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) [1] 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 $\mathcal{ALC}$ θ-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 θ-Non-clausal $\mathcal{ALC}$ θ-Connection Calculus is based on the $\mathcal{ALC}$ θ-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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$ θ-connection proofs into $\mathcal{ALC}$ 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 $\mathcal{ALC}$ is presented in the next section; Section 3 brings an $\mathcal{ALC}$ non-clausal Connection Calculus for $\mathcal{ALC}$; Section 4 introduces the $\mathcal{ALC}$ 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 $\mathcal{ALC}$

An ontology O in $\mathcal{ALC}$ 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 th following formulae are also *concepts*: (i) $C \sqcap D$, (ii) $C \sqcup D$, (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) $C \sqsubseteq D$; (ii) $C \equiv D$, standing for $C \sqsubseteq D$ and $D \sqsubseteq C$. 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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$ θ-Connection Calculus

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

Definition 2. ***(Literal, clause, matrix).*** *$\mathcal{ALC}$ **Literals** are atomic concepts or roles, possibly negated or instantiated in the form L or $\neg L$. An $\mathcal{ALC}$ disjunction is either a literal L, a disjunction $(E_0 \sqcup E_1)$ or an universal restriction $\forall r.E_0$. An $\mathcal{ALC}$ 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. ***($\mathcal{ALC}$ Non-Clausal Matrix).*** *An $\mathcal{ALC}$ **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 1, 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 $\mathcal{ALC}$ formula F^p.

Type	F^p	$M(F^p)$
Atomic	A^0	$\{\{A^0\}\}$
	A^1	$\{\{A^1\}\}$
α	$(\neg C)^0$	$M(C^1)$
	$(\neg C)^1$	$M(C^0)$
	$(C \sqcap D)^1$	$\{\{M(C^1)\},\{M(D^1)\}\}$
	$(C \sqcup D)^0$	$\{\{M(C^0)\},\{M(D^0)\}\}$
	$(C \sqsubseteq D)^0$	$\{\{M(C^1)\},\{M(D^0)\}\}$
	$(C \models D)^0$	$\{\{M(C^1)\},\{M(D^0)\}\}$
β	$(C \sqcap D)^0$	$\{\{M(C^0),M(D^0)\}\}$
	$(C \sqcup D)^1$	$\{\{M(C^1),M(D^1)\}\}$
	$(C \sqsubseteq D)^1$	$\{\{M(C^0),M(D^1)\}\}$
γ	$(\forall rD)^1$	$\{\{M(\underline{r^0}),M(\underline{D^1})\}\}$
	$(\exists rD)^0$	$\{\{M(\underline{r^0}),M(\underline{D^0})\}\}$
δ	$(\forall rD)^0$	$\{\{M(\underline{r^1})\},\{M(\underline{D^0})\}\}$
	$(\exists rD)^1$	$\{\{M(\underline{r^1})\},\{M(\underline{D^1})\}\}$

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 1); restrictions with indexes are represented with horizontal lines; restrictions without indexes with vertical lines.*

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

$$\left.\begin{array}{r}\forall x((\exists y\; hasPet(x,y) \land Cat(y)) \rightarrow CatOwner(x))\\ \forall z(OldLady(z) \rightarrow \exists v(hasPet(z,v) \land Animal(v)))\\ \land \forall k(hasPet(z,k) \rightarrow Cat(k)))\end{array}\right\} \models \forall u(OldLady(u) \rightarrow CatOwner(u))$$

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

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

and by the following $\mathcal{ALC}$ 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):

$$\{\{\underline{hasPet^0}, \underline{Cat^0}, CatOwner^1\}, \{OldLady^0, \{\{\underline{hasPet^1_1}\}, \{\underline{Animal^1_1}\}, \{\underline{hasPet^0}, \underline{Cat^1}\}\}\},$$
$$\{OldLady(a)^1\}, \{CatOwner(a)^0\}\}$$

So, the graphical representation of M_1 is:

$$\left[\left[\begin{array}{c|}hasPet^0\\ Cat^0\\ CatOwner^1\end{array}\right]\left[\begin{array}{c}OldLady^0\\ \left[\underline{[hasPet^1_1][Animal^1_1]}\left[\begin{array}{c|}hasPet^0\\ Cat^1\end{array}\right]\right]\end{array}\right][OldLady(a)^1][CatOwner(a)^0]\right]$$

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., $\bigcup_{i=1}^{n}\{L_i\}$ with $L_i \in C_i$. 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, θ-substitution, θ-complementary connection).*** *A* ***connection*** *is a pair of literals $\{E, \neg E\}$ with the same concept/role name, but different polarities. A* ***θ-substitution*** *assigns to each (possibly omitted) variable an individual or another variable (in the whole matrix). A* ***θ-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 θ-substitutions. The application of a θ-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, $x^{\theta} = \theta(x)$.

Example 2. ***(Path, Connection, θ-substitution, θ-complementary connection).*** *In the matrix M_1 of Example 1, $\{hasPet^0 \mid, \underline{hasPet_1^1}, \underline{Animal_1^1}, hasPet^0 \mid, OldLady(a)^1, CatOwner(a)^0\}$ and $\{Cat^0, \underline{hasPet_1^1}, \underline{Animal_1^1}, Cat^1 \mid, OldLady(a)^1, CatOwner(a)^0\}$ are some paths through M_1. $\{Cat^0 \mid, Cat^1\}$ is a connection. $\theta(OldLady^0) = OldLady(\theta(y))^0$ and $\theta(hasPet^0) = hasPet(\theta(y), x)^0$, where $\theta(y) = a$, are examples of θ-substitution, and $\{OldLady^0, OldLady(a)^1\}$ is a θ-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. $\tau(x) \overset{def}{=} \{E(x) \in Path\}$, 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, $\forall a |\{\underline{E^i(a)} \in Path\}| \leq 1$, with a a variable/individual, and i a column index.*

Definition 9. ***(α-Related Clause).*** *Let C be a clause in a matrix M and L be a literal in M. C is α-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* ***α-related clause*** *to a set of literals $\mathcal{L}$, iff C is α-related to all literals $L \in \mathcal{L}$.*

Example 3. ***(α-Related Clause)*** *In the matrix of Example 1, $\{\underline{Animal_1^1}\}$ is α-related to $\{hasPet^0, Cat^1\}$.*

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, $\{OldLady^0, \{\{\underline{hasPet_1^1}\}, \{\underline{Animal_1^1}\}, \{hasPet^0, Cat^1\}\}\}$ is parent clause of $\{\underline{hasPet_1^1}\}$.*

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 α-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 $\mathcal{ALC}$ θ-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 α-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 β-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 1, $C = \{OldLady^0, \{\{\underline{hasPet_1^1}\}, \{\underline{Animal_1^1}\}, \{\underline{hasPet^0}, \underline{Cat^1}\}\}\}$ is an extension clause with respect to $p = \{CatOwner(a)^0, \underline{Cat^0}\}$, while the clause $\{OldLady^0, \{\{\underline{hasPet_1^1}\}, \{\underline{Animal_1^1}\}, \{hasPet^0\}\}\}$ is a β-Clause of C with respect to $L = \underline{Cat^1}$.*

3.1 The Formal Non-Clausal $\mathcal{ALC}$ 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 $\neg O \vee \alpha$ is valid. This opposes to classical refutation methods, like tableaux and resolution, which builds a proof by testing whether $O \cup \{\neg\alpha\} \models \bot$. Hence, in the CM, the whole knowledge base KB should be negated. Given $O = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, α_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 $\neg O \vee \alpha$ (i.e., $\neg\alpha_1 \vee \neg\alpha_2 \vee \ldots \vee \neg\alpha_n \vee \alpha$) 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 (θ-complementary) connection (representing a subformula $A \sqcup \neg A$ in a disjunction, what makes this disjunction valid), then the matrix is valid.

Definition 13. ***(Non-Clausal $\mathcal{ALC}$ θ-Connection Calculus)*** *Figure 1 shows the rules of the formal non-clausal $\mathcal{ALC}$ θ-connection calculus. Rules are applied bottom-up. The words of the calculus are tuples $C, M, Path$, where C is a clause, M is a matrix corresponding to query $O \models \alpha$ and $Path$ 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^μ denotes that C^μ is the μ-th copy of clause C, increased when Copy is applied for that clause (the variable x in C^μ is denoted x_μ). 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_μ^θ (if it is new, then $x_\mu^\theta \notin N_O$, as in the condition) is only created if the set of concepts of the previously created individual $\tau(x_{\mu-1}^\theta)$ is not a subset of the set of concepts of the penultimate copied individual, i.e., $\tau(x_{\mu-1}^\theta) \not\subseteq \tau(x_{\mu-2}^\theta)$.*

The calculus consists of six rules. The Axiom, Start, Reduction and Copy rules are the same as the ones from the $\mathcal{ALC}$ θ-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 μ, a set of θ-substitutions $\langle\theta_i\rangle$ and a set of connections S, s.t. every path through M^μ, the matrix with copied clauses, contains a θ-complementary connection L_1^θ, L_2^θ in S, i.e. a connection with $\theta(L_1) = \theta\left(\overline{L_2}\right)$. The tuple $\langle\mu, \langle\theta_i\rangle, S\rangle$ is called a matrix proof.*

Example 6. ***(Non-Clausal $\mathcal{ALC}$ θ-Connection Calculus).*** *Figure 2 shows the proof for the F_1 of Example 1 using the matrix representation.*

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

$$
\begin{array}{rl}
Axiom(A) & \dfrac{}{\{\}, M, Path} \\[2ex]
Start(S) & \dfrac{C_1, M, \{\}}{\varepsilon, M, \varepsilon} \text{ with } C_1 \in \alpha \\[2ex]
Reduction(R) & \dfrac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}} \\
 & \text{with } \theta(L_1) = \theta(\overline{L_2}) \text{ and the Skolem condition holds} \\[2ex]
Extension(E) & \dfrac{C_3, M, Path \cup \{L_1\} \qquad C, M, Path}{C \cup \{L_1\}, M, Path} \text{ with } C_3 := \beta\text{–}clause_{L_2}(C_2), \\
 & C_2 \text{ is an extension clause of } M \text{ wrt. } Path \cup \{L_1\}, \\
 & L_2 \in C_2,\ \theta(L_1) = \theta(\overline{L_2}) \text{ and the Skolem condition holds} \\[2ex]
Decomposition(D) & \dfrac{C \cup C_1, M, Path}{C \cup \{M_1\}, M, Path} \text{ with } C_1 \in M_1 \\[2ex]
Copy(C) & \dfrac{C \cup \{L_1\}, M \cup \{C_2^{\mu}\}, Path}{C \cup \{L_1\}, M, Path} \text{ with } C_2^{\mu} \text{ is a copy of } C_1, \\
 & L_2 \in C_2^{\mu},\ \theta(L_1) = \theta(\overline{L_2}) \text{ and the blocking condition holds}
\end{array}
$$

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

CatOwner^1 by an extension step and instance a is the θ-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 θ-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 θ-substituted by y (i.e., $\theta(y) = a$), thus forming the pair *(a,x)*. This θ-substitution over y is then propagated through the path. Since every path through M_1 contains a θ-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.

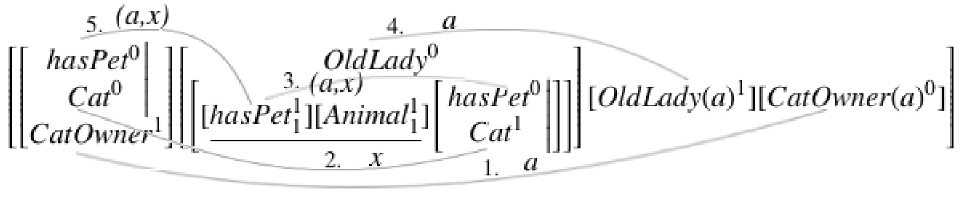


Figure 2: The $\mathcal{ALC}$ non-clausal matrix proof of the F_1 using the graphical matrix representation.

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 inferences 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 $\forall$-*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 [6].*** *Let S be a set of sequents (axioms) and s an individual sequent. $S \vdash_{SC} s$, 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, $\frac{\Gamma \vdash \Delta, A \qquad A, \Sigma \vdash \Pi}{\Gamma, \Sigma \vdash \Delta, \Pi}$, 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 1.*

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.

Rules for propositional formulae

$$\frac{X, a, b \vdash Y}{X,\, a \sqcap b \vdash Y}\ (l\sqcap) \qquad \frac{X \vdash a, Y \quad X \vdash b, Y}{X, \vdash a \sqcap b, Y}\ (r\sqcap)$$

$$\frac{X, \neg a \vdash Y \quad X, \neg b \vdash Y}{X, \neg(a \sqcap b) \vdash Y}\ (l\neg\sqcap) \qquad \frac{X \vdash \neg a, \neg b, Y}{X \vdash \neg(a \sqcap b), Y}\ (r\neg\sqcap)$$

$$\frac{X, a \vdash Y \quad X, b \vdash Y}{X,\, a \sqcup b \vdash Y}\ (l\sqcup) \qquad \frac{X \vdash a, b, Y}{X \vdash a \sqcup b, Y}\ (r\sqcup)$$

$$\frac{X, \neg a, \neg b \vdash Y}{X, \neg(a \sqcup b) \vdash Y}\ (l\neg\sqcup) \qquad \frac{X \vdash \neg a, Y \quad X \vdash \neg b, Y}{X \vdash \neg(a \sqcup b), Y}\ (r\neg\sqcup)$$

$$\frac{X, a \vdash Y}{X, \neg\neg a \vdash Y}\ (l\neg\neg) \qquad \frac{X \vdash a, Y}{X \vdash \neg\neg a, Y}\ (r\neg\neg)$$

Rules for quantified formulae

$$\frac{X' \vdash b, Y'}{X \vdash \forall r.b, Y}\ (r\forall) \qquad \frac{X', b \vdash Y'}{X, \exists r.b \vdash Y}\ (l\exists)$$

$$\frac{X', \neg b \vdash Y'}{X, \neg\forall r.b \vdash Y}\ (l\neg\forall) \qquad \frac{X' \vdash \neg b, Y'}{X \vdash \neg\exists r.b, Y}\ (r\neg\exists)$$

where $X' = \{a \mid \forall r.a \in X\} \cup \{\neg a \mid \neg\exists r.a \in X\}$, and
$Y' = \{a \mid \exists r.a \in Y\} \cup \{\neg a \mid \neg\forall r.a \in Y\}$

Termination axioms

$X, a \vdash a, Y$	$(=)$	$X, \neg a \vdash \neg a, Y$	$(=)$
$X, a, \neg a \vdash Y$	$(\mathrm{l}\uparrow)$	$X \vdash a, \neg a, Y$	$(\mathrm{r}\uparrow)$
$X, \bot \vdash Y$	$(\mathrm{l}\bot)$	$X \vdash \top, Y$	$(\mathrm{l}\top)$

Cut rule

$$\frac{\Gamma \vdash \Delta, A \qquad A, \Sigma \vdash \Pi}{\Gamma, \Sigma \vdash \Delta, \Pi}$$

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

$$\dfrac{\dfrac{OL \vdash \exists h.A \sqcap \forall h.C \qquad \dfrac{\dfrac{\dfrac{\dfrac{\text{TRUE}}{A, C \vdash C}\,=}{\exists h.A, \forall h.C \vdash \exists h.C}\,\mathrm{l}\exists}{\exists h.A \sqcap \forall h.C \vdash \exists h.C}\,\mathrm{l}\sqcap}{OL \vdash \exists h.C}\,\mathrm{cut} \qquad \exists h.C \vdash CO}{(\exists h.C \vdash CO,\ OL \vdash \exists h.A \sqcap \forall h.C) \vdash (OL \vdash CO)}\,\mathrm{cut}}{\Big(((\exists h.C \vdash CO) \sqcap (OL \vdash \exists h.A \sqcap \forall h.C)) \vdash (OL \vdash CO)\Big)}\,\mathrm{l}\sqcap$$

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

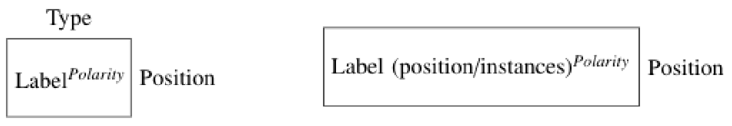
5 Conversion Method

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.

5.1 Building the Formula Tree

Definition 14. ***(Formula Tree, Position, Label, Polarity, Type).*** *A* ***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:*

Position: *an index that identifies each element (predicate or connective) in the formula. Its represented as $a_0, a_1, a_2, \ldots$;* ***Label:*** *either a connective ($\sqcap, \sqcup, \neg, \sqsubseteq, \models$), 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: α, β, α', β', γ and δ. 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 α and its successor nodes have polarity 1.*



(a) Internal Node (b) Leaf Node

Figure 5: Node Representation.

Table 2: Polarity and types of nodes for $\mathcal{ALC}$

Type α		Type β		Type δ	
$(A \sqcap B)^1$	A^1 B^1	$(A \sqcap B)^0$	A^0 B^0	$(\forall rA)^0$	r^1 A^0
$(A \sqcup B)^0$	A^0 B^0	$(A \sqcup B)^1$	A^1 B^1	$(\exists rA)^1$	r^1 A^1
$(\neg A)^1$	A^0				
$(\neg A)^0$	A^1				
Type α'		Type β'		Type γ	
$(A \sqsubseteq B)^0$	A^1 B^0	$(A \sqsubseteq B)^1$	A^0 B^1	$(\forall rA)^1$	r^0 A^1
$(A \models B)^0$	A^1 B^0			$(\exists rA)^0$	r^0 A^0

Nodes of type α and α' correspond to sequent rules that do not cause proof branching. Nodes of type γ and δ correspond to quantifier rules. Rules associated to type δ have the eigenvariable condition in the sequent calculi (where the term t, the eingevariable in the inference, appears in the main formula of inference and in no other formula in the sequent. In the case of the l$\exists$ rule for the existential quantifier and r$\forall$ rule for the universal quantifier). Nodes of type β and β' (i.e., $\sqcap^0$, $\sqcup^1$, and $\sqsubseteq^1$) are particularly important, since their respective rules in sequents (described in table 3) 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 β or β' is marked with a letter (a,b,c,...).

Leaf nodes with instances are children of nodes type α, α' or β. 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 γ, δ and β' 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 1: $((\exists h.C \sqsubseteq CO) \sqcap (OL \sqsubseteq \exists h.A \sqcap \forall h.C)) \models (OL(a) \sqsubseteq CO(a))$. 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 α'; 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.*

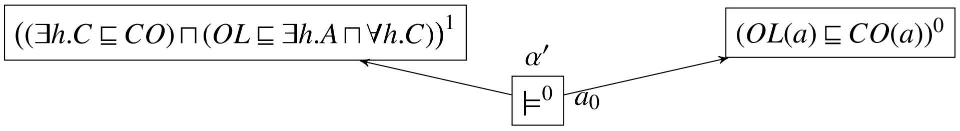

$((\exists h.C \sqsubseteq CO) \sqcap (OL \sqsubseteq \exists h.A \sqcap \forall h.C))^1$
$(OL(a) \sqsubseteq CO(a))^0$
α'
$\models^0$
a_0

Figure 6: Step 01 Process of building the formula tree for F_1.

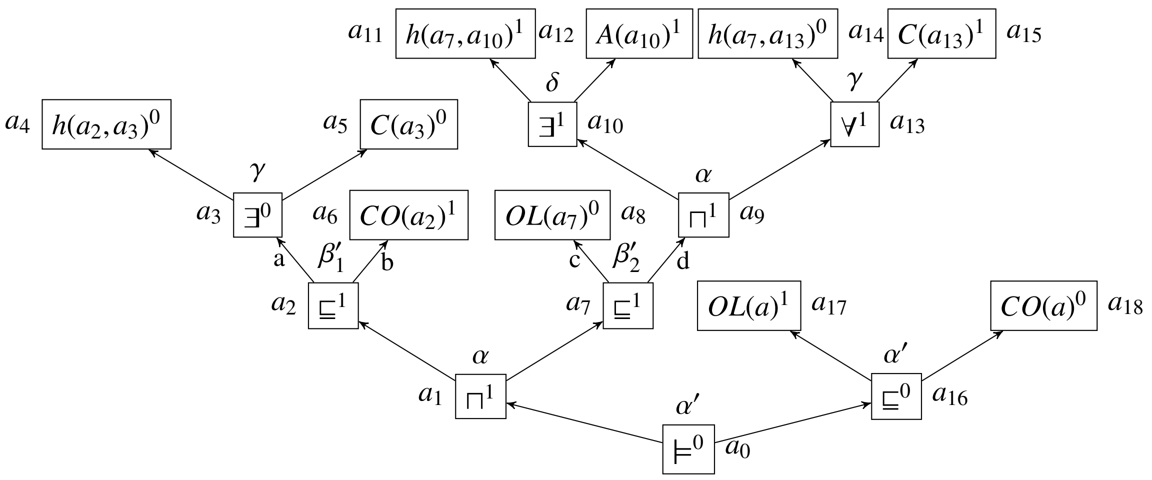

a_{11} $h(a_7,a_{10})^1$ a_{12} $A(a_{10})^1$ $h(a_7,a_{13})^0$ a_{14} $C(a_{13})^1$ a_{15}
δ γ
a_4 $h(a_2,a_3)^0$ a_5 $C(a_3)^0$ $\exists^1$ a_{10} $\forall^1$ a_{13}
γ α
a_3 $\exists^0$ a_6 $CO(a_2)^1$ $OL(a_7)^0$ a_8 $\sqcap^1$ a_9
a β'_1 b c β'_2 d
a_2 $\sqsubseteq^1$ a_7 $\sqsubseteq^1$ $OL(a)^1$ a_{17} $CO(a)^0$ a_{18}
α α'
a_1 $\sqcap^1$ α' $\sqsubseteq^0$ a_{16}
$\models^0$ a_0

Figure 7: Formula Tree for F_1 with labels, polarities and types.

For a given formula A, A', B, B', Γ and Δ are used to denote the sets of node positions of type α, α', β, β', γ, and δ, respectively.

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

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

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

Definition 16. ***(Substitution of positions*** $\sigma_{\beta'}$***).*** *It replaces positions of type β', γ, δ for instances or positions of type β'. Positions of the nodes of type β', γ and δ, 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 β'. 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 β' 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*** σ_{Final}***).*** *It is a combination of σ_δ and $\sigma_{\beta'}$. A σ_{Final}* ***substitution*** *consists of a substitution σ_δ and a substitution $\sigma_{\beta'}$, where $\sigma_{Final} := \sigma_\delta \cup \sigma_{\beta'}$.*

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

Definition 18. ***(Connection,*** σ_{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 σ_{Final}, the connection is a σ_{Final}-****complementary connection****.*

Example 12. ***(Connection,*** σ_{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 $\sigma_{Final} = \{a_2/a_7, a_3/a_{10}\}$.*

Definition 19. ***(Tree Ordering*** $\prec$***).*** *The* ***tree ordering*** $\prec$ *of an F formula is the partial ordering of the nodes positions in the tree formula. $\prec$ 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: $a_7 \prec a_9 \prec a_{13} \prec a_{15}$ and $a_0 \prec a_1 \prec a_2 \prec a_3$.*

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

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

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

Definition 21. ***(*** σ_{Final} ***Admissible Substitution).*** *An σ_{Final}* ***Substitution is admissible*** *if the reduction order $\lhd$ 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 $\sqcap^1$ the rule is l$\sqcap$, while for node $\sqcap^0$ it is r$\sqcap$. 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.

Not preceded						Preceded			
Type α	Rule	Type β	Rule	Type δ	Rule	Type α	Rule	Type β	Rule
$\sqcap^1$	l$\sqcap$	$\sqcap^0$	r$\sqcap$	$\forall^0$	r$\forall$	$\neg^1$	r$\neg\neg$	$\sqcap^0$	l$\neg\sqcap$
$\sqcup^0$	r$\sqcup$	$\sqcup^1$	l$\sqcup$	$\exists^1$	l$\exists$	$\neg^0$	l$\neg\neg$	$\sqcup^1$	r$\neg\sqcup$
$\neg^1$	$\varnothing$					$\sqcap^1$	r$\neg\sqcap$		
$\neg^0$	$\varnothing$					$\sqcup^0$	l$\neg\sqcup$		
Type α'	Rule	Type β'	Rule	Type γ	Rule			Type δ	Rule
$\sqsubseteq^0$	$\varnothing$	$\sqsubseteq^1$	Cut	$\forall^1$	$\varnothing$			$\forall^0$	l$\neg\forall$
$\models^0$	$\varnothing$			$\exists^0$	$\varnothing$			$\exists^1$	r$\neg\exists$

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.

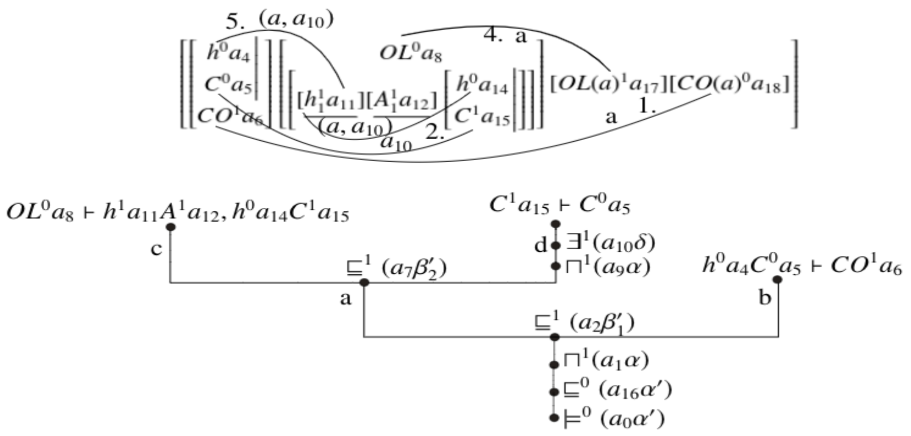

5. (a, a_{10})
4. a
OL^0a_8
h^0a_4
C^0a_5
CO^1a_6
$[h^1_1a_{11}][A^1_1a_{12}]$
(a, a_{10})
a_{10}
2.
h^0a_{14}
C^1a_{15}
$[OL(a)^1a_{17}][CO(a)^0a_{18}]$
a
1.
$OL^0a_8 \vdash h^1a_{11}A^1a_{12}, h^0a_{14}C^1a_{15}$
c
$\sqsubseteq^1\ (a_7\beta'_2)$
$C^1a_{15} \vdash C^0a_5$
d
$\exists^1(a_{10}\delta)$
$\sqcap^1(a_9\alpha)$
$h^0a_4C^0a_5 \vdash CO^1a_6$
a
b
$\sqsubseteq^1\ (a_2\beta'_1)$
$\sqcap^1(a_1\alpha)$
$\sqsubseteq^0\ (a_{16}\alpha')$
$\models^0\ (a_0\alpha')$

Figure 8: Steps representation in the connection proof/sequent $\mathcal{ALC}$ for F_1.

- **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 $\lhd$, 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 β and β'. 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 4. 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 $\sqsubset_\sigma$ between the nodes of that path and there are two tree orderings given by $a_0 \prec a_{16}$ and $a_0 \prec a_1 \prec a_2$, 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: $a_0 \lhd a_{16} \lhd a_1 \lhd a_2$. Since the node with position a_2 is of type β', 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 β' are associated with the cut rule (see table 3). In the second connection, $\underline{C^0}$, with position a_5 in branch a, is connected to $\underline{C^1}$, 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 $a_7 \prec a_9 \prec a_{13}$ and the relations $a_{10} \sqsubset_\delta a_3$ and $a_{10} \sqsubset_\delta a_{13}$. At the moment it is only possible to reduce the node with position a_7 and then the node with position a_9, that is, $a_7 \lhd a_9$. Since the node with position a_7 is of type β', 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 $\underline{h^0}$, with position a_{14}, is connected to $\underline{h^1}$, 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 $a_{10} \sqsubset_\delta a_{13}$ and $a_{10} \sqsubset_\delta a_3$, 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 $OL^0 \vdash h^1A^1, h^0C^1$, due to the cut rule. This connection is complementary under $\sigma_{\beta'} = \{a_7/a\}$. On the fifth and last connection, which connects $\underline{h^0}$ to $\underline{h^1}$, 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_7/a were $\sigma_{\beta'}$ previous substitutions. All connections are complementary under a substitution σ_{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 $\mathcal{ALC}$ 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 α'. Rule l$\sqcap$ is applied. Then, the a_2 position node, with type β', reduced by means of the cut rule on the query α, 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 β', and its reduction

Table 4: Relation between connections, substitutions and orderings

N	Nodes	σ_δ	$\sigma_{\beta'}$	$\sqsubset_\delta$	$\lhd$
1	$CO(a_2)^1a_6, CO(a)^0a_{18}$		a_2/a		$a_0 \lhd a_{16} \lhd a_1 \lhd a_2$
2	$C(a_3)^0a_5, C(a_{13})^1a_{15}$	a_{13}/a_{10}, a_3/a_{10}		$a_{10} \sqsubset_\delta a_3$, $a_{10} \sqsubset_\delta a_{13}$	$a_7 \lhd a_9$
3	$h(a_7,a_{13})^0a_{14}, h(a_7,a_{10})^1a_{11}$	a_{13}/a_{10}			a_{10}
4	$OL(a_7)^0a_8, OL(a)^1a_{17}$		a_7/a		
5	$h(a_2,a_3)^0a_4, h(a_7,a_{10})^1a_{11}$	a_3/a_{10}	a_2/a		
$\sigma_{Final} = a_2/a,\ a_{13}/a_{10},\ a_3/a_{10},\ a_7/a$				$a_0 \lhd a_{16} \lhd a_1 \lhd a_2 \lhd a_7 \lhd a_9 \lhd a_{10}$	

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 $l\exists$. This ends the F_1 sequent proof, as shown in figure 9:

$$\dfrac{\dfrac{OL \vdash \exists h.A \sqcap \forall h.C \qquad \dfrac{\dfrac{\dfrac{}{A,C \vdash C}\,=}{\exists h.A, \forall h.C \vdash \exists h.C}\,\mathrm{l}\exists}{\exists h.A \sqcap \forall h.C \vdash \exists h.C}\,\mathrm{l}\sqcap}{OL \vdash \exists h.C}\,\mathrm{cut} \qquad \exists h.C \vdash CO}{(\exists h.C \vdash CO,\ OL \vdash \exists h.A \sqcap \forall h.C) \vdash (OL \vdash CO)}\,\mathrm{cut}$$

$$\dfrac{(\exists h.C \vdash CO,\ OL \vdash \exists h.A \sqcap \forall h.C) \vdash (OL \vdash CO)}{\big(((\exists h.C \vdash CO) \sqcap (OL \vdash \exists h.A \sqcap \forall h.C)) \vdash (OL \vdash CO)\big)}\,\mathrm{l}\sqcap$$

Figure 9: Complete proof in $\mathcal{ALC}$ sequents for F_1.

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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$ connections proofs into more readable proofs. The approach consists in transforming these proofs into proofs in the $\mathcal{ALC}$-Sequent Calculus [4]. Hence,

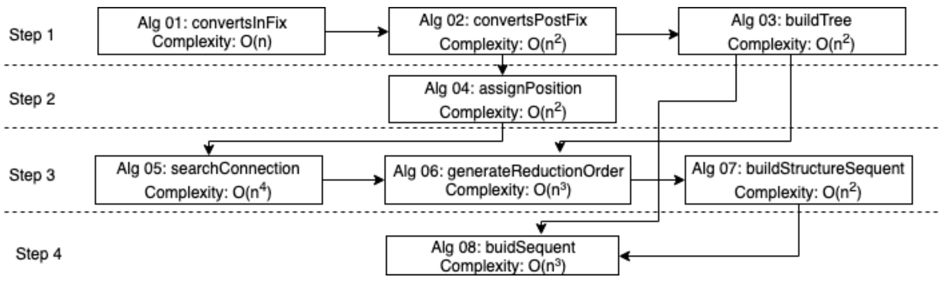

Step 1
Alg 01: convertsInFix Complexity: O(n)
Alg 02: convertsPostFix Complexity: O(n²)
Alg 03: buildTree Complexity: O(n²)
Step 2
Alg 04: assignPosition Complexity: O(n²)
Step 3
Alg 05: searchConnection Complexity: O(n⁴)
Alg 06: generateReductionOrder Complexity: O(n³)
Alg 07: buildStructureSequent Complexity: O(n²)
Step 4
Alg 08: buidSequent Complexity: O(n³)

Figure 10: Overview of the main algorithms' order.

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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$ 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 $\mathcal{ALC}$*. 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 $\mathcal{ALC}$ 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.