_R, N_O) [1], where N_C is the set of *concept names* or *atomic concepts* (unary predicate symbols), N_R is the set of *role or property names* (binary predicate symbols), and N_O the set of *individual names* (constants), instances of N_C and N_R.

N_C contains concepts (like Bird, Animal, etc) as well as other concept definitions as follows. If r *is* a *role* ($r \in N_R$) and C and D are *concepts* ($C, D \in N_C$) then the following definitions belong to the *set of $\mathcal{ALC}$ concepts*: (i) $C \sqcap D$ (the intersection of two concepts); (ii) $C \sqcup D$ (the union of two concepts); (iii) $\neg C$ (the complement of a concept); (iv) $\forall r.C$ (the universal restriction of a concept by a role); and (v) $\exists r.C$ (the existential restriction of a concept by a role). Note that, in the definitions above, C and D can be inductively replaced by other complex concept expressions.

There are two axiom types allowed in $\mathcal{ALC}$: (i) *Assertional axioms*, which are concept assertions C(a), or role assertions R(a,b), where $C \in N_C$, $r \in N_R$, $a,b \in N_O$ and (ii) *Terminological axioms*, composed of any finite set of GCIs (*general concept inclusion*) in one of the forms $C \sqsubseteq D$ or $C \equiv D$, the latter meaning $C \sqsubseteq D$ and $D \sqsubseteq C$, C and D being concepts. An ontology or knowledge base (*KB*) is referred to as a pair $(\mathcal{T},\mathcal{A})$, where $\mathcal{T}$ is the *terminological box* (or *Tbox*) which stores terminological axioms, and $\mathcal{A}$ is the *assertional box (ABox)* which stores assertional axioms. $\mathcal{T}$ may contain *cycles*, in case at least an axiom of the form $C \sqsubseteq D$, D can be expanded to an expression that contains C.

There are two major ways of defining $\mathcal{ALC}$ semantics. The most frequently used one relies on the definitions of interpretation, model, fixpoints, etc, over a domain Δ [1]. Another way is mapping $\mathcal{ALC}$ constructs to first order logic (FOL) (like in [6]) and exploiting the semantics defined for FOL. FOL semantics is available, for instance in [3]. The latter was chosen, with the purpose of taking advantage of the already formalized completeness and soundness theorems for the work.

Next, the $\mathcal{ALC}$ disjunctive normal form used by the inference system is presented.

2.1. An $\mathcal{ALC}$ Positive Matrix Form

Definition 1 ($\mathcal{ALC}$ disjunctive normal form ($\mathcal{ALC}$ DNF), clause). An *$\mathcal{ALC}$ DNF formula* is a disjunction of conjunctions, in the form $C_1 \vee ... \vee C_n$, where each C_i is a *clause.* Clauses are conjunctions of $\mathcal{ALC}$ predicates ($\mathcal{ALC}$ concepts or relations, instantiated or not), like $L_1 \wedge ... \wedge L_m$, also denoted as $\{L_1, ..., L_m\}$. $\mathcal{ALC}$ formulae can be also expressed in *$\mathcal{ALC}$ disjunctive clausal form* or *$\mathcal{ALC}$ positive matrix form*, as $\{C_1, ..., C_n\}$,. A formula stated this way is called an *$\mathcal{ALC}$ matrix*. In an $\mathcal{ALC}$ matrix, each clause occupies a column.

To reach this normal form, the first two actions to be made over the axioms are: (i) splitting equivalence axioms of the form $C \equiv D$ into two axioms $C \sqsubseteq D$ and $D \sqsubseteq C$, and (ii) converting all the axioms into a *Negated Normal Form* (NNF), in which negations occurs only on literals [1]. Next, some necessary definitions are introduced.

Definition 2 ($\mathcal{ALC}$ disjunction, $\mathcal{ALC}$ conjunction). An $\mathcal{ALC}$ disjunction is either a literal, a disjunction $E_0 \sqcup E_1$ or an universal restriction $\forall r.E_0$. An $\mathcal{ALC}$ conjunction is either a literal, a conjunction $E_0 \sqcap E_1$ or an existential restriction $\exists r.E_0$. E_0 and E_1 are arbitrary concept expressions.

Definition 3 ($\mathcal{ALC}$ pure disjunction). The set S_D of $\mathcal{ALC}$ pure disjunctions is the smallest set where: (i) $D_0 \in S_D$ for every literal D_0; (ii) $D_0 \sqcup D_1 \in S_D$, in case $D_0, D_1 \in S_D$; (iii) if $D_0 \in S_D$ then $\forall r.D_0 \in S_D$. An element $\breve{D} \in S_D$ is an $\mathcal{ALC}$ pure disjunction. An $\mathcal{ALC}$ *non-pure disjunction* is an $\mathcal{ALC}$ disjunction that is not pure.

Definition 4 ($\mathcal{ALC}$ *pure* conjunction). The set S_C of $\mathcal{ALC}$ pure conjunctions is the smallest set where: (i) $C_0 \in S_C$ for every literal C_0; (ii) $C_0 \sqcap C_1$ S_C iff $C_0, C_1 \in S_C$; and (iii) if $C_0 \in S_C$ then $\exists r.C_0 \in S_C$. An element $\hat{C} \in S_C$ is an $\mathcal{ALC}$ pure conjunction. An $\mathcal{ALC}$ *non-pure conjunction* is an $\mathcal{ALC}$ conjunction that is not pure.

Definition 5 (Impurity on a non-pure expression). Impurities on non-pure $\mathcal{ALC}$ DL expressions are either conjunctive expressions in a non-pure disjunction or disjunctive expressions in a non-pure conjunction. The set of impurities is called $\mathcal{ALC}$ *impurity set*, and is denoted by S_I.

Example 1 (Impurities on non-pure expressions).
The expression $(\forall r.(D_0 \sqcup ... \sqcup D_n \sqcup (C_0 \sqcap ... \sqcap C_m) \sqcup (A_0 \sqcap ... \sqcap A_p))$, a non-pure disjunction, contains two impurities: $(C_0 \sqcap ... \sqcap C_m)$ and $(A_0 \sqcap ... \sqcap A_p)$.

Definition 6 (Positive normal form). An $\mathcal{ALC}$ axiom is in positive normal form iff it is in one of the following forms: (i) $\hat{C} \sqsubseteq \breve{D}$; (ii) $C \sqsubseteq \exists r.\hat{C}$; and (iii) $\forall r.\breve{D} \sqsubseteq C$; where C is a concept name, $\hat{C}$ a pure conjunction and $\breve{D}$ a pure disjunction.

In [7], algorithms for transforming $\mathcal{ALC}$ axioms to this normal form are available.

2.2 Translation Rules for the normalization

With all axioms in the normal form defined just above, it is easy to map them to matrix form, by applying the rules given in Table 1. Table 2 brings the mapping treatment of recursive sub-cases of existential and universal restrictions, when they occur inside any of the the normal form $\hat{C} \sqsubseteq \breve{D}$,. An improvement of the approach is, as the usual DL notation, variables are not needed, since all relations are binary and dash lines make for disambiguation, as soon will be seen.

Some important remarks must be stated at this point for the sake of clarity.

The first is that the whole knowledge base *KB* is negated during this transformation. This is due to the fact that in order to prove the validity of $KB \models \alpha$ (α being a subsumption axiom) using a direct method, $\neg KB \vee \alpha$ must be proven a tautology (coming from $KB \rightarrow \alpha$, according to the Deduction Theorem [3]).

Table 1. Translation rules to map $\mathcal{ALC}$ into FOL positive NNF and matrices.

Axiom type	FOL Positive NNF mapping	Matrix
$\mathrm{C} \sqsubseteq \exists \mathrm{r}.\hat{\mathrm{C}}$, where $\hat{\mathrm{C}} = \bigwedge_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{A_i}$ with $\mathrm{A_i} \in S_C$ (pure conjunction)	$(C(x) \wedge \neg r(x,f(x))) \vee (C(x) \wedge \neg A_1(f(x))) \vee \ldots \vee (C(x) \wedge \neg A_n(f(x)))$	$\begin{bmatrix} C & C & \cdots & C \\ \neg r & \neg A_1 & \cdots & \neg A_n \end{bmatrix}$
$\forall \mathrm{r}.\breve{\mathrm{D}} \sqsubseteq \mathrm{C}$, where $\breve{\mathrm{D}} = \bigvee_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{A'_j}$ with $A'_j \in S_D$ (pure disjunction)	$(\neg r(x,f(x)) \wedge \neg C(x)) \vee (\neg A'_1((f(x)) \wedge \neg C(x)) \vee \ldots \vee (\neg A'_m((f(x)) \wedge \neg C(x))$	$\begin{bmatrix} \neg r & A'_1 & \cdots & A'_m \\ \neg C & \neg C & \cdots & \neg C \end{bmatrix}$
$\hat{\mathrm{C}} \sqsubseteq \breve{\mathrm{D}}$, where $\hat{\mathrm{C}} = \bigwedge_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{A_i}$, $\breve{\mathrm{D}} = \bigvee_{\mathrm{j}=1}^{\mathrm{m}} \mathrm{A'_j}$ with $\mathrm{A_i} \in S_C$ (pure conjunction) and $\mathrm{A'_j} \in S_D$ (pure disjunction)	$A_1(x) \wedge \ldots \wedge A_n(x) \wedge \neg A'_1(x) \wedge \ldots \wedge \neg A'_m(x)$	$\begin{bmatrix} A_1 \\ \vdots \\ A_n \\ \neg A'_1 \\ \vdots \\ \neg A'_n \end{bmatrix}$

Table 2. Recursive sub-cases of restrictions for axioms of type $\hat{\mathrm{C}} \sqsubseteq \breve{\mathrm{D}}$,.

Axiom type	FOL Positive NNF mapping	Matrix
A is an existential restriction: ... $\sqcap \exists \mathrm{r}.\mathrm{A} \sqcap$ ... , with $\mathrm{A} \in S_C$ (pure conjunction)	... $\wedge$ $r(x,y) \wedge$ $A(y) \wedge$ ...	$\begin{bmatrix} \vdots \\ r \\ A \\ \vdots \end{bmatrix}$
A' is an universal restriction: ...$\sqcup \forall \mathrm{r}.\mathrm{A'} \sqcup$ ..., with $\mathrm{A'} \in S_D$ (pure disjunction)	… $\wedge$ $r(x,y) \wedge$ $\neg A'(y) \wedge$ …	$\begin{bmatrix} \vdots \\ r \\ \neg A' \\ \vdots \end{bmatrix}$

The first consequence of that is that subsumption axioms of the form $\mathrm{C} \sqsubseteq \mathrm{D}$, which are usually logically translated as $\mathrm{C} \rightarrow \mathrm{D}$, because negated ($\neg(\mathrm{C} \rightarrow \mathrm{D})$, indeed), are now translated to $\mathrm{C} \wedge \neg \mathrm{D}$, instead of $\neg \mathrm{C} \vee \mathrm{D}$. Moreover, to establish a uniform set of

rules applicable over $\mathcal{ALC}$ formulae, $\neg\alpha$ is dealt with, instead of α, so as to consider $\mathcal{ALC}$ axioms as $\neg A_1 \vee \ldots \vee \neg A_n \vee \alpha$ (coming from $\neg A_1 \vee \ldots \vee \neg A_n \vee \neg\neg\alpha$), where $A_i \in \mathcal{T}$ (axioms in the TBox). The translation rules are then applied over $\neg\alpha$ and all A_i.

Another remark regards *implicit skolemization***.** In the original CM, once the whole *KB* should be negated, universal quantifiers (∀) are skolemized instead of existential, and all variables in the resulting DNF are then (implicitly) existentially quantified. Aiming at keeping a close correspondence with the original CM, only the cases in which $\mathcal{ALC}$ axioms, when translated to FOL, contain Skolem functions need to solved. For instance, the axiom $\mathsf{C} \sqsubseteq \exists \mathrm{r}.\,\mathrm{B}$, negated, translates to FOL Skolem DNF as $\exists x(A(x) \wedge \neg r(x,f(x))) \vee (A(x) \wedge \neg B(f(x)))$ (coming from the skolemization of $\forall y$ in $\exists x\,(A(x) \wedge (\forall y\,(\neg r(x,y) \wedge \neg B(y))))$. In such cases, vertical and horizontal dash lines are employed to stand for the relations in the matrix. Vertical dash lines represent existential restrictions (∃r.C), which are not skolemized, while horizontal dash lines represent universal restrictions ($\forall \mathrm{r}.\,\mathsf{C}$), in which the qualifier concept corresponds to a skolemized concept. An example should makes things clearer.

Example 2 (Positive normal form, skolemization, clause, matrix). The query

$$\left.\begin{array}{l}\mathrm{Animal} \sqcap \exists \mathrm{hasPart.Bone} \sqsubseteq \mathrm{Vertebrate} \\ \mathrm{Bird} \sqsubseteq \mathrm{Animal} \sqcap \exists \mathrm{hasPart.Bone} \sqcap \exists \mathrm{hasPart.Feather}\end{array}\right\} \models \mathrm{Bird} \sqsubseteq \mathrm{Vertebrate}$$

reads in FOL as *(∀w(Animal(w) ∧ (∃z (hasPart(w,z) ∧ Bone(z)) → Vertebrate(w))) ∧ (∀x(Bird(x)→Animal(x)∧ ∃y(hasPart(x,y)∧ Bone(y))∧ ∃v(hasPart(x,v) ∧ Feather(v))) → ∀t(Bird(t) → Vertebrate(t)))*

where the variables *y* and *t* were respectively skolemized by the function *f(x)* and the constant *c*. In positive matrix clausal form, the formula is represented by

{{*Bird(x)* ,¬*Animal(x)*}, {*Bird(x)* ,¬*hasPart(x,f(x))*}, {*Bird(x)* ,¬*Bone(f(x))*}, {*Bird(x)* ,¬*hasPart(x,g(x))*}, {*Bird(x)* ,¬*Feather(g(x))*}, {*Animal(w), hasPart(w,z), Bone(z),* ¬*Vertebrate(w)*}, {¬*Bird(c)*}, {*Vertebrate(c)*)}}. Figure 1 shows it as a matrix.

Bird	*Bird*	*Bird*	*Bird*	*Bird*	*Animal*	¬*Bird*(*c*)	*Vertebrate*(*c*)
¬*Animal*	¬*hasPart*	¬*Bone*	¬*hasPart*	¬*Feather*	*hasPart*		
					Bone		
					¬*Vertebrate*		

Figure 1. An $\mathcal{ALC}$ formula and query in disjunctive clausal form represented as a matrix.

The vertical line connecting the symbols hasPart and Bone in the antepenultimate column has the function of denoting that the concept Bone has implicitly in FOL as argument the variable that is the 2nd argument of the relation hasPart (coming from the FOL translation *∃x∃y hasPart(x,y) ∧ Bone(y)* that arose from the axiom ∃hasPart.Bone ⊑ Vertebrate. On its turn, the horizontal dash line connecting the symbols ¬hasPart and ¬Feather represents the $\mathcal{ALC}$ expression ∃hasPart.Feather in the original axiom Bird ⊑ ∃hasPart.Feather. This expression, negated, corresponds in FOL Skolem NF to *∃x ¬hasPart(x,f(x)) ∨¬Feather(f(x))*. Thus, during reasoning, skolemized predicates such as the last ones can only be unified with variables, and not with concrete individuals or other with distinct Skolem functions.

Dash lines can superpose other dash lines, for instance to represent the axiom $A \sqsubseteq \exists r_1.(\exists r_2.B)$. In this normal form they cannot cross, although this would not consist in a problem for the $\mathcal{ALC}$ method.

A last remark regards memory usage. By using this normal form, the number of newly introduced symbols (and also the number of axioms) grows linearly with the number of impurities. It is in most cases a gain compared with other normalizations (e.g., $\mathcal{EL}$ [4] and resolution [6]), whose number of new axioms grows linearly to the axioms' length, although in the worst case they are equal. See [7] for more examples and further discussion regarding this topic.

3 An $\mathcal{ALC}$ Connection Method

The connection method (CM) [3] was created by W. Bibel in the 70s, and earned good reputation in the field of automated theorem proving in the 80s and 90s. Apart from needing less memory than other first-order logic inference systems, it offers the same advantages, according to its author, since any reductions, compressions, optimizations and improvements that fit to resolution or tableaux in can also be properly applied to it (see [3], 4.4). As a result, it is probably feasible to finding ways to implement typical DL tableaux optimizations to the $\mathcal{ALC}$ connection method under development, although such issues are out of the scope of the current paper.

Such distinctive features naturally led to the idea of proposing a connection method especially tailored to infer over description logic Semantic Web ontologies. The core of the paper, the $\mathcal{ALC}$ connection calculus, is explained next.

3.1 An $\mathcal{ALC}$ Connection Sequent and Matrix Calculus

Definition 7 (Path, connection, unifier, substitution). A *path* is a set of literals from a matrix in which every clause (or column) contributes with one literal. A *connection* is a pair of complementary literals from different clauses, like $\{L_1^\sigma, \neg L_2^\sigma\}$, where $\sigma(L_1)$ (or $\sigma(\overline{L_2})$) is the most general unifier (mgu) between predicates L_1 and $\neg L_2$. σ is the set of *substitutions*, which are mappings from variables to terms. All occurrences of the variable contained in a substitution would be replaced by the term indicated in σ.

Example 4 (Path, connection, unifier, substitution). Some paths of the matrix of Figure 1 are*: {Bird(x), ¬Bone(f(x)), Animal(w) ,¬Bird(c), Vertebrate(c)}* and *{¬Animal(w), Bird(x), ¬Bone(f(x)), Animal(w) ,¬Bird(c), Vertebrate(c)}*. Some connections from the matrix are *{Bird(x), ¬Bird(c)}*, *{¬hasPart(x,f(x)), hasPart(w,z)}*, and *σ={x/c, w/c, z/f(c)}* is the whole matrix' mgu.

Lemma 1 (Validity, active path, set of concepts). An $\mathcal{ALC}$ formula represented as a matrix M is *valid* when every path contains a connection $\{L_1, \neg L_2\}$, provided that $\sigma(L_1) = \sigma(\neg\overline{L_2})$. This is due to the fact that a connection represents the tautology $L_1^\sigma \vee \neg L_2^\sigma$ in DNF. As a result, the connection method aims at finding a connection in each path, together with a unifier for the whole matrix. During the

proof, the current path is called *active path* and denoted by $\mathcal{B}$. The *set of concepts* τ of a variable or instance *x* during a proof is defined by $\tau(x) \stackrel{\text{def}}{=} \{C | C(x) \in \mathcal{B}\}$ [9].

Definition 8 ($\mathcal{ALC}$ connection calculus in sequent style). Figure 2 brings the $\mathcal{ALC}$ connection calculus rules in sequent style of the, adapted from [10]. The basic structure is the tuple *{C,M,Path}*, where the clause C is the open sub-goal, M is the matrix corresponding to the query $KB \models \alpha$, and *Path* is the active path.

$$Axiom\ (Ax)\ \frac{}{\{\}, M, Path}$$

$$Start\ Rule\ (St)\ \frac{C_2, M, \{\}}{\varepsilon, M, \varepsilon}$$
$$where\ M\ is\ the\ matrix\ KB \models \alpha,\ C_2\ is\ a\ copy\ of\ C_1 \in \alpha$$

$$Reduction\ Rule\ (Red)\ \frac{C^\sigma, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$$
$$with\ \sigma(L_1) = \sigma(\overline{L_2})$$

$$Extension\ Rule\ (Ext)\ \frac{C_2^\sigma \backslash \{L_2^\sigma\}, M, Path \cup \{L_1\} \quad C^\sigma, M, Path}{C \cup \{L_1\}, M, Path}$$
$$with\ \ C_2\ a\ copy\ of\ C_1 \in M, L_2 \in C_2, \sigma(L_1) = \sigma(\overline{L_2}),$$

$$Copy\ Rule\ (Cop)\ \frac{C \cup \{L_1\}, M \cup \{C_2^\mu\}, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$$
$$with\ L_2 \in C_2, \mu \leftarrow \mu + 1, and$$
$$\left(x_\mu^\sigma \notin N_O\ or\ \tau(x_\mu^\sigma) \nsubseteq \tau\left(x_{\mu-1}^\sigma\right)\right)\ with\ \sigma(L_1) = \sigma(\overline{L_2})\ (blocking\ conditions)$$

Figure 2. The $\mathcal{ALC}$ connection sequent calculus rules (adapted from [10]).

Rules are applied bottom-up. To start, the matrix *M* representing $KB \models \alpha$ is put in the bottom of the *St* rule, and a clause $C_1 \in \alpha$ is chosen as the first clause. Then, the three last rules are applied. The key rule for the calculus is the *Extension rule*, once it finds connections of the form $\{C(x), \neg C(y)\}$, ($\{C, \neg C\}$ in the notation used here) or {*r(x,y),¬r(z,w)*}and the proper unifier σ for it (In the first case σ={*x/y*} and in the second σ={*x/z, y/w*}). Besides this slight change in the *Start rule* ($C_1 \in \alpha$), a blocking mechanism was included as a new rule, the *Copy rule.* If the current literal can only connect a clause already in the active path, this clause is copied to the matrix, and the indexing function $\mu: M \rightarrow \mathbb{N}$, that assigns the number of copies of each clause, is incremented. The use of this function avoids ordering of individuals, as done in the original CM. An important remark about the rule is that the copy is virtual, in the sense that only an index μ is created. CM requires only one copy of the matrix in memory. This is one of its main advantages. Its use of memory may be better than tableaux in cases the proof demands many derivations, like with cyclic ontologies.

Blocking didn't occur in the original CM due to FOL semi-decidability, but it consists in a common practice in DL to guarantee termination. The first action to be accomplished is incrementing μ. Then, before making a copy of clause C_2 (this copy

will be named C_2^{μ}), it is necessary to check whether the set of concepts τ associated to the variable x_{μ}^{σ} (i.e., if the new x_{μ} was unified) of the new literal L_2^{μ} from C_2^{μ} being created by the Cop rule is not contained in the set of concepts of the original variable $x\,(\tau(x^{\sigma}))$, from $L_2(x)$ of C_2 [9].

Examples of the $\mathcal{ALC}$ CM calculus' application can be found at [7].

Theorem 1 (Soundness and completeness) An $\mathcal{ALC}$ formula in positive matrix form is valid iff there is a connection proof for "$\varepsilon, M, \varepsilon$", i.e. the application of the rules makes the *Axiom rule (Ax)* applicable to all leaves of the generated tree.

Proof The introduced changes in rules do not lose the soundness of the original CM (whose proof is available at [3], III.3.9 and 3.10), because such changes made the new rules only more restrictive, so as to making them fire in less cases than in the original CM. They also do not affect completeness, *viz*: (i) The original *Start rule* allows any clause to start the method. By restricting it to begin just with a literal from the consequent to be proven (α), proofs departing from possibly inconsistent matrices are avoided, from which any axiom can be summoned. In other words, it ensures the participation of α in the proof. If the axiom α holds, then a proof starting by one of its literals is found by the original CM as well. (ii) The blocking conditions hold only when: (a) a new clause is being created from a variable (thus, it is not an instance, as stated by the first blocking condition $x_{\mu}^{\sigma} \notin N_O$), and, (b) the variable being copied is already a copy of another variable attached to the same concepts (as stated by the conditions $\tau(x_{\mu}^{\sigma}) \nsubseteq \tau(x^{\sigma})$). Therefore it prevents infinite applications of the *Copy rule* that would not help proving $KB \vDash \alpha$, and thus, such condition does not interfere in the original CM's completeness. ■

Theorem 2 (Termination). $\mathcal{ALC}$ CM always terminates.

Proof Having the mindset that the *Copy rule* is the only source of non-termination, the proof has three subcases: (i) *When KB contains no cycle*, the calculus finishes in a finite number of steps because the number of choices arising from the finite number of clauses and possible connections is also finite. Since the *Copy rule* is never fired, and there are no loops, $\mathcal{ALC}$ CM always terminates. (ii) *When KB contains cycles and* $KB \vDash \alpha$, the system always terminates, because $\mathcal{ALC}$ CM behaves exactly equal the original CM, whose termination proof can be found at [3]), except for the blocking conditions. However, blocking only plays the role of a termination condition, since it prevents the *Copy rule* to be fired, thus assuring termination; (iii) The case *when KB contains cycles and* $KB \nvDash \alpha$ is the only aspect not covered by the original CM, given FOL's semi-decidability. Nevertheless, $\mathcal{ALC}$ CM always terminates for this case, due to the fact that the *Copy rule* obliges the set of concepts of the newly created instance of literal L_2 not to be a subset of any of the sets of concepts of other introduced instances of the literal. Once every set of concepts is a subset of N_C and N_C is finite, the number of distinct sets is limited to $2^{|N_C|}$. So, the number of copies for literals is finite. Thus $\mathcal{ALC}$ CM always terminates in all three cases. ■

The system can also be expressed in an easier way using matrices to build proofs. Figure 5 portraits an example using matrices. An $\mathcal{ALC}$ CM calculus' algorithm of the system was adapted [7] from the one described in [3], III.7.2.

Example 5 ($\mathcal{ALC}$ connection calculus). Figure 3 deploys the proof of the query stated in example 1. In the figure, literals of the active path are in boxes and arcs denote connections. For building a proof, firstly a clause from the consequent (*Start rule*) is chosen, say, the clause {¬Bird(c)} and a literal from it (¬Bird(c)).

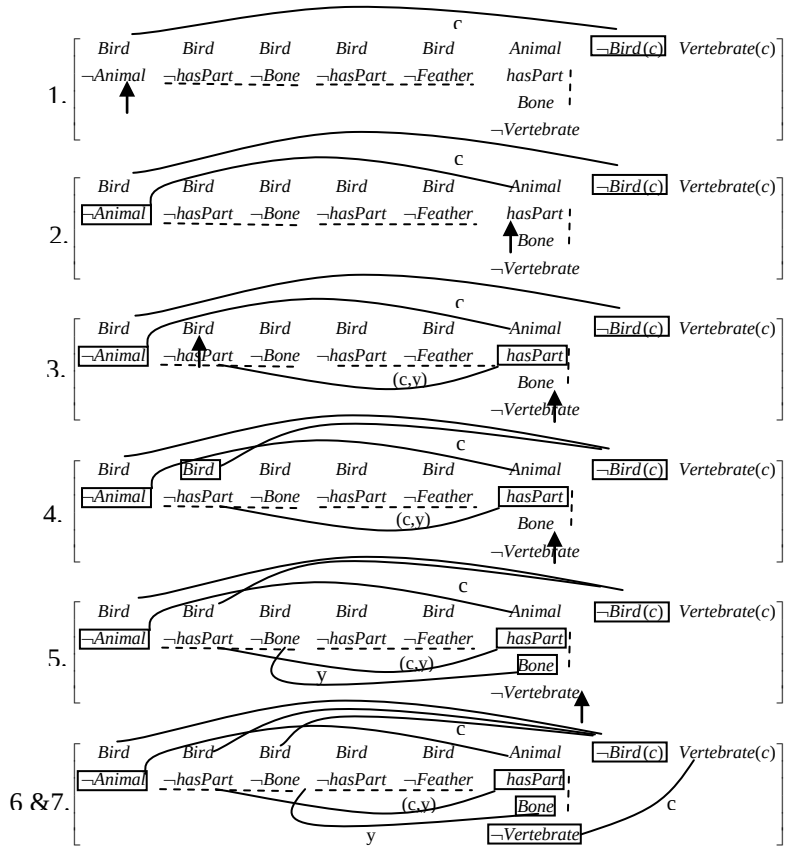

Figure 3. The connection proof example in matrix form.

Step 1 connects this clause with the first matrix clause. An instance or variable - representing a fictitious individual that is being predicating about -, appears in each arc; in this connection, the instance c. The arrow points to literals still to be checked in the clause (only the literal ¬Animal in Step 1), that should be checked afterwards. After step 2, the connection {¬Animal, Animal} is not enough to prove all paths stemming from the other clause (the one with literal ¬Animal). In order to assure that, the remaining literals from that clause, *viz* hasPart, Bone and ¬Vertebrate, have still to be connected. Then, in step 3, when the predicate hasPart is connected, instance c

being dealt with any more, but a relation between it and another variable or fictitious individual, say y (indicated by (c,y)).

Until that moment, only the *Extension rule* was applied. However, in step 4, the *Reduction rule* is used, triggered by its two enabling conditions: (i) there is a connection for the current literal already in the proof; and (ii) unification can take place. Unification would not be possible between different individuals and/or skolemized functions (in $\mathcal{ALC}$, equality among individuals is not necessary).

Next, a connection for the literal Bone was needed. That time, the variable (or fictitious individual) y is being referred, for the reason that it stands for a part of the individual c (hasPart(c,y)).

In case the system is able to summon the query, the processing finishes when all paths are exhausted and have their connections found. In case a proof cannot be entailed, the system would have tried all available options of connections, unifiers and clause copies, having backtracked to the available options in case of failure.

In [7], the interested reader could also find how the system deals with cycles and blocking, the DL services it is able to perform (such as a new way for inconsistency checking) and the reductions to subsumption and unsatisfiability.

4 Discussion

While planning the $\mathcal{ALC}$ CM, i.e. adapting the original CM to $\mathcal{ALC}$, the notation was designed in order to avoid any ambiguities to arise from the lack of variables in the representation – that was the rationale of dashlines replacing the quantifiers, so as to eliminate possible ambiguities with skolemization for the connections. This notation can be changed even further, since assertions are better stored outside the matrices, bringing two benefits: matrix size can be dramatically shrunk and powerful indexing mechanisms for retrieving assertions, such as relational databases, can be employed. However, even sticking only to TBoxes, matrices can be sparse, a memory waste. A simple solution for this consists in storing matrices as arrays of arrays.

Concerning reasoning, the first aspect to be taken into account resides in the strong similarity between CM and tableaux [10]. For that reason, reductions and optimizations designed over the latter are very likely to be incorporated to the former. Additionally, connection calculi are more goal-oriented than tableau calculi; therefore, they tend to be more efficient. As a support from this belief, this fact could be empirically observed in a comparison between a connection prover (leanCoP) and a tableaux connection prover (leanTAP) [8], although proving this requires more formal work and experiments Departing from such premises, in case the DL tableaux optimizations fit to $\mathcal{ALC}$ CM, a performance at least comparable to other leading DL tableaux approaches is expected to an $\mathcal{ALC}$ leanCoP , which is under development, given that CM itself usually displays a quite competitive performance (it already won a CASC competition once [11]). Of course, a deeper investigation on the cases in which could be advantageous to rely on tableaux or $\mathcal{ALC}$ CM is on the research agenda of this work.

5 Conclusions and Future Work

A connection method to take on the DL $\mathcal{ALC}$ was formalized in this work, by adapting the CM calculus formalized in sequent style from [10] and including a new rule. Notational improvements were also introduced, the key one being the representation without variables. Of course, this work is planned to continue in many research directions, such as implementations, other DLs, Semantic Web, etc.

First, the work presented here shall be extended to more complex description logic languages in a near future. Particularly, formalizations and implementations for the DLs $\mathcal{EL}$++, $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$ will be practically useful for applications related to the Semantic Web provided that good solutions for dealing with equality are available, like eq-connections [3].

Another potential reasoning advantage to be pursued could be the memory required for $\mathcal{ALC}$ CM. Despite the fact that, given a connection proof, the size of an equivalent tableau proof is not significantly bigger (perhaps a quadratic increase in size); the proof search might indeed take up much more space, as the search is essentially saturation-based and many parts of a problem maybe not be required for the proof. However, these statements demand additional research to be proven true.

Last but not least, lean implementations written in Prolog, in the flavor of leanCop [8], that demand small memory space, can serve applications that are constrained in memory, such as stream reasoning in mobile applications, for instance.

References

1. Baader, F., Calvanese, D. McGuinness, D,, Nardi, D., Patel-Schneider, P. (Eds.): The Description Logic Handbook. Cambridge University Press, 2003.
2. Patel-Schneider, P., Hayes, P., Horrocks, I.: OWL - Web Ontology Language Semantics and Abstract Syntax W3C Recommendation. www.w3.org/tr/2004/rec-owl-semantics-20040210/ , 2004. [Accessed 10 Jan 2010].
3. Bibel, W. Automated theorem proving. Vieweg Verlag, Wiesbaden, 1987.
4. Baader, F., Brandt, S., Lutz, C. Pushing the EL Envelope. LTCS-Report 05-01, Chair for Automata Theory, Inst. for Theoretical Computer Science, TU Dresden, Germany, 2005.
5. Ghilardi, S., Lutz, C., Wolter, F. Did I damage my ontology: A Case of Conservative Extensions of Description Logics. Proceedings of the Tenth International Conference of Principles of Knowledge Representation and Reasoning 2006 (KR'06), AAAI Press, 2006
6. Schlicht, A. Stuckenschmidt, H. Peer-to-peer Reasoning for Interlinked Ontologies. Int. Journal of Semantic Computing, Special Issue on Web Scale Reasoning, 2010
7. Freitas, F., Schlicht, A., Stuckenschmidt, H. A Connection Method for Reasoning with the Description Logic $\mathcal{ALC}$. Technical report. University of Mannheim. 2010. www.cin.ufpe.br/~fred/CM-ALCTechRep.doc [Accessed 10 Dec 2010].
8. Otten, J., Bibel, W. leanCoP: Lean Connection-Based Theorem Proving. Journal of Symbolic Computation, Volume 36, pages 139-161. Elsevier Science, 2003.
9. Schmidt, R., Tishkovsky, D. Analysis of Blocking Mechanisms for Description Logics. In Proceedings of the Workshop on Automated Reasoning, 2007.
10. Otten, J. Restricting backtracking in connection calculi. AI Comm., 23(2-3): 159-182 2010.
11. Moser, M., Ibens, O., Letz, R., Steinbach, J., Goller, C., Schumann, J., Mayr, K. SETHEO and e-SETHEO - the CADE-13 systems. J. of Automated Reasoning, 18(2):237{246, 1997.