# Towards Derivationless Inference for the Semantic Web: A Connection Method for $\mathcal{ALC}$

Fred Freitas[1,2], Anne Schlicht[2], Heiner Stuckenschmidt[2]

[1]Informatics Center - Federal Universidade of Pernambuco (CIn - UFPE)
Av. Prof. Luis Freire, s/n, Cidade Universitária, 50740-540, Recife – PE, Brazil
[2]Computer Science Institute, B6, 26, 68159, University of Mannheim, Germany

fred@cin.ufpe.br, {anne, heiner}@informatik.uni-mannheim.de

**Abstract.** The connection method earned good reputation in the field of automated theorem proving for around three decades, due to its simplicity, clarity and parcimonious use of memory. It employs easy to understand and implement matrices that deal with formulae and theorems directly, in opposition to the classical refutation methods like resolution and tableaux, which require formulae to be negated. Such distinctive features attracted our attention in order to propose a connection method especially tailored to infer over description logic (DL) Semantic Web ontologies. This paper is a first formal attempt in this direction. First, we present an ALC normal form which saves columns in the matrix, and then our $\mathcal{ALC}$ connection method is formalized in a sequent-style, although matrices should be employed instead of sequents for practical reasons. The inference system has some interesting features: it is able to perform all classical DL reasoning services and, as the most widespread DL tableaux systems, reduces them to subsumption or validity.

**Keywords**: connection method, description logic inference, reasoning, description logic classification, $\mathcal{ALC}$

## 1 Introduction

The problem of reasoning over ontologies written in Description Logic (DL) [1] has been receiving strong interest from researchers, particularly since the Semantic Web inception. The necessity of creating reasoners to deal with the specific fragments of the first-order logic (FOL) represented by the Description Logic in the OWL language [2] in its various versions posed interesting research questions for inference systems, making the problem earn the reputation of complex and mysterious, whose users see often solutions as black boxes. Therefore, not many reasoners became well known to the public; indeed, tableaux based methods [1] took over the field for years, and recently a system based on superposition calculus, Hermit [3], gained popularity.

Apart from the reasoning algorithms, many other practical issues that impact usability have brought about, stemming from the natural features of the Web. Scalability is probably one of the most relevant, since the volume of information to be dealt during an inference can be huge, what motivated the creation of specific scientific events devoted to works that handle billions of knowledge and data itens. Regarding this issue, the use of memory is certainly one important asset for a good reasoning performance. Therefore, methods that do not require a huge amount of memory may promise an advance to the field.

This paper is an attempt to tackle the two issues raised in the former paragraph. We are proposing a formalized inference system which seems adequate to address understandability and the use of a small amount of memory. The inference system proposed here is based on the connection calculus [4,5], which is a simple, clear and effective inference method that has been used successfully over first order logic (FOL). Its main features clearly meet with the demands: (i) it is a direct method, in opposition to the refutation approach that encompass most of the other approaches, like tableaux and resolution-based solutions; and (ii) it keeps only one copy of each logical sentence in memory; and (iii) it does not derive new sentences from the stored ones. A data structure composed of unifier and some control information on the proof search, together with the knowledge base, comprise the pieces of information to be manipulated by the method during an inference.

Our formal system, the $\mathcal{ALC}$ Connection Method (CM) has some interesting features: it is able to perform all inference classification services (namely, subsumption, unsatisfiability, equivalence and disjointness), and, as the most widespread DL tableaux systems, reduces all of them to subsumption or validity [1]. Some practical issues, such as notations, properties and possible optimizations are also discussed throughout the paper, in particular strategies for implementation and possible gains in terms of memory usage.

The rest of the article is organized as follows: section 2 defines first order formulae and the disjunctive normal formal to be used by the method; section 3 defines the syntax of the Description Logic $\mathcal{ALC}$, and brings the $\mathcal{ALC}$ ontologies' matricial normal form that fits to the connection method; section 4 describes the $\mathcal{ALC}$ adapted connection method, states the key logical properties for inference systems (soundness, completeness and termination) and explains how the approach deals with cyclical ontologies and blocking; section 5 discusses the notational improvements, reasoning advantages and future reductions, as relevant issues for implementation; and section 6 presents future work and conclusions.

## 2 Positive Matricial Normal form

In order to describe the connection method as a formal inference system (in section 4), and the positive matricial normal formal used in it, we will briefly describe the notation we use for first order logic, before examining the method. We are presuming readers to be acquainted to first order logic.

**Definition 1 (First-order logic syntax).** The following alphabets compose FOL syntax notation: predicate symbols (denoted by $P, Q, R, S$), variables (denoted by $x,y, z$), constants (denoted by $a, b, c$), and function symbols (denoted by $f, g$). Terms (denoted by $t_1, \ldots, t_n$) are composed from the other alphabets. *Atoms or atomic formulae* are predicates in the form $P(t_1, \ldots, t_n)$. A *first-order formula* is a set of atoms linked by connectives, whose variables are existentially or universally quantified (using the symbols $\exists$ and $\forall$ respectively). The connectives used are: $\neg$ (unary negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication), all of them binary, except negation. A *literal* is either an atom or its negation, and is denoted by $L$. The complement of a literal $L$ is its negation $\neg L$ and vice-versa.

**Definition 2 (Disjunctive normal form (DNF), clause, positive matricial form).** A *formula in DNF* is a disjunction of conjunctions, being in the form $C_1 \vee \ldots \vee C_n$, where each $C_i$ is a *clause.* Clauses are conjunctions of literals like $L_1 \wedge \ldots \wedge L_m$, also denoted as $\{L_1, \ldots, L_m\}$. Formulae can be also expressed in *disjunctive clausal form* as $\{C_1, \ldots, C_n\}$. Formulae stated this way are also in *positive matricial form,* since they can be represented as a matrix. In the matrix, each clause occupies a column.

**Definition 3 (Skolemization).** In order to deal with formulae in DNF, quantifiers should be removed by skolemization. In contrast to the common skolemization that preserves satisfiability, formulae transformed into DNF are valid iff the original formula is valid. Hence, instead of existential quantifiers, universal quantifiers ($\forall$) are replaced by constants or Skolem functions. Variables in the resulting DNF are then (implicitly) existentially quantified.

In the next section, we present one of the most basic description logic languages, $\mathcal{ALC}$, together with the transformation to positive matricial form. We started with the Description Logic $\mathcal{ALC}$ (Attributive Concept Language with Complements), because it constitutes the foundations of many other DLs.

## 3 The Description Logic $\mathcal{ALC}$

Description Logic (DL for short) is a family of knowledge representation formalisms that have been gaining strong interest from researchers in the last two decades, after the language OWL (Ontology Web Language) [2], based in DL, has been 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 ordered triple $(N_C, N_R, N_O)$ [1], where $N_C$ is the set of *concept names* or *atomic concepts* (unary predicates), $N_R$ is the set of *role or property names* (binary relations), and $N_O$ is the set of *individual names* (or *objects*), instances of $N_C$ and $N_R$.

$N_c$ is also known as Tbox and can contain $\top$ (the *top* concept, to which each instance $i \in N_O$ belongs to), $\bot$ (the *bottom concept or empty concept*), as well as other concept definitions as follows. If $r$ *is* a *role* ($r \in N_R$) and $C$ and $D$ are either *concepts* ($C, D \in N_C$) then the following are also *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 $C$ and $D$ can be inductively replaced by recursive complex concept expressions.

$N_O$ is composed of *concept assertions C(a),* where $C \in N_C$ , and *role assertions*

*R(a,b),* where $r \in N_R$. The axioms are composed of $N_O$ and any finite set of GCIs (*general concept inclusion*) in one of the forms $C \sqsubseteq D$ or $C \equiv D$, the latter one meaning both $C \sqsubseteq D$ and $D \sqsubseteq C$, where $C$ and $D$ are *concepts.* An ontology or knowledge base (*KB*) is also referred as a pair $(\mathcal{T},\mathcal{A})$, where $\mathcal{T}$ is a terminological box (or Tbox), which stores all but $N_O$. $N_O$ is stored in the assertional box (ABox) $\mathcal{A}$.

There are two major ways of defining $\mathcal{ALC}$ semantics. The most used is relying on the definitions of interpretation, model, fixpoints, etc, over a domain $\Delta$, [1]. Another (and easiest) way is mapping $\mathcal{ALC}$ constructs to first order logic (FOL) and benefitting from the semantics already defined for FOL, for instance in [4], III.2. We chose the latter, with the purpose of inheriting FOL semantics to take advantage of the formalized completeness and soundness theorems for our work.

In the next section we present the $\mathcal{ALC}$ disjunctive normal form used in the system.

## 3.1. An $\mathcal{ALC}$ Positive Matricial Normal Form

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 subsumptiom axioms $C \sqsubseteq D$ and $D \sqsubseteq C$ and, (ii) converting all the axioms into a *Negation Normal Form*, in which negations occurs only on literals [6,7]. Next, we define the normal form, preceded by some necessary definitions.

**Definition 4 ($\mathcal{ALC}$ 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 5 ($\mathcal{ALC}$ pure disjunction).** An $\mathcal{ALC}$ *pure disjunction* is an $\mathcal{ALC}$ disjunction, that in negation normal form, contains no existential quantifier ($\exists$) and no conjunctions ($\sqcap$). The set of $\mathcal{ALC}$ *pure disjunctions* is denoted by $S_D$, Elements $\check{D} \in S_D$ are marked by a circumflex for readability. An $\mathcal{ALC}$ *non-pure disjunction* is an $\mathcal{ALC}$ disjunction that is not a pure disjunction.

**Definition 6 ($\mathcal{ALC}$ pure conjunction).** Analogously, an $\mathcal{ALC}$ *pure conjunction* is an $\mathcal{ALC}$ conjunction that in negation normal form, contains no universal quantifier ($\forall$) and no disjunctions ($\sqcup$). The set of $\mathcal{ALC}$ *pure conjunctions* is denoted by $S_C$, Elements $\hat{C} \in S_D$ are marked by a caron for readability. An $\mathcal{ALC}$ *non-pure conjunction* is an $\mathcal{ALC}$ conjunction that is not an $\mathcal{ALC}$ pure conjunction.

**Definition 7 (Impurity on a non-pure expression).** During a lazy evaluation parsing [8], impurities of 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 $(\forall r.(D_0 \sqcup \ldots \sqcup D_n \sqcup (C_0 \sqcap \ldots \sqcap C_m) \sqcup (A_0 \sqcap \ldots \sqcap A_p))$, a non-pure disjunction, contains two impurities: $(C_0 \sqcap \ldots \sqcap C_m)$ and $(A_0 \sqcap \ldots \sqcap A_p)$.

On the other hand, the expression $\exists r.(\forall r'.((D_0 \sqcup \ldots \sqcup D_m) \sqcup (A_0 \sqcap \ldots \sqcap A_p)))$, a non-pure conjunction, has one impurity $\forall r'.((D_0 \sqcup \ldots \sqcup D_m) \sqcup (A_0 \sqcap \ldots \sqcap A_p))$.

**Definition 8 (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.

The aim of these normal forms' definition is to assure creating only the necessary columns in a matrix representation, at the expense of longer ones. One important remark is that the normalized resulting TBox is a *conservative extension* [9] of the original TBox, since every model of the former is a model of the latter too [6,7].

## 3.2 Translation Rules for the normalization

With all the axioms in negated DNF, it is easy to map them to positive matricial form, by applying the rules given in Table 1. A representational improvement of the approach, as the usual DL notation, consists in the absence of variables, since all relations are binary.

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

| **Axiom type** | **FOL Positive DNNF mapping** | **Matrix** |
|---|---|---|
| $C \sqsubseteq \exists r.\hat{C}$, where $\hat{C} = \bigcap_{i=1}^{n} A_i$, with $A_i \in S_C$ (pure conjunction) | $(C(x) \land \neg r(x,f(x))) \lor$ $(C(x) \land \neg A_1(f(x)))$ v... v $(C(x) \land \neg A_n(f(x)))$ | $\begin{bmatrix} C & C & \cdots & C \\ \neg r & \neg A_1 & \cdots & \neg A_n \\ \hdashline \end{bmatrix}$ |
| $\forall r.\check{D} \sqsubseteq C$, where $\check{D} = \bigcup_{j=1}^{m} A'_j$, with $A'_j \in S_D$ (pure disjunction) | $(\neg r(x,f(x)) \land \neg C(x)) \lor$ $(\neg A'_1(f(x)) \land \neg C(x))$ v...v $(\neg A'_m(f(x)) \land \neg C(x))$ | $\begin{bmatrix} \neg r & A'_1 & \cdots & A'_m \\ \hdashline \neg C & \neg C & \cdots & \neg C \end{bmatrix}$ |
| $\hat{C} \sqsubseteq \check{D}$, where $\hat{C} = \bigcap_{i=1}^{n} A_i$, $\check{D} = \bigcup_{j=1}^{m} A'_j$, $A_i \in S_C$ (pure conjunction), $A'_j \in S_D$ (pure disjunction) | $A_1(x) \land \ldots \land A_n(x) \land$ $\neg A'_1(x) \land \ldots \land \neg A'_m(x)$ | $\begin{bmatrix} A_1 \\ \vdots \\ A_n \\ \neg A'_1 \\ \vdots \\ \neg A'_n \end{bmatrix}$ |

Besides the lack of variables, another new notational feature was introduced. Dashlines link roles to their qualified concepts in order to distinguish if the variable employed in the qualified concepts *C(y)* or *C(f(x))* from the ones used in other concepts (*x*, for examp*le*). There are two types of links: *horizontal* lines connect universally quantified roles to skolemized concepts ($\forall r.C$, designated by *r(x,f(x))* and *C(f(x))*), while *vertical* links *existentially* quantified roles ($\exists r.C$), to concepts (*C(y)*).

Table 2 brings the mapping treatment of recursive sub-cases of existential and universal restrictions, when they occur inside any of the three normal forms. Note, however, that position constraints apply here: existential restrictions occur only on the left side of axioms (like in $\exists r.C \sqsubseteq E$), while universal take place on the right side.

**Table 2.** Recursive sub-cases of existential and universal restrictions.

| **Axiom type** | **FOL Positive DNNF mapping** | **DNNF Positive Matrix** | **Direct Matrix** |
|---|---|---|---|
| $A_i$ is an existencial restriction: $\ldots \sqcap \exists r.A \sqcap \ldots$, with $A \in S_C$ (pure conjunction) | ... ∧ *r(x,y)*∧ *A(y)*∧ ... | $\begin{bmatrix} \vdots \\ r(x,y) \\ A(y) \\ \vdots \end{bmatrix}$ | $\begin{bmatrix} \vdots \\ r \\ A \\ \vdots \end{bmatrix}$ |
| $A'_j$ is an universal restriction: $\ldots \sqcup \forall r.A' \sqcup \ldots$, with $A' \in S_C$ (pure disjunction) | …∧ *r(x,y)* ∧ ¬*A'(y)* ∧ … | $\begin{bmatrix} \vdots \\ r(x,y) \\ \neg A'(y) \\ \vdots \end{bmatrix}$ | $\begin{bmatrix} \vdots \\ r \\ \neg A' \\ \vdots \end{bmatrix}$ |

The whole knowledge base *KB* should be negated during this transformation. This is justified as follows: once we wish to entail that $KB \vDash \alpha$ using a direct method (in opposition to the classical refutation methods like tableaux and resolution), we must prove $KB \rightarrow \alpha$, or, in other words, $\neg KB \vee \alpha$.

Regarding skolemization, one representational advantage of the approach resides in the clearer matrix representation of universally quantified roles' ($\forall r.C$ or in the matrices, the negated $\exists r.C$). This construct, by definition, has the interpretation $(\forall r.C)^I = \{\forall b, (a,b) \in R^I \rightarrow b \in C^I\}$. Hence, for an axiom of the form $A \equiv \forall r.C$, the definition does not oblige concept $A$ to have instances – indeed a very common error from DL users. Nonetheless, maybe it is not their fault: for instance, tableaux proofs over such axioms don't stress this semantics, in the sense that it allows instances of $A$ without any role instances from $r$ associated to it. In the $\mathcal{ALC}$ CM, the matricial representation explicits this situation: either there are no role instances ($\neg r$) or when it has a role instance (*a,b*), *b* has to be an instance of concept *C*.

Back to the normal form, the number of new introduced symbols (and also the number of axioms) grows linearly with the number of impurities. It is a substantial gain compared with other normalizations (from $\mathcal{EL}$ [6,7] and for distributed resolution [10]), whose number of new axioms grows linearly with the axioms' length.

In [11], algorithms for transforming $\mathcal{ALC}$ axioms to this normal form are available.

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

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

reads in FOL as $(\forall w(Animal(w) \land (\exists z\,(hasPart(w,z) \land Bone(z)) \rightarrow Vertebrate(w))) \land (\forall x(Bird(x) \rightarrow Animal(x) \land \exists y(hasPart(x,y) \land Bone(y)) \land \exists v(hasPart(x,v) \land Feather(v))) \rightarrow \forall t(Bird(t) \rightarrow Vertebrate(t)))$

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

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

$$\begin{bmatrix} Bird & Bird & Bird & Bird & Bird & Animal & \neg Bird(c) & Vertebrate(c) \\ \neg Animal & \neg hasPart & \neg Bone & \neg hasPart & \neg Feather & hasPart & & \\ & & & & & Bone & & \\ & & & & & \neg Vertebrate & & \end{bmatrix}$$

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

## 4 An $\mathcal{ALC}$ Connection Method

The connection method (CM) was created by W. Bibel in the 70s, and earned good reputation in the field of automated theorem proving for some decades. The reasons for this recognition are manifold:

- The method was the first easy-to understand and -implement method to deal with formulae and theorems directly (i.e., not by refutation) [4];
- It was well formalized and supportive material was extensively published (books, articles, etc, for instance, [4,5]), and are still under use nowadays in some universities to teach automated theorem proving;
- Any reductions, compressions, optimizations and improvements that fit to resolution or tableaux, can also be properly applied to it [5], 4.4.;
- Efficient implementations are available, including probably the smallest lean provers ever coded, e.g., leanCoP [12].

Such distinctive features attracted our attention in order to propose a connection method especially tailored to infer over description logic Semantic Web ontologies.

### 4.1 An $\mathcal{ALC}$ Connection Sequent and Matricial Calculus

**Definition 9 (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 $\neg L_2$. $\sigma$ 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 $\sigma$.

**Example 3 (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 $\sigma$*={x/c, w/c, z/f(c)}* is the whole matrix' mgu.

**Corollary 1 (Validity, active path, set of concepts).** An $\mathcal{ALC}$ formula represented as a matrix is *valid* when every path contains a connection $\{L_1, \neg L_2\}$, provided that $\sigma(L_1) = \sigma(\overline{L_2})$. 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* $\tau$ of a variable or instance $x$ during a proof is defined by $\tau(x) \overset{\text{def}}{=} \{C | C(x) \in \mathcal{B}\}$ [13,14].

**Definition 10 (ALC connection sequent calculus).** Figure 2 brings the rules in sequent style of the $\mathcal{ALC}$ connection calculus, adapted from [15].

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

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

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

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

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

**Figure 2.** The $\mathcal{ALC}$ connection sequent calculus rules (adapted from [15]).

The rules must be applied bottom-up. The matrix $M$ representing $KB \vDash \alpha$ is put in the bottom of the *Start rule*, and a clause $C_1 \in \alpha$ is chosen as the first clause to ensure that $\alpha$ takes part in the proof. Then, the three last rules are applied. The key rule for the calculus is the *Extension rule*, once it finds the connections of the form $\{E(x), \neg E(y)\}$, ($\{E, \neg E\}$ in the notation used here) or $\{r(x,y), \neg r(z,w)\}$and the proper unifier $\sigma$ at the same time. In the first case $\sigma$=$\{x/y\}$ and in the second $\sigma$=$\{x/z, y/w\}$. If an active path contains a connection and the set of literals $C$ is empty, the subgoal is proved, and the *Axiom rule* is applied.

Besides this slight change in the *Start rule*, the major differences from the original CM sequent calculus [15] are a new *Copy rule (Cop)*, together with a blocking mechanism. 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 $\mu: M \to \mathbb{N}$, that assigns the number of copies of each clause, is incremented. The use of this function avoids the ordering of individuals, as done in the original connection method.

Note that the existence of this rule was not mandatory, since the original system already disposes of a similar behavior. Nevertheless, here it serves the purposes of isolating the blocking from the *Ext* rule, as well as letting explicit the copy process. *Ext* precedes over *Cop*, so that clauses are copied only when no extension is possible.

Another important remark about the rule is that the copy is virtual, in the sense that only the index $\mu$ is incremented. The connection method requires only one copy of the matrix in memory, and this main advantage should not be neglected. Its use of memory can 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. In order to carry out that, we have to check whether the set of concepts $\tau$ associated to the variable $x_\mu^\sigma$ (i.e., if the new $x_\mu$ was unified) of the new literal $L_2^\mu$ being created by the *Cop* rule is not contained in the set of concepts of the original variable $x$ from $L_2(x)$ (in the rule, $\tau(x^\sigma)$) [13,14]. Examples of the $\mathcal{ALC}$ CM calculus' application, as well as an algorithm description of the system based on [4], III.7.2., can be found at [11].

**Theorem 1 (Correctness and completeness)** An $\mathcal{ALC}$ formula in positive matricial 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 changes in rules do not lose the correctness and completeness of the original CM [4], III.3.9 and 3.10, *viz*: (i) The original *Start rule* allows any clause to start the method; (ii) blocking does not affect correctness and completeness; it only guarantees termination instead. ■

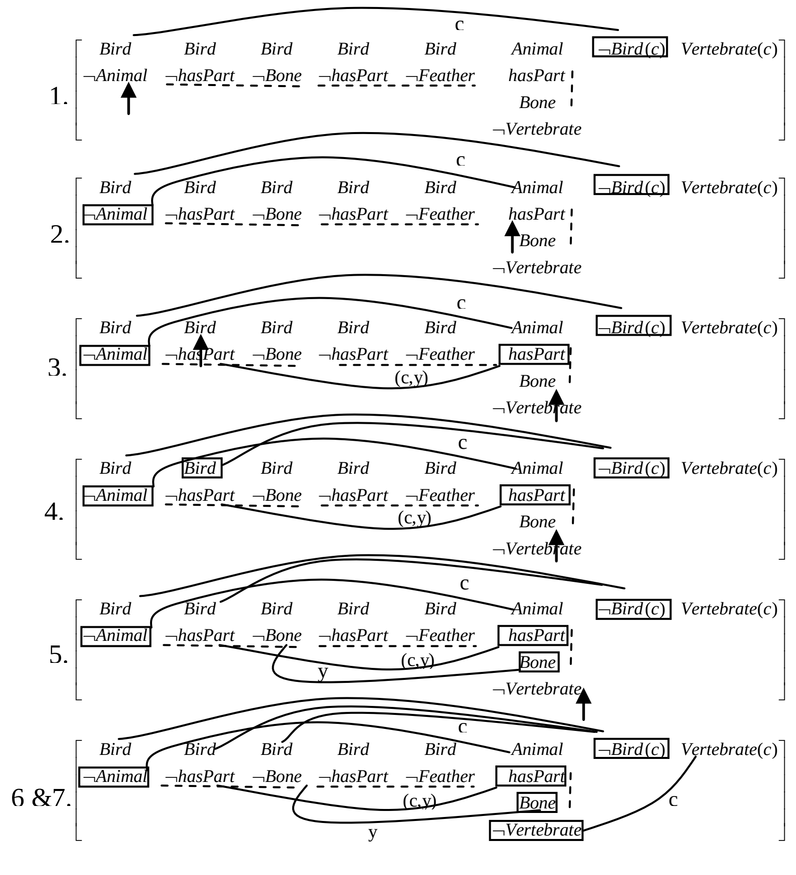
**Theorem 2 (Termination).** The $\mathcal{ALC}$ CM always terminates.

*Proof* If there are no cycles, then 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. If $KB \vDash \alpha$, even in the presence of cycles, the system always terminates (proof at [4]).

For $KB \nvDash \alpha$, with a cyclic $KB$, 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 by $2^{|N_C|}$. Therefore, the number of copies for any literal is finite and hence $\mathcal{ALC}$ CM always terminates. ■

The system can also be expressed in an easier way using matrices to build proofs.

**Example 5 ($\mathcal{ALC}$ connection calculus).** Figure 4 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, we first choose a clause from the consequent (*Start rule*), say, the clause {¬*Bird(c)*} and a literal from it (¬*Bird(c)*).



1.
2.
3.
4.
5.
6 &7.
Bird Bird Bird Bird Bird Animal ¬Bird(c) Vertebrate(c)
¬Animal ¬hasPart ¬Bone ¬hasPart ¬Feather hasPart
Bone
¬Vertebrate
c
(c,y)
y

**Figure 4.** A connection proof example in matricial form.

Step 1 connects this clause with the first matrix clause. An instance or variable - representing a fictitious individual that we are predicating about -, appears in each arc, for this connection, the instance *c*. The arrow points to literals still to be checked in the clause (¬*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 we connect *hasPart*, we are not talking about instance *c* any more, but about a relation between it and another variable or fictitious individual, say *y* (indicated by *(c,y)*).

Until that moment, we were only applying the *Extension rule.* However, in step 4, we use the *Reduction rule*, 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 if we were referring to different individuals or skolemized functions (in $\mathcal{ALC}$, equality among individuals is not necessary).

A small note on unification is necessary here, because it brings a small trick to the calculus. Since horizontal dashlines represent universal restrictions ($\forall r.C$), the qualifier concept ($C$, represented as $\neg C$ in the matrix) correspond to a skolemized concept (say *C(f(c))*). Therefore, it can only be unified with variables, but not with concrete individuals or other skolemized qualifier concepts.

Resuming to the example, a connection for the literal *Bone* was needed. That time we are referring to the variable (or fictitious individual) *y*, 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 the next section, we comment on the DL services $\mathcal{ALC}$ CM is able to perform.

## 4.2 Inference services

There are four standard inference services for any DL [1]: checking subsumption, unsatisfiability, equivalence, and disjointness. An interesting result of this work is that all these inference services are available for the $\mathcal{ALC}$ CM. As occurs with most widespread DL tableaux systems, these services are offered in our $\mathcal{ALC}$ CM either by reduction to subsumption and to validity (in opposition to the unsatisfiability reduction carried out by tableaux or by any other refutation proof method) [11]. The only change consists in the way unsatisfiability (or inconsistency) checking is done:

**Proposition 1. (Unsatisfiability)** Proving an arbitrary concept *C unsatisfiable* in any direct proof method is equivalent to proving $\mathcal{T} \vDash \top \sqsubseteq \neg C$.

*Proof* Proving a class inconsistent stands for proving $\mathcal{T} \vDash C \sqsubseteq \bot$. Since $A \sqsubseteq B$ is equivalent to $\neg B \sqsubseteq \neg A$, then $\mathcal{T} \vDash C \sqsubseteq \bot$ can be proved by $\mathcal{T} \vDash \top \sqsubseteq \neg C$. ■

**Example 5. (Unsatisfiability).** If we consider concepts Man and Woman disjoint (or vice-versa) the axiom Human ⊑ Man ⊓ Woman is inconsistent, as could be seen in the matrix of figure. Note that the matrix is $\mathcal{T}$ with a last column for ¬Human.

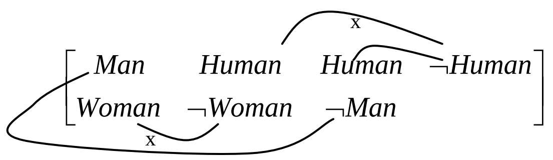


**Figure 5.** Inconsistency proof of the concept Human, if Human ⊑ Man ⊓ Woman.

Having discussed how the system works, in the next section we describe how it deals with cycles and blocking.

## 4.3 Cycles, blocking

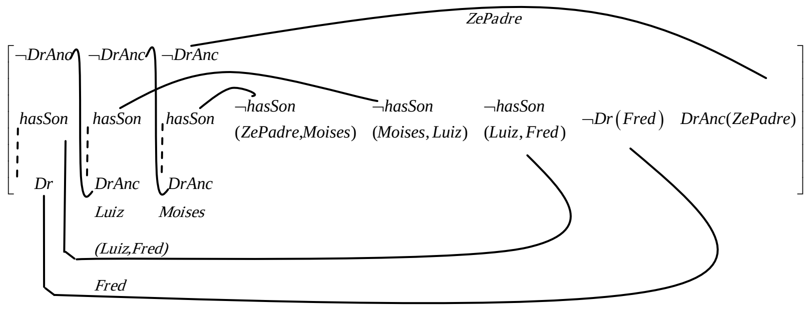
A relevant ability for a DL reasoner is dealing with cyclic ontologies. $\mathcal{ALC}$ CM not only can cope with the task (thanks to the *Copy rule*), but also saves memory during the process, due to needing only one copy of the matrix in memory [4,5].

During the proof search of an assertional query or sub-query (i.e., one which refers to assertions), connections to instances of the ABox should be privileged, rather than to variables, otherwise blocking could happen when connections are still available. An example on cycling shows how $\mathcal{ALC}$ CM works in this case.

**Example 6 (Cycles).** If the TBox ∃hasSon.(Dr ⊔ DrAncestor) ⊑ DrAncestor receives the query KB ⊨ DrAncestor(ZePadre), under the ABox:

hasSon (ZePadre,Moises), hasSon (Moises,Luiz), hasSon (Luiz,Fred), Dr (Fred), then the proof is represented by Figures 6a and 6b.

a)



b)

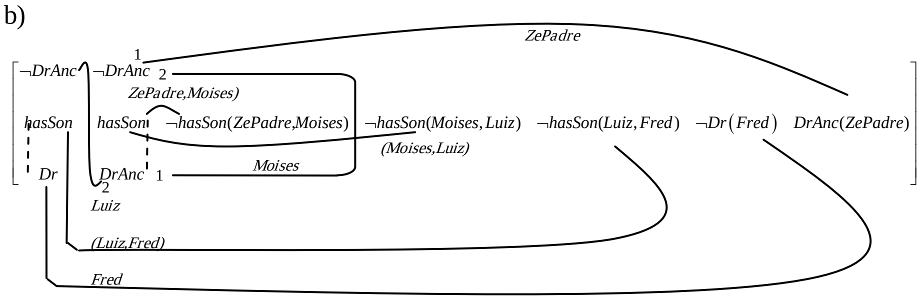


**Figure 6.** Proof representations of a cyclic axiom, with (a) explicit and (b) implicit copies.

Figure 6a brings an explicit copy of a clause which participates in the proof twice. This presentation is surely more user-friendly than the one from Figure 6b. Figure 6b uses indices to denote how the only copy was used with the different individuals and instantiations. Let us explain this last figure in detail.

First, a connection with the consequent (DrAncestor (ZePadre)) was settled. The connected clause was unified with instance ZePadre, generating two literals to be connected: hasSon (ZePadre, x) and DrAncestor(x). The first was connected to hasSon (ZePadre, Moises), and since σ(x) = Moises, then DrAncestor (Moises) needs to be proved. After that, instead of copying the clause, a connection to a literal in the same column was set. The index representing $\mu$ to denote virtual copies was incremented when this connection was created; that is indicated by the numbers 1 and 2 in the extremes of the connection. This practice already exists in the CM, under the name *implicit amplification* [16]; we adopt it here with exactly the same notation.

The last connection generated literals hasSon (Moises, y) and DrAncestor (y), which were unified with ABox instances, with σ(y) = Luiz. Next, already with $\mu$=2, DrAncestor (Luiz), was connected to the first column. The remaining literals ¬hasSon (Luiz, z) and Dr(z) were then connected to instances hasSon (Luiz, Fred) and Dr(Fred), closing all possible paths and consequently proving the query.

Next, we look into the functioning of blocking, since it is based on copies.

**Example 7 (Blocking).** Suppose the last example contained no ABox. The system does not summon the query, as can be seen in Figure 7.

At first, the connection {DrAncestor (ZePadre),¬DrAncestor(z)} was found with σ(z) = ZePadre. Next, we had literals hasSon(ZePadre, x) and DrAncestor(x) yet to be proved. Assuming that the variable x plays the role of a fictitious individual, then we had to prove that, if x exists, then DrAncestor(x) is also true. Thus, we set $\mu$ to 2 and made a cyclical connection in the same clause, so leaving the literals hasSon(x,y) and DrAncestor(y) yet to be proved.

Next, the same connection is repeated, creating another virtual copy of the clause, with $\mu$ set to 3. However, y could not be accepted as a fictitious individual this time, once the blocking conditions were met: (i) y is a variable, not a concrete instance (y $\notin N_O$); (ii) $\tau$(y) ⊆ $\tau$(x) (both equal to {DrAncestor}), and (ii) $\mu(C_y) > \mu(C_x)$. Thus, y was blocked and the query was not entailed.

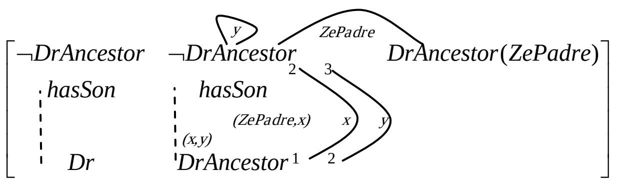


**Figure 7.** A blocking representation example.

Therefore, during proof search for an assertional query (i.e., one which refers to assertions) or sub-query, connections to the ABox must be privileged, rather than to variables. This is justified by the possible presence of blocking as shown.

In the next section, we discuss $\mathcal{ALC}$ CM in the light of knowledge representation, potential advantages on reasoning and implementation details to be incorporated.

## 5 Discussion

While planning the $\mathcal{ALC}$ CM, i.e. adapting the original CM to the DL $\mathcal{ALC}$, we designed its notation trying to avoid any ambiguities to arise from the fact that variables do not take part of the representation – that were the reasons for the dashlines replacing the quantifiers. We plan to the change this notation even further, since atomic assertions are better stored outside the matrices, causing at least two benefits: matrix size can be dramatically shrunk and the possibility of relying on powerful indexing mechanisms for retrieving assertions, such as relational databases. The use of deductive databases can bring additional profits for reducing the reasoning strain on the $\mathcal{ALC}$ CM. However, even keeping only TBoxes, matrices can be sparse, causing a memory waste. A simple solution is storing matrices as arrays of arrays.

Still concerning representation, although we have proposed a normal form which includes less introduced artificial concepts, thus being more efficient than others [6, 7, 10], an even compacter normal form is possible, if sub-matrices come into play. Y doing so, many columns could be saved, as portrayed in Figure 8. In this example, the matrix have four less predicates than the original one (from Example 2). Moreover, such representation would require less reasoning overhead for the backtracking, since one connection on a shared column part serves many clauses.

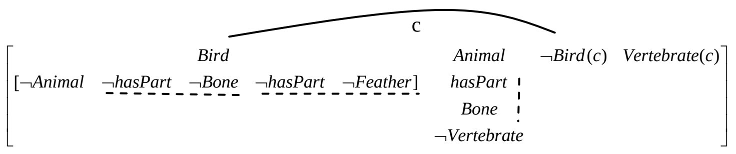


**Figure 8.** An $\mathcal{ALC}$ formula and query represented as a matrix, with a shared connection.

As for the reasoning, the first fact to be taken into account is the strong similarity between CM and tableaux [5]. Therefore, reductions and optimizations designed over the latter are very likely to be incorporated to the former. Additionally, a performance at least comparable to DL tableaux is expected, given the fact that CM itself usually displays a good performance (it already won a CASC competition once [17]). Of course, a deeper investigation on the cases in which could be advantageous to rely on tableaux or the CM is on our research agenda.

Another aspect regarding reasoning resides in $\mathcal{ALC}$ CM specific reductions. For instance, to infer over two equivalence axioms, that usually occupy many columns, an $\mathcal{ALC}$ CM reasoner will need only one of the two subsumption axioms of each, thus reducing both matrix size and proof search space.

## 6 Conclusions and Future Work

We have formalized a connection method to take on the DL $\mathcal{ALC}$, by adapting the CM sequent rules from [15] and introducing some notational improvements, the key one. being the representation without variables. Of course, we plan to continue this work in many research directions, such as implementations, other DLs, Semantic Web, etc.

First, we intend to extend the work presented here to more complex description logic languages in a near future. Particularly, formalizations and implementations for the DLs $\mathcal{EL}$, $\mathcal{EL}$++, $\mathcal{SHIQ}$ and $\mathcal{SROIQ}$ will be practically useful for Semantic Web and for some other biomedical applications that we are involved in.

Last but not least, lean implementations written in Prolog, in the flavor of leanCop [12], that demand small memory space, can serve applications that are constrained in memory, such as stream reasoning in mobile applications, for instance. They are also in our research agenda.

## Acknowledgments

The first author thanks the Brazilian research agency "Fundação Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Capes)" for the partial support of the work, under Project number 317/09.

## 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. Motik, B., Shearer, R., and Horrocks, I. Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research, 36:165–228, 2009.
4. Bibel, W. Automated theorem proving. Vieweg Verlag, Wiesbaden, 1987.
5. Bibel, W. Deduction: Automated Logic. Academic Press, London, 1993.
6. Baader, F., Brandt, S., Lutz, C. Pushing the EL Envelope. In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05, Edinburgh, UK, 2005.
7. 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.
8. Watt, D. Programming Language Concepts and Paradigms. Prentice Hall. 1993.
9. 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
10. Schlicht, A. Stuckenschmidt, H. Peer-to-peer Reasoning for Interlinked Ontologies. Int. Journal of Semantic Computing, Special Issue on Web Scale Reasoning, 2010
11. Freitas, F., Schlicht, A., Stuckenschmidt, H. A Connection Method for Reasoning with the Description Logic *ALC*. Technical report. University of Mannheim. 2010. www.cin.ufpe.br/~fred/CM-ALCTechRep.doc [Accessed 10 Dec 2010].
12. Otten, J., Bibel, W. leanCoP: Lean Connection-Based Theorem Proving. Journal of Symbolic Computation, Volume 36, pages 139-161. Elsevier Science, 2003.Schmidt,
13. R., Tishkovsky, D. Analysis of Blocking Mechanisms for Description Logics. In Proceedings of the Workshop on Automated Reasoning, 2007.
14. Tishkovsky, D. Blocking Mechanisms in Description Logics, A General Approach. www.uivt.cas.cz/semweb/download.php?file=08-Tishkovsky-Logic-Blocking&type=pdf Prague, 2008. [Accessed 10 Dec 2010].
15. Otten, J. Restricting backtracking in connection calculi. AI Comm., 23(2-3): 159-182 2010.
16. Bibel, W. Matings in Matrices. Communications of the ACM. Vol. 26 Issue 11, Nov. 1983.
17. 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.