Converting $\mathcal{ALC}$ Connection Proofs into $\mathcal{ALC}$ Sequents

Eunice Palmeira  
Federal Institute of Alagoas  
Maceió - AL, Brazil  
eunicepalmeira@ifal.edu.br

Fred Freitas  
Federal University of Pernambuco  
Recife - PE, Brazil  
fred@cin.ufpe.br

Jens Otten  
University of Oslo  
Oslo, Norway  
jeotten@ifi.uio.no

The connection method has earned good reputation in the area of automated theorem proving, due to its simplicity, efficiency and rational use of memory. This method has been applied recently in automatic provers that reason over ontologies written in the description logic $\mathcal{ALC}$. However, proofs generated by connection calculi are difficult to understand. Proof readability is largely lost by the transformations to disjunctive normal form applied over the formulae to be proven. Such a proof model, albeit efficient, prevents inference systems based on it from effectively providing justifications and/or descriptions of the steps used in inferences. To address this problem, in this paper we propose a method for converting matricial proofs generated by the $\mathcal{ALC}$ connection method to $\mathcal{ALC}$ sequent proofs, which are much easier to understand, and whose translation to natural language is more straightforward. We also describe a calculus that accepts the input formula in a non-clausal $\mathcal{ALC}$ format, what simplifies the translation.

1 Introduction

Description Logics (DLs) are a family of knowledge representation formalisms considered as a fundamental foundation for the Semantic Web, as it constitutes the formalism underlying the Web Ontology Language (OWL) language. DL is an expressive, decidable subset of First Order Logic (FOL), successfully applied in several areas. DL provides a precise and unambiguous meaning to DL descriptions due to its formal semantics, and fast reasoners have been produced to the many fragments available.

One of them, the $\mathcal{ALC}$ $\theta$-Connections Calculus, and its automated reasoner RACCOON (Reasoner based on the Connection Calculus Over ONtologies), is based on the Connection Method [5, 8], and was specifically developed to infer over the Description Logic $\mathcal{ALC}$ [5, 8]. The calculus includes typical DL features and techniques, such as notation without variables, absence of Skolem functions/unification and, inclusion of a blocking rule to handle cycles, which guarantees termination to make for the case of cyclic ontologies. The Connection Calculus has earned good reputation in the area of automated theorem proving due to its simplicity, efficiency and rational use of memory. The method represents formulae as matrices, whose columns are conjunctive clauses; its proof procedure consists of horizontally traversing paths through the matrix in order to connect complimentary literals (e.g., $L$ with its complement $\neg L$). A pair $\{L, \neg L\}$, is called a connection, which corresponds to the validity the path being checked. Thus, a formula is valid if every path through the matrix corresponding to it has a connection.

Both calculi mentioned above, before attempting to find a proof, convert a formula into a disjunctive normal form. The translation to this clausal form often obscures the structure of the original formula and transforms some simple theorem proofs into difficult ones [11]. In complex cases, the deductions' premise(s) and conclusion can no longer be clearly identified, once the transformation has been applied [3]. Thus, proof readability and understandability is largely lost, and consequently, it becomes quite difficult to provide justifications and/or descriptions of the steps used during inferences.

The $\theta$-Non-clausal $\mathcal{ALC}$ $\theta$-Connection Calculus is based on the $\mathcal{ALC}$ $\theta$-Connection Calculus and works directly on the structure of the original formula, thus avoiding the translation into a clausal form.

11 Description Logics (DLs) are a family of knowledge representation formalisms considered as a fundamental foundation for the Semantic Web, as it constitutes the formalism underlying the Web Ontology Language (OWL) language. DL is an expressive, decidable subset of First Order Logic (FOL), successfully applied in several areas. DL provides a precise and unambiguous meaning to DL descriptions due to its formal semantics, and fast reasoners have been produced to the many fragments available [7].

One of them, the $\mathcal{ALC}$ $\theta$-Connections Calculus, and its automated reasoner RACCOON (Reasoner based on the Connection Calculus Over ONtologies), is based on the Connection Method [5, 8], and was specifically developed to infer over the Description Logic $\mathcal{ALC}$ [5, 8]. The calculus includes typical DL features and techniques, such as notation without variables, absence of Skolem functions/unification and, inclusion of a blocking rule to handle cycles, which guarantees termination to make for the case of cyclic ontologies. The Connection Calculus has earned good reputation in the area of automated theorem proving due to its simplicity, efficiency and rational use of memory. The method represents formulae as matrices, whose columns are conjunctive clauses; its proof procedure consists of horizontally traversing paths through the matrix in order to connect complimentary literals (e.g., $L$ with its complement $\neg L$). A pair $\{L, \neg L\}$, is called a connection, which corresponds to the validity the path being checked. Thus, a formula is valid if every path through the matrix corresponding to it has a connection.

Both calculi mentioned above, before attempting to find a proof, convert a formula into a disjunctive normal form. The translation to this clausal form often obscures the structure of the original formula and transforms some simple theorem proofs into difficult ones [11]. In complex cases, the deductions’ premise(s) and conclusion can no longer be clearly identified, once the transformation has been applied [3]. Thus, proof readability and understandability is largely lost, and consequently, it becomes quite difficult to provide justifications and/or descriptions of the steps used during inferences.

The $\theta$-Non-clausal $\mathcal{ALC}$ $\theta$-Connection Calculus is based on the $\mathcal{ALC}$ $\theta$-Connection Calculus and works directly on the structure of the original formula, thus avoiding the translation into a clausal form.
Nevertheless, its proof format is still not intuitive, once, like other connection calculi, it consists of a set of complementary pairs found in each path through the matrix, when the formula is valid.

The motivation of this work is to make a connection proof for ALC more readable so that, in a near future, justifications can be generated automatically in natural language. Therefore, this article proposes a conversion method that translates non-clausal ALC $\theta$-connection proofs into ALC sequent proofs. Sequent calculi have a more friendly proof representation than connection calculus; it conveys proofs in a formal logic argument style, where each proof line is a conditional tautology. Such translation should therefore contribute to a better user interaction with DL reasoners based on the Connection Method.

The DL ALC is presented in the next section; Section 3 brings an ALC non-clausal Connection Calculus for ALC; Section 4 introduces the ALC Sequent Calculus, to which proofs will be translated; the conversion process and its main concepts in Section 5; an overview of the main algorithms for the conversion method with its computational complexities in Section 6; and conclusions in Section 7.

2 The Description Logic ALC

An ontology $O$ in ALC is a set of axioms over a signature $(N_C, N_R, N_O)$, where $N_C$ is the set of concept names (unary predicate symbols), $N_R$ is the set of role or property names (binary predicate symbols); $N_O$ is the set of individual names (constants) \[1\]. Concept expressions are inductively defined as follows. $N_C$ includes $\top$, the universal concept that subsumes all concepts, and $\bot$, the bottom concept subsumed by all concept names belong to $N_C$. If $r \in N_R$ is a role and $C, D \in N_C$ are concepts, then the following formulae are also concepts: (i) $C \cap D$, (ii) $C \cup D$, (iii) $\neg C$, (iv) $\forall r.C$, (v) $\exists r.C$.

A knowledge base in DL consists of a set of basic axioms (TBox), and a set of axioms specific to a particular situation (ABox). Two axiom types are allowed in a TBox $\mathcal{T}$: (i) $C \subseteq D$; (ii) $C \equiv D$, standing for $C \subseteq D$ and $D \subseteq C$. An ABox $\mathcal{A}$ w.r.t. a TBox $\mathcal{T}$ is a finite set of assertions of two types: (i) a concept assertion is a statement of the form $C(a)$, where $a \in N_O$, $C \in N_C$ and (ii) a role assertion $r(a,b)$, where $a, b \in N_O$, $r \in N_R$. An ALC formula is either an axiom or an assertion; an ontology $O$ is an ordered pair $(\mathcal{T}, \mathcal{A})$. The semantics of concepts and ontologies is defined in the usual way - see, e.g., \[1\].

3 The Non-clausal ALC $\theta$-Connection Calculus

Definition 1. (Query). A query $O \models \alpha$ is an ALC formula to be proven valid, where $O$ is an ALC ontology, and $\alpha$ is either a TBox or an ABox axiom to be proven a logical consequence from $O$.

Definition 2. (Literal, clause, matrix). ALC Literals are atomic concepts or roles, possibly negated or instantiated in the form $L$ or $\neg L$. An ALC disjunction is either a literal $L$, a disjunction $(E_0 \sqcup E_1)$ or an universal restriction $\forall r.E_0$. An ALC conjunction is either a literal $L$, a conjunction $(E_0 \sqcap E_1)$ or an existential restriction $\exists r.E_0$, where $E_0$ and $E_1$ are expressions of arbitrary concepts (see DLs and its Mapping to FOL in \[2\]). Clauses are conjunctions of literals and matrices in the form $L_1 \sqcap \ldots \sqcap L_m$, where each $L_i$ is a literal or a matrix. A matrix of a formula (in DNF) is its representation as a set $\{C_1, \ldots, C_n\}$, where each $C_i$ is a clause.

Definition 3. (Formula with polarity). A formula with polarity, denoted by $F^p$, consists of a formula $F$ and a polarity $p$, where $p \in \{0,1\}$, that is, 0 is positive and 1 is negative. This concept is used to denote negation in a matrix, i.e., literals or matrices $A$ and $\neg A$ are represented by $A^0$ and $A^1$, respectively.

Definition 4. (ALC Non-Clausal Matrix). An ALC non-clausal matrix is a set of clauses in which a clause is a set of literals and matrices. Let $F$ be a formula and $p$ be a polarity. The matrix of $F^p$, denoted
by \(M(F^p)\), is inductively defined according to Table \([7]\) which indicates how the polarity is inherited by the (sub-)matrices of an \(F^p\). The matrix of \(F^p\) is the matrix \(M(F^0)\). Literals or (sub-)matrices involved in a universal restriction (\(\forall r.C\)) or in an existential restriction (\(\exists r.C\)) are underlined in the matrix.

Table 1: Matrix of an \(\mathbb{ALC}\) formula \(F^p\).

| Type   | \(F^p\) | \(M(F^p)\) |
|--------|---------|------------|
| Atomic | \(A^0\) | \([A^0]\)   |
|        | \(A^1\) | \([A^1]\)   |
| \(\alpha\) | \(\neg C\) | \(M(C^1)\) |
|        | \(C \cap D\) | \([M(C^1)], [M(D^1)]\) |
|        | \(C \cup D\) | \([M(C^0)], [M(D^0)]\) |
|        | \(C \subseteq D\) | \([M(C^1)], [M(D^0)]\) |
|        | \(C \models D\) | \([M(C^1)], [M(D^0)]\) |

| Type   | \(F^p\) | \(M(F^p)\) |
|--------|---------|------------|
| \(\beta\) | \((C \cap D)^0\) | \([M(C^0)], [M(D^0)]\) |
|        | \((C \cup D)^1\) | \([M(C^1)], [M(D^1)]\) |
| \(\gamma\) | \((\forall r.D)^1\) | \([M(\gamma^1)], [M(D^0)]\) |
|        | \((\exists r.D)^0\) | \([M(\gamma^0)], [M(D^0)]\) |
| \(\delta\) | \((\forall r.D)^0\) | \([M(\gamma^0)], [M(D^0)]\) |
|        | \((\exists r.D)^1\) | \([M(\gamma^0)], [M(D^0)]\) |

Definition 5. (Positive) Graphical Representation of the Matrix). In the (positive) graphical representation of a matrix, its clauses are arranged horizontally, while the literals and (sub-)matrices of each clause are arranged vertically. The restrictions are represented by solid lines; when a restriction involves more than one clause, its literals are indexed in the bottom with the same index in the matrix column in the written representation, for example, the notation \(L_i\) (see example \([7]\)); restrictions with indexes are represented with horizontal lines; restrictions without indexes with vertical lines.

Example 1. (Query, clause, \(\mathbb{ALC}\) non-clausal matrix, formula with polarity, graphical representation of a matrix). The query \(F_1 = [\exists hasPet.Cat \subseteq CatOwner, OldLady \subseteq \exists hasPet.Animal \land \forall hasPet.Cat] \models OldLady \subseteq CatOwner\) is read in FOL as:

\[
\forall x((\exists y hasPet(x,y) \land Cat(y)) \rightarrow CatOwner(x)) \\
\forall z(\text{OldLady}(z) \rightarrow \exists v(hasPet(z,v) \land Animal(v))) \\
\land \forall k(hasPet(z,k) \rightarrow Cat(k))
\]

and is represented by the FOL matrix (\(a\) is a Skolem terms, \(f\) a function symbol):

\[
[\{\text{hasPet}(x,y), \text{Cat}(y), \neg \text{CatOwner}(x), \text{OldLady}(z), \{\neg \text{hasPet}(z,f(z)), \neg \text{Animal}(f(z)), \text{hasPet}(w,k), \neg \text{Cat}(k))\}, \{\neg \text{OldLady}(a), \text{CatOwner}(a)\}]
\]

and by the following \(\mathbb{ALC}\) non-clausal matrix \(M_1\), which is defined according to \([1]\)(column indices relate the two clauses involved in a same restriction; variables are omitted as they are specified implicitly):

\[
[\{\text{hasPet}^0, \text{Cat}^0, \text{CatOwner}^1\}, \text{OldLady}^0, \{\text{hasPet}^1, \text{Animal}^1, \text{hasPet}^0, \text{Cat}^1\}], \\
\{\text{OldLady}^0, \{\text{CatOwner}^0\}\}]
\]

So, the graphical representation of \(M_1\) is:

\[
\begin{bmatrix}
\text{hasPet}^0 & \text{Cat}^0 & \text{CatOwner}^1 \\
\text{Cat}^0 & \text{hasPet}^0 & \text{Cat}^1 \\
\end{bmatrix}
\begin{bmatrix}
\text{OldLady}^0 \\
\text{OldLady}^0 \\
\end{bmatrix}
\begin{bmatrix}
\text{hasPet}^0 \\
\text{Animal}^1 \\
\end{bmatrix}
\begin{bmatrix}
\text{Cat}^0 \\
\text{Cat}^1 \\
\end{bmatrix}
\begin{bmatrix}
\text{CatOwner}^0 \\
\text{CatOwner}^0 \\
\end{bmatrix}
\]

Matrices of the form \(M = \{\ldots, C_1, \ldots, C_n, \ldots\}\) can be simplified to \(M' = \{\ldots, C_1, \ldots, C_n, \ldots\}\), where \(C_1, \ldots, C_n\) are clauses.

Clauses of the form \(C = \{\ldots, M_1, \ldots, M_m, \ldots\}\) can be simplified to \(C' = \{\ldots, M_1, \ldots, M_m, \ldots\}\), where \(M_1, \ldots, M_m\) are matrices.
Definition 6. (Path). A path through a matrix \( M = \{C_1, \ldots, C_n\} \) is a set of literals containing a literal \( L_i \) of each clause \( C_i \in M, \) i.e., \( \cup_{i=1}^{n} \{L_i\} \) with \( L_i \subseteq C_i \). A path through a matrix \( M \) (or a clause \( C \)) is inductively defined as follows. The (only) path through a literal \( L \) is \( \{L\} \). If \( p_1, \ldots, p_n \) are paths through the clauses \( C_1, \ldots, C_n \) respectively, then \( p_1 \cup \ldots \cup p_n \) is a path through the matrix \( M = \{C_1, \ldots, C_n\} \). If \( p_1, \ldots, p_n \) are paths through the matrices/literals \( M_1, \ldots, M_n \) respectively, then \( p_1, \ldots, p_n \) are also paths through the clause \( C = \{M_1, \ldots, M_n\} \).

Definition 7. (Connection, \( \theta \)-substitution, \( \theta \)-complementary connection). A connection is a pair of literals \( \{E, \neg E\} \) with the same concept/role name, but different polarities. A \( \theta \)-substitution assigns to each (possibly omitted) variable an individual or another variable (in the whole matrix). A \( \theta \)-complementary connection is a pair of \( \mathcal{ALC} \) literals \( \{E(x), \neg E(y)\} \) or \( \{p(x, v), \neg p(y, u)\} \), with \( \theta(x) = \theta(y), \theta(v) = \theta(u) \). The complement \( \overline{L} \) of a literal \( L \) is \( E \) if \( L = \neg E \), and it is \( \neg E \) if \( L = E \).

Simple term unification without Skolem functions is used to calculate \( \theta \)-substitutions. The application of a \( \theta \)-substitution to a literal is an application to its variables, i.e. \( \theta(E) = E(\theta(x)) \) and \( \theta(r) = r(\theta(x), \theta(y)) \), where \( E \) is an atomic concept and \( r \) is a role. Furthermore, \( \alpha^0 = \theta(x). \)

Example 2. (Path, Connection, \( \theta \)-substitution, \( \theta \)-complementary connection). In the matrix \( M_1 \) of Example 1 \( \{\text{hasPet}_0^1, \text{hasPet}_1^1, \text{Animal}_1^1, \text{hasPet}_0^1, \text{OldLady}_0^1, \text{CatOwner}_0^1\} \) and \( \{\text{Cat}_0^1, \text{hasPet}_1^1, \text{Animal}_1^1, \text{Cat}_1^1, \text{OldLady}_1^1, \text{CatOwner}_0^1\} \) are some paths through \( M_1 \), \( \{\text{Cat}_0^1, \text{Cat}_1^1\} \) is a connection. \( \theta(\text{OldLady}_0^1) = \text{OldLady}(\theta(y))^0 \) and \( \theta(\text{hasPet}_0^1) = \text{hasPet}(\theta(y), x)^0 \), where \( \theta(y) = a \), are examples of \( \theta \)-substitution, and \( \{\text{OldLady}_0^1, \text{OldLady}_1^1\} \) is a \( \theta \)-complementary connection.

Definition 8. (Set of concepts, Skolem condition). The set of concepts \( \tau(x) \) of a variable or individual \( x \) contains all concepts that were substituted/instantiated by \( x \) so far, i.e. \( \tau(x) \) def \( = \{E(x) \in \text{Path}\} \), where \( E \) is a concept and \( E(x) \) is a substituted/instantiated literal coming from this concept. The Skolem condition ensures that at most one concept is underlined in the graphical matrix. The condition is formally stated as, \( \forall a(\exists^{Path}\theta(a) \in \text{Path}) \leq 1 \), with \( a \) a variable/individual, and \( i \) a column index.

Definition 9. (\( \alpha \)-Related Clause). Let \( C \) be a clause in a matrix \( M \) and \( L \) be a literal in \( M \). \( C \) is \( \alpha \)-related to \( L \), iff \( M \) contains (or is equal to) a matrix \( \{C_1, \ldots, C_n\} \) such that \( C = C_i \) or \( C_i \) contains \( C \), and \( C_j \) contains \( L \) for some \( 1 \leq i, j \leq n \) with \( i \neq j \). \( C \) is \( \alpha \)-related clause to a set of literals \( \mathcal{L} \), iff \( C \) is \( \alpha \)-related to all literals \( L \in \mathcal{L} \).

Example 3. (\( \alpha \)-Related Clause) In the matrix of Example 1 \( \{\text{Animal}_1^1\} \) is \( \alpha \)-related to \( \{\text{hasPet}_0^1, \text{Cat}_1^1\} \).

Definition 10. (Parent Clause). Let \( M \) be a matrix and \( C \) be a clause in \( M \). The clause \( C' = \{M_1, \ldots, M_n\} \) in \( M \) is called the parent clause of \( C \) iff \( C \in M_i \) for some \( 1 \leq i \leq n \).

Example 4. (Parent Clause). In Example 1 \( \{\text{OldLady}_0^1, \{\text{hasPet}_1^1\}, \{\text{Animal}_1^1\}, \{\text{hasPet}_0^1, \text{Cat}_1^1\}\} \) is parent clause of \( \{\text{hasPet}_1^1\} \).

Definition 11. (Extension Clause). Let \( M \) be a matrix and \( P \) a path (be a set of literals). Then the clause \( C \) in \( M \) is an extension clause of \( M \) with respect to \( P \), iff either \( C \) contains a literal of \( P \), or \( C \) is \( \alpha \)-related to all literals of \( P \) occurring in \( M \) and if \( C \) has a parent clause, it contains a literal of \( P \).

In the extension rule of the \( \mathcal{ALC} \) Connection Calculus (3.1) the new subgoal clause (set of literals that need to be connected) is \( C_2 \setminus \{L_2\} \). In the non-clausal connection calculus the extension clause \( C_2 \) might contain clauses that are \( \alpha \)-related to \( L_2 \) and do not need to be considered for the new subgoal clause. Hence, these clauses can be deleted from the subgoal clause. The resulting clause is called the \( \beta \)-clause of \( C_2 \) with respect to \( L_2 \).
Definition 12. (β-Clause). Let \( C = \{M_1, \ldots, M_n\} \) be a clause and \( L \) be a literal in \( C \). The β-Clause of \( C \) with respect to \( L \), denoted by \( \beta\text{-Clause}_L(C) \), is inductively defined:

\[
\beta\text{-Clause}_L(C) := \begin{cases} 
C \setminus \{L\} & \text{if } L \in C, \\
M_1, \ldots, M_{i-1}, \{C^\beta\}, M_{i+1}, \ldots, M_n & \text{otherwise},
\end{cases}
\]

where \( C' \in M_i \) contains \( L \) and \( C^\beta := \beta\text{-Clause}_L(C') \).

Example 5. (Extension Clause, β-Clause). In Example 4 \( C = \{\text{OldLady}^0, \{\text{hasPet}^1\}, \{\text{Animal}^1\}, \{\text{hasPet}^0, \text{Cat}^1\}\} \) is an extension clause with respect to \( p = \{\text{CatOwner}(a)^0, \text{Cat}^0\} \), while the clause \( \{\text{OldLady}^0, \{\text{hasPet}^1\}, \{\text{Animal}^1\}, \{\text{hasPet}^0\}\} \) is a β-Clause of \( C \) with respect to \( L = \text{Cat}^1 \).

3.1 The Formal Non-Clausal ALC θ-Connection Calculus

Suppose we wish to entail if \( O \models \alpha \) is valid using a direct method, like the Connection Method (CM). By the Deduction Theorem [3], we must then prove directly if \( O \rightarrow \alpha \), or, in other words, if \( \neg O \lor \alpha \) is valid. This opposes to classical refutation methods, like tableaux and resolution, which builds a proof by testing whether \( O \lor \neg \alpha \models \bot \). Hence, in the CM, the whole knowledge base \( KB \) should be negated. Given \( O = \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \), \( \alpha_i \) being literal conjunctions in the clausal connection method, all (negated KB) formulae are converted to the Disjunctive Normal Form (DNF). A query then is the matrix \( \neg O \lor \alpha \) (i.e., \( \neg \alpha_1 \lor \neg \alpha_2 \lor \ldots \lor \neg \alpha_n \lor \alpha \)) to be proven valid. In the non-clausal calculus, instead of having clauses only with literals, they can also contain matrices, and no conversion is needed. If every path contains a \((\theta\text{-complementary})\) connection (representing a subformula \( A \lor \neg A \) in a disjunction, which makes this disjunction valid), then the matrix is valid.

Definition 13. (Non-Clausal ALC θ-Connection Calculus) Figure 7 shows the rules of the formal non-clausal ALC \( \theta \)-connection calculus. Rules are applied bottom-up. The words of the calculus are tuples \( C, M, \text{Path} \), where \( C \) is a clause, \( M \) is a matrix corresponding to query \( O \models \alpha \) and \( \text{Path} \) is a set of literals. \( C \) is called the subgoal clause. \( C_1, C_2 \) and \( C_3 \) are clauses. The index \( \mu \in \mathbb{N} \) of a clause \( C^\mu \) denotes that \( C^\mu \) is the \( \mu \)-th copy of clause \( C \), increased when Copy is applied for that clause (the variable \( x \) in \( C^\mu \) is denoted \( x_\mu \)). When Copy is used, it has to be followed by the application of Extension or Reduction, to avoid non-determinism in the rules application. The Blocking Condition is defined as follows: the new individual \( x^\theta_\mu \) (if it is new, then \( x^\theta_\mu \notin \{\text{Path}\} \), as in the condition) is only created if the set of concepts of the previously created individual \( \tau(x^\theta_{\mu-1}) \) is not a subset of the set of concepts of the penultimate copied individual, i.e., \( \tau(x^\theta_{\mu-1}) \not\subseteq \tau(x^\theta_{\mu-2}) \).

The calculus consists of six rules. The Axiom, Start, Reduction and Copy rules are the same as the ones from the ALC \( \theta \)-Connection Calculus. The Extension rule was modified to contain a β-Clause and the Decomposition rule [9] splits subgoal clauses into their sub-clauses.

Lemma 1. (Matrix characterization). A matrix \( M \) is valid iff there exist an index \( \mu \), a set of \( \theta \)-substitutions \( \langle \theta_i \rangle \) and a set of connections \( S \), s.t. every path through \( M^\mu \), the matrix with copied clauses, contains a \( \theta \)-complementary connection \( L_1^\beta, L_2^\beta \) in \( S \), i.e. a connection with \( \theta(L_1) = \theta(L_2) \). The tuple \( \langle \mu, \langle \theta_i \rangle, S \rangle \) is called a matrix proof.

Example 6. (Non-Clausal ALC θ-Connection Calculus). Figure 2 shows the proof for the \( F_1 \) of Example 7 using the matrix representation.

The proof starts (1) by choosing a clause from the consequent as the start clause, in this case, \( \{\text{CatOwner}(a)^0\} \), and a literal of that clause is selected, \( \text{CatOwner}(a)^0 \). This literal is connected to
Converting $\mathcal{ALC}$ Connection Proofs into $\mathcal{ALC}$ Sequents

Axiom(A) $\langle\cdot\rangle, M, Path$

Start(S) $\begin{array}{c}
C_1, M, \langle\cdot\rangle \\
\vdash C_1, M, e
\end{array}$ with $C_1 \in \alpha$

Reduction(R) $\begin{array}{c}
M, Path \cup \{L_2\} \\
C, M, Path \cup \{L_2\}
\end{array}$

Extension(E) $\begin{array}{c}
C_3, M, Path \cup \{L_1\} \\
C \cup \{L_1\}, M, Path
\end{array}$ with $C \vdash C_3 := \beta$–clause$_\uparrow$(C$_2$),

$C_2$ is an extension clause of $M$ wrt. $Path \cup \{L_1\}$, $L_2 \in C_2$, $\theta(L_1) = \theta(L_2)$ and the Skolem condition holds

Decomposition(D) $\begin{array}{c}
C \cup \{L_1\}, M, Path \\
C \cup \{M_1\}, M, Path
\end{array}$ with $C_1 \in M_1$

Copy(C) $\begin{array}{c}
C \cup \{L_1\}, M \cup \{C^e_2\}, Path \\
C \cup \{L_1\}, M, Path
\end{array}$ with $C^e_2$ is a copy of $C_1$, $L_2 \in C^e_2$, $\theta(L_1) = \theta(L_2)$ and the blocking condition holds

Figure 1: Non-clausal $\mathcal{ALC}$ $\theta$-Connection Calculus.

$CatOwner^1$ by an extension step and instance $a$ is the $\theta$-substitution of $CatOwner^1$ and $CatOwner(a)^0$. This connection is still not enough to prove all the paths starting from $CatOwner(a)^0$; the paths that start in it and pass through the literals from the other connected clause, namely, $Cat^0$ and $hasPet^0$, are still to be verified. Indeed, each connection creates two sets of literals to be checked, the remaining literals from each of the clauses involved in the connection. In the new extension step (2), the connection $\{Cat^0, Cat^1\}$ is established on the variable (or fictitious individual) $x$, as it is not necessary yet to commit the substitution with an already existing individual. There is still remaining literals to be verified, the ones resulting from the clause to which $Cat^0$ belongs. Next (3), the $hasPet^0$ predicate is connected, and the $\theta$-substitution generates the pair $(y, x)$ (not shown in figure), for the connection. $OldLady^0$ is connected to $OldLady(a)^1$ (4), and then (5), when the connection $\{hasPet^0, hasPet^1\}$ is settled (using a reduction step, as there was already a connection with the same literal in the path), $y$ was $\theta$-substituted by $y$ (i.e., $\theta(y) = a$), thus forming the pair $(a, x)$. This $\theta$-substitution over $y$ is then propagated through the path. Since every path through $M_1$ contains a $\theta$-complementary connection, $F_1$ is valid. However, the readability of the proof is largely lost by the transformations applied on the formulas to be proven, making it difficult to translate the steps into natural language.

Figure 2: The $\mathcal{ALC}$ non-clausal matrix proof of the $F_1$ using the graphical matrix representation.

Next, we present the Sequent Calculus to which $\mathcal{ALC}$ non-clausal proofs will be translated.
4 An $\mathcal{ALC}$ Sequent Calculus

According to [4], sequent calculi axiomatizes the relation of logical consequence (entailment), and this has an obvious parallel with the relation of subsumption, which is a keystone for DL representation and calculi. Bearing this in mind, Borgida et al proposed a sequent calculus for subsumption inères in $\mathcal{ALC}$ as an extension of the standard sequent calculus, in which there are no rules of implication, as they are indeed subsumption rules, so implication is replaced by $\vdash$ without loss of meaning. In their calculus, terms are not moved from one side to the other of the turnstile during the proof, thus preserving the structure of the original subsumption, and in the case of multiple subsumptions, parentheses help in identifying the main subsumptions. Because of that, additional rules were created in which the negation is inserted in front of each construct, thus eliminating negation rules ($l\neg$, $r\neg$), what requires changing sequent antecedents to successors and vice versa. The calculus is divided in three parts: the first two describe sets of rules, while the last describes a set of axioms (see Figure 3, where $a$ and $b$ are arbitrary formulas and $X$ and $Y$ are arbitrary sequences of formulae).

- **Rules for propositional formulae:** rules $\sqcap$ and $\sqcup$ are duplicated by adding the negation rules for these connectives ($\neg\sqcap$, $\neg\sqcup$), while the proper negation rules ($\neg$) were modified to include the double negation rule ($\neg\neg$);

- **Rules for quantified formulae:** in [4], modal formulae are used ($r\Box$, $l\Diamond$) and their negated rules ($l\neg\Box$, $r\neg\Diamond$). Here, we replace these rules by their equivalents ($r\forall$, $l\exists$ and $l\neg\forall$, $r\neg\exists$). The $\exists$-rules are the dual of $\forall$-rules. A condition is explicitly considered for the application of these rules: the rule applies only if all homologous universal and existential formulae (e.g. $\forall h.C$ and $\exists h.C$ are homologous, $\forall h.C$ and $\exists f.C$ not) are joined together on the left and right sides of the sequent in the precondition. The rule is then applied only once;

- **Termination axioms:** unlike the standard sequent calculus, there are six termination axioms; all of them can be reduced to $X,a \vdash a,Y$ by applying the rules. The application of the $\neg$-rules forces formulae from the antecedent to the successor or vice versa, to be transformed until it gets to $X,a \vdash a,Y$, a procedure that is avoided in this calculus. Therefore, the additional termination axioms are necessary to ensure that formulae are never shifted from one side of the sequent to the other.

Although not stated explicitly, the calculus contains a cut rule, and the cut elimination theorem is valid in this case; it is stated below.

**Theorem 1. Cut Elimination Theorem**. Let $S$ be a set of sequents (axioms) and $s$ an individual sequent. $S \vdash s$, if and only if, there is a proof in $SC$ of $s$ whose leaves are either logical or sequent axioms obtained by the substitution of $S$-belonging sequents, where the cut rule, $\frac{\Gamma \vdash \Delta, \Delta, \Sigma, \Pi}{\Gamma, \Sigma, \Delta, \Pi}$, is only applied with a premise being an axiom.

**Example 7. (Sequent Proof for $\mathcal{ALC}$ Subsumption).** Figure 4 shows Example 1’s proof using the sequent calculus for $\mathcal{ALC}$. The cut rule is applied to the initial assumptions, according to theorem 7.

This proof tree could be described by the following text in natural language: (1) If individuals who own at least one cat as a pet are owners of cats; and if the old ladies are, individuals who have at least one animal as a pet and all individuals who have only cat as pet. So this implies that old ladies own cats. (2) So, the old ladies are all people who have at least one cat as a pet. And all individuals who own at least one cat as a pet, own cats. (3) In addition to old ladies are all individuals who have at least one animal as a pet and all individuals who have only cat as pets; all individuals who have at least one animal as a pet and all individuals who have only cat as pets, are all individuals who have at least one cat as a pet. (4) Thus, an animal or a cat implies in a cat.
Cut rule

\[
\Gamma, \Delta, A \vdash \Pi \\
\Rightarrow \Gamma, \Delta, A, \Sigma \vdash \Pi
\]

Figure 3: The Sequent Calculus for \(\mathcal{ALC}\) Subsumption \([4]\).

\[
\begin{align*}
\text{TRUE} & \quad \vdash A, C \\
\forall h. A & \vdash C \\
\exists h. A & \vdash \forall h. C
\end{align*}
\]

\[
\begin{align*}
& OL + \exists h. A \sqcap \forall h. C \\
& \quad \vdash \exists h. A \sqcap \forall h. C
\end{align*}
\]

\[
\begin{align*}
& \quad \text{cut} \quad \exists h. C + CO \\
& \quad \vdash \exists h. C + CO
\end{align*}
\]

\[
\begin{align*}
& \quad \text{cut} \quad (\exists h. C + CO) \sqcap (OL + \exists h. A \sqcap \forall h. C) + (OL + CO)
\end{align*}
\]

Figure 4: \(\mathcal{ALC}\) sequent proof for \(F_1\). The names of the clauses and the roles are abbreviated.

\section{Conversion Method}

The process consists of two steps: building a formula tree and then converting this formula tree into sequents, given an \(\mathcal{ALC}\) query and its matrix non-clausal connection proof. They are explained below.

\subsection{Building the Formula Tree}

\textbf{Definition 14.} (Formula Tree, Position, Label, Polarity, Type). A \textit{formula tree} is a syntactic representation of a formula \(F\) as a tree, where each node can have up to two child nodes. Each node has:

\textbf{Position:} an index that identifies each element (predicate or connective) in the formula. Its represented as \(a_0, a_1, a_2, \ldots\); \textbf{Label:} either a connective \((\land, \lor, \neg, \exists, \forall)\), quantifier or predicate, if it is an atomic...
(sub-)formula. Nodes whose label is a predicate are leaves of the tree (figure 5b), while other nodes are internal (figure 5a): **Polarity:** can be 0 or 1. It is determined by the label and the parent node polarity. The root node of the tree has polarity 0; **Type:** the type of a node is a Greek letter: $\alpha$, $\beta$, $\alpha'$, $\beta'$, $\gamma$ and $\delta$. It is determined by its label and its polarity. Leaf nodes have no type. The polarity and type of a node are defined in table 2. For example, in the first line of this table, $(A \sqcap B)^1$ means that the node labelled $\sqcap$ and polarity 1 has type $\alpha$ and its successor nodes have polarity 1.

![Image 1](image1.png)

**Figure 5: Node Representation.**

| Type   | Label | Position |
|--------|-------|----------|
| $\alpha$ | $A \sqcap B$ | $A^1 B^1$ |
|        | $A \sqcup B$ | $A^0 B^0$ |
|        | $\neg A$      | $A^0$     |
|        | $\neg A$      | $A^1$     |

| Type   | Label | Position |
|--------|-------|----------|
| $\alpha'$ | $A \sqsubseteq B$ | $A^1 B^0$ |
|        | $A \models B$ | $A^1 B^0$ |

| Type   | Label | Position |
|--------|-------|----------|
| $\beta$ | $(A \sqcap B)^0$ | $A^0 B^0$ |
|        | $(A \sqcup B)^1$ | $A^1 B^1$ |

| Type   | Label | Position |
|--------|-------|----------|
| $\beta'$ | $(A \sqsubseteq B)^1$ | $A^0 B^1$ |
|        | $(A \models B)^0$ | $A^0 B^0$ |

| Type   | Label | Position |
|--------|-------|----------|
| $\delta$ | $(\forall r A)^1$ | $r^1 A^0$ |
|        | $(\exists r A)^1$ | $r^0 A^1$ |

Nodes of type $\alpha$ and $\alpha'$ correspond to sequent rules that do not cause proof branching. Nodes of type $\gamma$ and $\delta$ correspond to quantifier rules. Rules associated to type $\delta$ have the eigenvariable condition in the sequent calculi (where the term $t$, the eigenvariable in the inference, appears in the main formula of inference and in no other formula in the sequent. In the case of the $\exists$ rule for the existential quantifier and $\forall$ rule for the universal quantifier). Nodes of type $\beta$ and $\beta'$ (i.e., $r^0$, $\sqcup^1$, and $\sqsubseteq^1$) are particularly important, since their respective rules in sequents (described in table 2) split proof branching into two independent sub-proofs. Nodes have their types indexed in the formula tree to facilitate their identification, for example $\beta_1, \beta_2, \beta'_1, \beta'_2$. Each branch whose root is of type $\beta$ or $\beta'$ is marked with a letter (a,b,c,...).

Leaf nodes with instances are children of nodes type $\alpha$, $\alpha'$ or $\beta$. Leaf nodes without instances have labels attached to their closest predecessor nodes’ position, according to the following criteria: (1) if the leaf node label represents a concept, it has an unique position associated to its label; (2) if the leaf node label represents a role, it has two positions associated to its label in the form $(a_1, a_2)$, where $a_2$ is the of the nearest predecessor node’s position; (3) only type $\gamma$, $\delta$ and $\beta'$ node positions are associated to the labels. This helps to check for complementarity in a connection between two nodes.

The tree construction is guided by the identification of the (sub-)formulae’s main constructor (connective or quantifier), which will be a label in the tree node. This node has at most two branches that binds them to their child nodes, i.e., new (sub-)formulas. The node type and its childrens polarities are assigned according to table 2. If children nodes are not atomic (sub-)formulae, the process repeats itself by identifying these (sub-)formulae’s main constructor and then generating other nodes in the tree, until it reaches the leaves.
The proof matrix elements must correspond to the leaf nodes in the formula tree, indicated by the position of the corresponding predicate, as explained in section 5.2 step 2.

**Example 8. (Building the Formula Tree Process).** Figure 6 shows the first step in the tree construction for $F_1$ from Example 7. $(\exists h.C \subseteq CO) \cap (OL \subseteq \exists h.A \cap \forall h.C) \models (OL(a) \subseteq CO(a))$. Its main constructor is $\models$, the root node label, which, by definition has polarity 0; its position is $a_0$. According to table 2, its type is $\alpha'$; its children nodes, on the right and left, have polarities 0 and 1, respectively, and both are sub-formulas of $\models$ in $F_1$. This process continues until it reaches the leaf nodes, as shown in figure 7.

![Figure 6: Step 01 Process of building the formula tree for $F_1$.](image)

![Figure 7: Formula Tree for $F_1$ with labels, polarities and types.](image)

For a given formula $A$, $A'$, $B$, $B'$, $\Gamma$ and $\Delta$ are used to denote the sets of node positions of type $\alpha$, $\alpha'$, $\beta$, $\beta'$, $\gamma$, and $\delta$, respectively.

**Definition 15. (Substitution of positions $\sigma_\delta$, ordering relation $\sqsubseteq_\delta$).** It replaces positions of type $\gamma$ for positions of type $\delta$. A position substitution $\sigma_\delta$ is a mapping of the set $\Gamma$ of type $\gamma$ node positions to the set $\Delta$ of type $\delta$ node positions. The $\sigma_\delta$ substitution induces a partial ordering relation $\sqsubseteq_\delta$ in $\Delta \times \Gamma$ as follows: let $u \in \Gamma$ and $v \in \Delta$; if $\sigma_\delta(u) = p$ then $v \sqsubseteq_\delta u$ for all $v \in \Delta$ occurring in position $p$.

Since the sequent rules $r \forall$ and $\exists$ and their homologues $l \forall$ and $r \exists$ are restricted to the eigenvariable condition, the relation $v \sqsubseteq_\delta u$ expresses that the node labelled by $v$ must be reduced before reducing the one labelled by $u$.

**Example 9. (Substitution of positions $\sigma_\delta$, ordering relation $\sqsubseteq_\delta$).** Consider the formula tree in figure 7. Let $u$ be the node labelled by $\forall^1$, with position $a_{13}$ and type $\gamma$, and let $v$ be the node labelled by $\exists^1$, with position $a_{10}$ and type $\delta$. To replace the position of a Type $\gamma$ node by the position of a type $\delta$ node, It is necessary to reduce the type $\delta$ node first, then the node with the position $a_{13}$ must be reduced before the node with the position $a_{10}$. Thus, for this example, the ordering relation $\sqsubseteq_\delta$ is given by $\exists^1 a_{10} \sqsubseteq_\delta \forall^1 a_{13}$, and the substitution $\sigma_\delta(\forall^1 a_{13}) = a_{10}$. With this, we have $\sigma_\delta = \{a_{13} / a_{10}\}$. 

![Diagram with labels, polarities and types](image)
Definition 16. (Substitution of positions $\sigma_{\beta'}$). It replaces positions of type $\beta'$, $\gamma$, $\delta$ for instances or positions of type $\beta'$. Positions of the nodes of type $\beta'$, $\gamma$ and $\delta$, as well as instances, appear in atomic formulas, so a substitution of positions $\sigma_{\beta'}$ is a mapping of the set $B'/\Gamma/\Delta$ positions of nodes of type $\beta'/\gamma/\delta$ to instances or positions of nodes of type $\beta'$. Let $u$ be a leaf node with the positions of nodes of type $\beta'/\gamma/\delta$ associated to its label and $v \in B'$; if $\sigma_{\beta'}(u) = p$, where $p \in B'$ or $p$ is an instance.

Reducing a node means applying the sequent rule that corresponds to that node over a given (sub-)formula. Leaf nodes are not reduced.

Example 10. (Substitution of positions $\sigma_{\beta'}$). Consider the formula tree in Figure 7. Let $u$ be the node labelled by $OL^0$, with position $a_8$ and position $a_7$ of type $\beta'$ associated to its label, and let $v$ be the node labelled by $OL^1$, with position $a_{17}$ and instance $a$. The substitution for this in leaf $u$ in this case is $\sigma_{\beta'}(OL(a_7)^0) = a$. Therefore, $\sigma_{\beta'} = \{a_7/a\}$. 

Definition 17. (Substitution $\sigma_{\text{Final}}$). It is a combination of $\sigma_{\delta}$ and $\sigma_{\beta'}$. A $\sigma_{\text{Final}}$ substitution consists of a substitution $\sigma_{\delta}$ and a substitution $\sigma_{\beta'}$, where $\sigma_{\text{Final}} := \sigma_{\delta} \cup \sigma_{\beta'}$. 

Example 11. (Substitution $\sigma_{\text{Final}}$). Considering the two previous examples, $\sigma_{\text{Final}} = \{a_{13}/a_{10}, a_7/a\}$. 

Definition 18. (Connection, $\sigma_{\text{Final}}$-complementary connection). A connection is a pair of leaf nodes labelled with the same predicate symbol and the same position associated with the label or the same instance, but with different polarities. If they are identical under $\sigma_{\text{Final}}$, the connection is a $\sigma_{\text{Final}}$-complementary connection.

Example 12. (Connection, $\sigma_{\text{Final}}$-complementary connection). Let the formula tree in Figure 7 be. The leaf nodes $h(a_2, a_3)^0$ and $h(a_7, a_{10})^1$ with positions $a_4$ and $a_{11}$, respectively, form a connection that is complementary under $\sigma_{\text{Final}} = \{a_2/a_7, a_3/a_{10}\}$. 

Definition 19. (Tree Ordering $\prec$). The tree ordering $\prec$ of an $F$ formula is the partial ordering of the nodes positions in the formula tree. $\prec$ is defined as follows: (i) the root occupies the smallest position with respect to this ordering, (ii) $a_i \prec a_j$ if and only if the position $a_i$ is below $a_j$ in the formula tree. 

Example 13. (Tree Ordering $\prec$). In the tree from Figure 7 there are examples of tree ordering: $a_7 \prec a_9 < a_{13} < a_{15}$ and $a_0 < a_1 < a_2 < a_3$. 

Definition 20. (Reduction Order $\ll$). The transitive closure of the union of $\subseteq_{\delta}$, $\subseteq_{\beta'}$ and $\prec$ is called reduction order $\ll$, i.e., $\ll := (\subseteq_{\delta} \cup \subseteq_{\beta'} \cup \prec)^*$. 

Nodes $v \ll u$ means that the node $v$ must be reduced before the node labelled by $u$ in the sequent proof. $\ll$ determines the nodes’ reduction order, and helps determine which sequent rules are to be used and in which order.

Example 14. (Reduction Order $\ll$). In Figure 7 the nodes with positions $a_7$, $a_{10}$, $a_{16}$ and $a_{13}$, have the following reduction order $\ll$: (i) $a_7 \ll a_{10}$; (ii) $a_7 < a_{13}$; (iii) $a_{10} \ll a_{13}$. The orderings’ union and the tree ordering determine the reduction order for these nodes: $a_7 < a_{10} < a_{13}$. 

Definition 21. (\sigma_{\text{Final}} Admissible Substitution). An $\sigma_{\text{Final}}$ Substitution is admissible if the reduction order $\ll$ is not reflexive. In this case, it is possible to construct a sequent proof. 

A correspondence between node label, polarity and type with the sequent rules presented in section 4 is established in Table 3. Such correspondence is useful for the sequent proof construction, where the polarity helps in the identification of the rule. Polarity 1 represents a rule on the left (left or l); polarity 0, on the right (right or r), for cases where there is already an associated rule. For instance, in Table 3’s first line, for node $\Gamma$ the rule is $l\Gamma$, while for node $\Gamma^0$ it is $r\Gamma$. For cases where internal nodes are preceded by a node labelled by a negation, correspondences are in Table 3’s last four columns.
Table 3: Correspondence between label, polarity and type of a node, preceded or not by a node labelled with negation, to \(\mathcal{ALC}\) Sequent rules.

| Type \(\alpha\) | Rule | Not preceded | Type \(\beta\) | Rule | Type \(\delta\) | Rule | Preceded |
|----------------|------|--------------|----------------|------|----------------|------|----------|
| \(\lor^1\)    | \(\lor\) | \(\lor\)     | \(\lor^0\)    | \(\lor\) | \(\lor\)      | \(\lor\) | \(\lor\) |
| \(\lor^0\)    | \(\lor\) | \(\lor\)     | \(\lor^1\)    | \(\lor\) | \(\lor\)      | \(\lor\) | \(\lor\) |
| \(\neg\)      | \(\neg\) | \(\neg\)     | \(\neg\)      | \(\neg\) | \(\neg\)      | \(\neg\) | \(\neg\) |
| \(\neg^0\)    | \(\neg\) | \(\neg\)     | \(\neg^0\)    | \(\neg\) | \(\neg\)      | \(\neg\) | \(\neg\) |
| Type \(\alpha'\) | Rule | Type \(\beta'\) | Rule | Type \(\gamma\) | Rule | Type \(\delta\) | Rule |
| \(\exists^0\)  | \(\exists\) | \(\exists\)   | \(\exists^1\)  | \(\exists\) | \(\exists\)   | \(\exists\) | \(\exists\) |
| \(\exists^0\)  | \(\exists\) | \(\exists\)   | \(\exists^1\)  | \(\exists\) | \(\exists\)   | \(\exists\) | \(\exists\) |

5.2 Conversion to Sequents

Given an \(\mathcal{ALC}\) query and its matricial non-clausal connection proof, the conversion procedure transforms this proof into an \(\mathcal{ALC}\) sequent proof. This process performs four steps, which are described below:

- **Step 1- Formula tree construction:** A syntactic representation in tree form is constructed for the input formula, containing nodes, as described in 14. The position of each predicate is input to step 2, and the tree to steps 3 and 4. **Example:** The conversion process begins with the \(F_1\) formula tree construction, described in definition 14, which resulted in the formula tree represented in figure 7.

- **Step 2- Matrix elements’ positions assignment:** Since proof matrix elements correspond to predicates in the formula and also to leaf nodes in the formula tree, this step assigns to each matrix element the position of the corresponding predicate. Its input is the matrix non-clausal connection proof and the position of predicates. Its output is input to step 3. **Example:** Each element of the matrix is assigned with the position of the corresponding predicate in the formula, see matrix in 8.

![Figure 8: Steps representation in the connection proof/sequent \(\mathcal{ALC}\) for \(F_1\).](image)

- **Step 3- (partial) sequent proof structure Construction:** The matrix non-clausal connection proof with the positions of each element and the formula tree are inputs for this step. To each matrix connection, the formula tree is examined in search for the leaf nodes that correspond to the connection. The paths between the root node and these nodes in the tree are analyzed to determine the order of nodes to be worked on and thus build a structure of the (partial) proof in sequents. This
structure provides information about the reduction order $\prec$, which helps determine the rules to be applied, and on the existence of the proof branch, given by the identification of the nodes of type $\beta$ and $\beta'$. The (partial) sequent proof structure constructed will be the input for step 4. Example: The first connection links element $CO(a)^0$, from position $a_{18}$, to element $CO^1$, of position $a_6$, which are complementary under the substitution $\sigma_{\beta'} = [a_2/a]$, see table [2]. The path between these leaf nodes is $\{a_{18}, a_{16}, a_0, a_1, a_2, a_6\}$. Since there is no ordering relation $\subseteq_{\sigma}$ between the nodes of that path and there are two tree orderings given by $a_0 < a_{16}$ and $a_0 < a_1 < a_2$, it is possible to start with any of these tree orderings. Choosing the first, we have the order of reduction at that moment equal to: $a_0 < a_{16} < a_1 < a_2$. Since the node with position $a_2$ is of type $\beta'$, the sequent is divided into two branches, called a e b, as in the formula tree. Thus, this connection closes the branch b, branch where the node $CO^1$ is, and leads to the axiom $h'^0, C^0 \vdash CO^1$, because nodes of type $\beta'$ are associated with the cut rule (see table [3]). In the second connection, $C^0$, with position $a_5$ in branch a, is connected to $C^1$, with position $a_{15}$ in branch d, and the path between them is $\{a_5, a_3, a_2, a_1, a_7, a_9, a_{13}, a_{15}\}$. As the nodes with positions $a_1$ and $a_2$ have already been reduced, it is necessary to reduce the nodes with positions, $a_3$, $a_7$, $a_9$ and $a_{13}$, which have tree ordering $a_7 < a_9 < a_{13}$ and the relations $a_{10} \subseteq_{\delta} a_3$ and $a_{10} \subseteq_{\delta} a_{13}$. At the moment it is only possible to reduce the node with position $a_7$ and then the node with position $a_0$, that is, $a_7 < a_9$. Since the node with position $a_9$ is of type $\beta'$, its reduction divides branch 'a' into branches 'c' and 'd'. Then the node with position $a_9$, in branch 'd', is reduced. Since there are pendant nodes on this path, it is not yet possible to form an axiom and close the 'd' branch. The third connection is analyzed, where $h'^0$, with position $a_{14}$, is connected to $h^1$, with position $a_{11}$, both in branch 'd'. The path between the nodes is $\{a_{14}, a_{13}, a_9, a_{10}, a_{11}\}$. Since $a_9$ has already been reduced, and there are the relations $a_{10} \subseteq_{\delta} a_{13}$ and $a_{10} \subseteq_{\delta} a_3$, the $a_{10}$ position node is reduced, and 'together' with it the nodes with position $a_{13}$ and $a_3$. The reduction of the $a_{10}$ position node makes the third and second connection complementary under the substitutions $\sigma_{\delta} = \{a_{13}/a_{10}, a_3/a_{10}\}$. With this the last two connections are reflected in the sequent proof leading to the closure of the 'd' branch. Notice that the second connection was only reached in the tree after the third connection, this leads to the axiom in the form $C^1 \vdash C^0$. The fourth connection connects $OL^0$, with position $a_8$ in branch 'c', to $OL^1$, with position $a_{17}$. The path between the nodes with theses positions is $\{a_8, a_7, a_1, a_0, a_{16}, a_{17}\}$. As all nodes on this path have already been reduced, no reduction will be necessary in this step. Thus, 'c' branch is closed with an axiom in the form $OL^0 \vdash h^1 A^1, h'^0 C^1$, due to the cut rule. This connection is complementary under $\sigma_{\beta'} = [a_2/a]$. On the fifth and last connection, which connects $h^0$ to $h^1$, there is no need of node reduction, since all nodes in the path were reduced. The connection is complementary under $\sigma_{\delta} = \{a_3/a_{10}\}$. Note that $a_2/a$ and $a_2/a$ were $\sigma_{\beta'}$ previous substitutions. All connections are complementary under a substitution $\sigma_{\text{Final}}$, all branches of the proof structure in sequent were closed, and the reduction order is not reflexive, as shown in figure [8] and in Table [4].

• Step 4- Construction of the complete sequent proof: Here, the process builds a complete sequent proof (output) from the (partial) sequent proof structure and the correspondence between nodes and sequent rules, described in [3]. The input is (partial) sequent proof structure, the formula tree and $\text{ALC}$ sequent rules. Example: The structure obtained in step 3 is traversed. The proof begins with the reduction of $a_1$ position node, since the first two tree nodes do not have associated rule, because they are of type $\alpha'$. Rule $\Gamma \vdash \alpha$ is applied. Then, the $a_2$ position node, with type $\beta'$, reduced by means of the cut rule on the query $\alpha$, that is, on $(OL \vdash CO)$. The proof is divided into branches 'a' and 'b'. The 'b' branch is closed with the initial axiom $\exists h.C \vdash CO$, while branch 'a' is open, in which $OL \vdash \exists h.C$ must be proved. The next node is of position $a_7$, of type $\beta'$, and its reduction
Table 4: Relation between connections, substitutions and orderings

| N | Nodes | $\sigma_\delta$ | $\sigma_{\beta'}$ | $\subseteq_\delta$ | $\prec$ |
|---|-------|-----------------|------------------|-----------------|--------|
| 1 | CO($a_2$)$^1$/$a_6$, CO($a$)$^0$/$a_{18}$ | $a_2$/$a$ | $a_2$/$a$ | $a_0 < a_1 < a_2$ |
| 2 | C($a_3$)$^0$/$a_5$, C($a_{13}$)$^1$/$a_{15}$ | $a_{13}$/$a_{10}$, $a_3$/$a_{10}$ | $a_{10} \subseteq_\delta a_3$, $a_{10} \subseteq_\delta a_{13}$ | $a_7 < a_9$ |
| 3 | OL($a_7$)$^\nu$/$a_8$, OL($a$)$^1$/$a_{17}$ | $a_7$/$a$ | $a_7$/$a_2$ | $a_0 < a_1 < a_2 < a_7 < a_9 < a_{10}$ |
| 4 | h($a_2$, $a_3$)$^3$/$a_4$, h($a_7$, $a_{10}$)$^1$/$a_{11}$ | $a_3$/$a_{10}$ | $a_2$/$a$ | $a_0 < a_1 < a_2 < a_7 < a_9 < a_{10}$ |
| 5 | h($a_2$, $a_3$)$^3$/$a_4$, h($a_7$, $a_{10}$)$^1$/$a_{11}$ | $a_3$/$a_{10}$ | $a_2$/$a$ | $a_0 < a_1 < a_2 < a_7 < a_9 < a_{10}$ |

divides the branch 'a' into branches 'c' and 'd', by means of the application of a new cut rule on $OL \vdash \exists h.C$. The 'c' branch is closed with the initial axiom $OL \vdash \exists h.A \sqcap \forall h.C$, while the 'd' branch stays open. To close the 'd' branch, the $a_9$ position node is reduced with the $l \sqcap$ rule, followed by the node with position $a_{10}$, through rule $\exists$. This ends the $F_1$ sequent proof, as shown in figure 9.

$$
\begin{align*}
\text{Figure 9: Complete proof in } ALC \text{ sequents for } F_1.
\end{align*}
$$

6 Complexity

This section presents a very brief overview of the main algorithms for the conversion method with its complexities, according to the 4 steps seen in section 5.2. All the algorithms are demonstrated in [10]. Time complexities were analyzed according to the input size of each algorithm. For example, some algorithms receive an $ALC$ formula $F$ as input, so the input size $n$ represents the number of symbols of $F$. Other algorithms accept an $F$ proof matrix as input; in this case, the input size is the matrix number of symbols, including connections between literals. This input is represented by $m$.

Figure 10 presents the main algorithms' execution order. Lines with arrows indicate that the output of one algorithm is input to another. For example, the output from algorithm 02 (called convertsPostFix) is conveyed as input for algorithm 03 (called buildTree and 04 (called assignPosition). The complexity of algorithm 05 (Search Connections) is the highest among the algorithms: $O(n^4)$, up to four iterations over structures based on the input size $m$.

7 Conclusions

This work presents a method to convert Non-clausal $ALC$ connections proofs into more readable proofs. The approach consists in transforming these proofs into proofs in the $ALC$-Sequent Calculus [4]. Hence,
this conversion assumes that the input formulae will always be in non-clausal form, i.e., without the need to transform these formulae into any normal form. A tree representation of formulae is used as a guide in this conversion and a sequent proof is created while the connection proof is traversed. This conversion must contribute to describe how the reasoners based on the ALC Connection Method summon their inferences and may facilitate the creation of natural language explanations, given the ease of converting sequents to texts. The evaluation of the main algorithms’ computational complexities demonstrates its practical feasibility, since they display polynomial complexity. In this perspective, the scientific contributions of this work should characterize the importance of the logical proofs, clarify the reasoning process and increase inferences’ readability, thus providing better user interaction with connection reasoners.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi & P. F. Patel-Schneider, editors (2003): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press.

[2] F. Baader, I. Horrocks & U. Sattler (2008): Description Logics. In: Handbook of Knowledge Representation, Foundations of Artificial Intelligence 3, Elsevier, pp. 135–179, doi:10.1016/S1574-6526(07)03003-9.

[3] W. Bibel (1993): Deduction - automated logic. Academic Press.

[4] A. Borgida, E. Franconi & I. Horrocks (2000): Explaining ALC Subsumption. In: ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, 2000, pp. 209–213.

[5] F. Freitas & J. Otten (2016): A Connection Calculus for the Description Logic ALC. In: Advances in Artificial Intelligence - 29th Canadian Conference on Artificial Intelligence, Canadian AI 2016, Victoria, BC, Canada, May 31 - June 3, 2016. Proceedings, pp. 243–256, doi:10.1007/978-3-319-34111-8_30.

[6] Jean-Yves Girard, Paul Taylor & Yves Lafont (1989): Proofs and Types. Cambridge University Press.

[7] I. Horrocks (2008): Ontologies and the semantic web. Commun. ACM 51(12), pp. 58–67, doi:10.1145/1409360.1409377.

[8] D. Melo, F. Freitas & J. Otten (2017): RACCOON: A Connection Reasoner for the Description Logic ALC. In: LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017, pp. 200–211.

[9] J. Otten (2011): A Non-clausal Connection Calculus. In: Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings, pp. 226–241, doi:10.1007/978-3-642-22119-4_18.

[10] E. Palmeira (2017): Conversion of Proof in Description Logic ALC Generated by Connection Method into Sequents. Ph.D. thesis, Federal University of Pernambuco.

[11] D. A. Plaisted & S. Greenbaum (1986): A Structure-Preserving Clause Form Translation. J. Symb. Comput. 2(3), pp. 293–304, doi:10.1016/S0747-7171(86)80028-1.