EPiC Series in Computing

Volume 46, 2017, Pages 200–211

LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning

RACCOON: A Connection Reasoner for the Description Logic $\mathcal{ALC}$

Dimas Melo Filho[1], Fred Freitas[1] and Jens Otten[2]

[1] Informatics Center, Federal University of Pernambuco (CIn - UFPE), Brazil

[2] Department of Informatics, University of Oslo, Norway

{dldmf, fred}@cin.ufpe.br, jeotten@ifi.uio.no

Abstract

In this paper, we introduce RACCOON, a reasoner based on the connection calculus $\mathcal{ALC}\theta$-CM for the description logic $\mathcal{ALC}$. We describe the calculus, and present details of RACCOON's implementation. Currently, RACCOON carries out only consistency checks, and can be run online; its code is also publicly available. Besides, results of a comparison among RACCOON and other reasoners on the ORE 2014 and 2015 competition problems with $\mathcal{ALC}$ expressivity are shown and discussed.

1 Introduction

Description Logics (DLs) (Baader et al, 2003) are a set of knowledge representation formalisms, which received strong interest in the recent years, particularly after the Semantic Web inception. They constitute the basis for the W3C standard Web Ontology Language (OWL). DL reasoning has also deserved plenty of attention from practitioners from the field of automated reasoning.

Surprisingly, DL reasoners based on the well-known connection method (Bibel, 1993) did not show up in this scenario so far, given the goal-directed search it implements and the fact that they have successfully been adapted to some popular non-classical logics, such as first-order intuitionistic and modal logics (Otten 2008, 2014). These distinctive features led us to the idea of proposing a connection method and its implementation specially tailored to infer over DL Semantic Web ontologies.

In this paper, we present RACCOON (Reasoner based on the Connection Calculus Over ONtologies), an inference engine for the $\mathcal{ALC}$ description logic fragment. It is capable of parsing and reasoning over OWL 2 $\mathcal{ALC}$ ontologies, translating them into a functional syntax and then to a RACCOON internal format. The reasoner was developed using the C++11 standard for performance and portability. It is based on the DL connection method $\mathcal{ALC}\theta$-CM (Freitas and Otten, 2016).

The paper is organized as follows. Section 2 describes the connection method implemented in our reasoner. Section 3 presents details of the system's architecture, internal data structures and pseudo-code. Section 4 shows and discusses performance results. Section 5 concludes the article with a summary and outlook on ongoing and future work.

2 The Description Logic $\mathcal{ALC}$ and the $\mathcal{ALC}\,\theta$-CM calculus

The RACCOON system currently supports the well-known DL $\mathcal{ALC}$, whose concepts are formed according to the following syntax rule: $C ::= A \mid \top \mid \bot \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall r.C \mid \exists r.C$, where A ranges over concept names, r over role names, and C, D over concepts. Two more restricted DLs, $\mathcal{ALE}$ and $\mathcal{AL}$ are also used in this paper: $\mathcal{ALE}$ is equal to $\mathcal{ALC}$ excluding disjunctions ($C \sqcup D$), and complements ($\neg C$) apply only to concept names ($\neg A$); $\mathcal{AL}$ is equal to $\mathcal{ALE}$, replacing $\exists r.C$ by $\exists r.\top$.

An $\mathcal{AL}/\mathcal{E}/\mathcal{C}$ ontology $\mathcal{O}$ is an ordered pair ($\mathcal{T}$,$\mathcal{A}$), where the TBox $\mathcal{T}$ is a set of basic axioms in one of the two forms $C \sqsubseteq D$ or $C \equiv D$, and an ABox $\mathcal{A}$ w.r.t. a TBox $\mathcal{T}$ is a set of assertions of two types: (i) a *concept assertion* is a statement of the form $C(a)$, and (ii) a *role assertion* $r(a,b)$, where a,b are individuals. An $\mathcal{ALC}$ formula is either an axiom or an assertion. The semantics of concepts and ontologies is defined in the usual way - see, e.g., (Baader et al, 2013).

2.1 $\mathcal{ALC}$ Matrix Representation

The Connection Method (CM) (Bibel, 1993) deduces if an ontology $\mathcal{O}$ entails a formula α ($\mathcal{O} \models \alpha$) by checking $\mathcal{O} \rightarrow \alpha$, i.e., if the formula $C_1 \wedge ... \wedge C_n \rightarrow \alpha$ is valid; this holds iff $\neg\mathcal{O} \vee \alpha$ is a tautology (in disjunctive normal form). The effects for the conversion of the formulae to DNF are: (i) axioms of the form $E \sqsubseteq D$, since negated, translate into $E \wedge \neg D$; (ii) ABox assertions are negated; (iii) the consequent α is not negated. Below, we present definitions concerning $\mathcal{ALC}$ ontologies' representation as matrices for the ($\mathcal{ALC}\,\theta$-)CM.

Definition 1 ($\mathcal{ALC}$ literal, DNF, clause, (graphical) matrix). *$\mathcal{ALC}$ Literals* are atomic concepts or roles, possibly negated. An $\mathcal{ALC}$ formula in *disjunctive normal form (DNF)* is a disjunction of conjunctions (like $C_1 \vee ... \vee C_n$), where each *clause* C_i has the form $L_1 \wedge ... \wedge L_m$ and each L_i is a literal. An *($\mathcal{ALC}$) matrix* is a set $\{C_1, ..., C_n\}$, where each C_i has the form $\{L_1, ..., L_m\}$. Literals involved in an universal or existential restriction ($\forall r.C$ or $\exists r.C$) are underlined. When a restriction involves more than one clause, its literals are indexed with a same new column index number at their top. In a *graphical matrix*, clauses are columns, restrictions with indices are horizontal lines, while restrictions without indices are vertical lines (see Examples 2, 3 and 4).

Definition 2 (Impurity, pure conjunction/disjunction). *Impurity* in an $\mathcal{ALC}$ formula is a disjunction in a conjunction, or a conjunction in a disjunction. A *pure conjunction* (PC) or *disjunction* (PD) does not contain impurities.

Example 1 (Impurity, pure conjunction/disjunction). (a) $\exists r.A$ and $\bigwedge_{i=1}^{n} A_i$ are pure conjunctions, i.e., A and each A_i are also PCs. (b) $\forall r.\left(D_0 \sqcup ... \sqcup D_n \sqcup (C_0 \sqcap ... \sqcap C_m) \sqcup \left(A_0 \sqcap ... \sqcap A_p\right)\right)$ is not a pure disjunction, as it contains two impurities: $(C_0 \sqcap ... \sqcap C_m)$ and $\left(A_0 \sqcap ... \sqcap A_p\right)$.

Definition 3 (Two-lined disjunctive normal form). An $\mathcal{ALC}$ axiom is in *two-lined DNF* iff it is in DNF and in one of the normal forms (NFs): (i) $\hat{E} \sqsubseteq \check{D}$; (ii) $E \sqsubseteq \hat{E}$; (iii) $\check{D} \sqsubseteq E$, where E is a concept name[*], $\hat{E}$ *is* a pure conjunction, and $\check{D}$ is a pure disjunction.

Example 2 (Two-lined disjunctive normal form). The axioms (i) $\hat{E} \sqsubseteq \check{D}$; (ii) $E \sqsubseteq \exists r.\hat{E}$ and (iii) $\forall r.\check{D} \sqsubseteq E$, where $\hat{E} = \bigwedge_{i=1}^{n} C_i$ and $\check{D} = \bigvee_{j=1}^{m} D_j$ are represented in Figure 1. C_i, D_j are $\mathcal{ALC}$ literals.

[*] The symbols E and $\hat{E}$ were chosen here to designate a concept name and a pure conjunction rather than the usual C and $\hat{C}$, to avoid confusion with clauses, that are also denoted by C.

$$i)\ 1NF\text{:}\begin{bmatrix} C_1 \\ \vdots \\ C_n \\ \neg D_1 \\ \vdots \\ \neg D_m \end{bmatrix} \qquad ii)\ 2NF\text{:}\begin{bmatrix} E & \cdots & \cdots & E \\ \neg r & \neg C_1 & \cdots & \neg C_n \end{bmatrix} \qquad iii)\ 3NF\text{:}\begin{bmatrix} \neg r & D_1 & \cdots & D_m \\ \neg E & \cdots & \cdots & \neg E \end{bmatrix}$$

Figure 1: Examples of the three two-lined normal forms' representations in $\mathcal{ALC}$.

Example 3 (Two-lined DNF). Table 1 shows examples of quantification restrictions. Vertical lines represent existential restrictions ($\exists r.C$), horizontal lines represent universal restrictions ($\forall r.C$) on the left-hand side axiom's sub-formula or the opposite on the right-hand side. Note also that, if written in first-order logic (FOL), Skolem functions should appear in the two last NFs in Table 1 (e.g., *¬r(x,f(x))* would replace $\exists y$ ...*¬r(x,y)*).

Axiom	Matrix	Negated FOL mapping
$\exists r.\hat{E} \sqsubseteq \forall s.\check{D}$ with $\hat{E}$ a pure conjunction, $\check{D}$ a pure disjunction	$\left[\begin{array}{c\|} r \\ E_1 \\ \vdots \\ E_n \\ s \\ \neg D_1 \\ \vdots \\ \neg D_m \end{array}\right]$	$\exists x \exists y \exists z$ $(r(x,y) \wedge$ $E_1(y) \wedge ... \wedge E_n(y)$ $\wedge$ $(s(x,z) \wedge$ $\neg D_1(z) \wedge ... \wedge \neg D_m(z))$
$A \sqsubseteq \exists r.\hat{E}$ A is a concept name, $\hat{E}$ as above	$\begin{bmatrix} A & \cdots & \cdots & A \\ \neg r & \neg E_1 & \cdots & \neg E_n \end{bmatrix}$	$\forall x \forall y((A(x) \wedge \neg r(x,y))$ $\vee (A(x) \wedge \neg E_1(y))$ $\vee ... \vee (A(x) \wedge \neg E_n(y)))$
$\forall r.\check{D} \sqsubseteq A$ $A, \check{D}$ as above	$\begin{bmatrix} \neg r & D_1 & \cdots & D_m \\ \neg A & \cdots & \cdots & \neg A \end{bmatrix}$	$\forall x \exists y$ $(\neg r(x,y) \wedge \neg A(x)) \vee$ $(D_1(y) \wedge \neg A(x)) \vee ... \vee$ $(D_m(y) \wedge \neg A(x))$

Table 1: Examples of quantification restrictions.

Remark 1 (Two-lined DNF). Relying on these normal forms saves memory by avoiding redundancies in the matrix. To reach these forms, new symbols may be introduced (see Example 4). These symbols are neither allowed to occur in the original formula nor are they introduced in former steps of the normalization procedure. Nevertheless, normalized, "purified" TBoxes are *conservative extensions* (Ghilardi et al, 2006) of their originals, since to every model of the former there is a (sometimes distinct) model of the latter and, thus, validity is preserved.

Definition 4 (Cycle, cyclic/acyclic ontologies and matrices). If *A* and *B* are atomic concepts in an ontology O, *A directly uses B,* if *B* appears in the right-hand side of a subsumption axiom whose left-hand side is *A*. Let the relation *uses* be the transitive closure of *directly uses*. A *cyclic ontology* or *matrix* has a cycle when an atomic concept *uses* itself; otherwise it is *acyclic* (Baader et al. 2003); for instance, $O = \{A \sqsubseteq \exists r.B, B \sqsubseteq \exists s.A\}$ is a cyclic ontology.

Example 4 (Clause, $\mathcal{ALC}$ matrix). $\{Woman \sqcap \exists hasChild.Person \sqsubseteq Mother,\ Mother \sqcap \forall hasChild.Healthy \sqsubseteq Happy,\ Woman(a),\ hasChild(a,b),\ Person(b),\ Healthy(b)\} \models Happy(a)$ reads

$$\left.\begin{array}{l} \forall x \left(Woman(x) \wedge \exists y \left(hasChild(x,y) \wedge Person(y)\right)\right) \rightarrow Mother(x) \\ \forall z \left(Mother(z) \wedge \forall k \left(hasChild(z,k) \rightarrow Healthy(k)\right)\right) \rightarrow Happy(z) \\ Woman(a), \quad hasChild(a,b), \quad Person(b), \quad Healthy(b) \end{array}\right] \models Happy(a)$$

in FOL, and is represented by the following two-lined DNF FOL matrix (f is a Skolem function and A is a new symbol, introduced to transform the formula into the two-lined normal form):

{{*Woman(x), hasChild(x,y), Person(y)*, ¬*Mother(x)*}, {*Mother(z), ¬Happy(x), A*} {¬*A, ¬hasChild(x,f(x))*}, {¬*A, ¬Healthy(f(x))*}, {¬*Woman(a)*}, {¬*hasChild(a,b)*}, {¬*Person(b)*}, {¬*Healthy(b)*}, {*Happy(a)*}} .

The following $\mathcal{ALC}$ matrix represents the same matrix in our DL notation (the column indices mark the clauses' pairs involved in a same restriction):

{{*Woman*, *hasChild, Person,* ¬*Mother*}, {*Mother*, ¬*Happy, A*} {¬*A*, $\underline{\neg hasChild^1}$ }, {¬*A*, $\underline{\neg Healthy^1}$ }, {¬*Woman(a)*}, {¬*hasChild(a,b)*}, {¬*Person(b)*}, {¬*Healthy(b)*}, {*Happy(a)*}}

$$\begin{bmatrix} W & Mo & \neg A & \neg A & \neg W(a) & \neg hC(a,b) & \neg P(b) & \neg He(b) & Ha(a) \\ hC \;\big|\; \neg Ha & \underline{\neg hC} & \underline{He} & & & & & & \\ P \;\big|\; A & & & & & & & & \\ \neg Mo & & & & & & & & \end{bmatrix}$$

Figure 2: Example 4 represented as an $\mathcal{ALC}$ matrix. A is a new symbol, introduced to transform the formula to the two-lined normal form (see Definition 3) . Predicate names are abridged.

2.2 The $\mathcal{ALC}\,\theta$-Connection Calculus ($\mathcal{ALC}\,\theta$-CM)

$\mathcal{ALC}\,\theta$-CM differs from the classical CM in three aspects, just as other DL systems: (i) it rules out variables from the representation, (ii) it replaces Skolem functions and unification by θ-substitutions**,** and, (iii) it employs blocking to assure termination. Below, we present the calculus' formalization. Proofs of soundness, completeness of the calculus can be found in (Freitas and Otten, 2016).

Definition 5 (Path, θ-substitution, (θ-complementary) connection). A path *through a matrix M* contains exactly one literal from each clause in M. A *θ-substitution* assigns each (possibly omitted) variable an individual or another variable (in the whole matrix), *respecting the Skolem condition* (see Definition 6 below). A *connection* is a pair of literals $\{E, \neg E\}$ with the same concept/role name, but different polarities. 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 $\bar{L}$ of a literal L is E if $L = \neg E$, and it is $\neg E$ if $L = E$.

Definition 6 (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. $\tau(x) \stackrel{\text{def}}{=} \{E(x) \in Path\}$, 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, $\forall a \left| \{\underline{E^i(a)} \in Path\} \right| \leq 1$, with a a variable/individual, and i a column index.

Remark 2 (θ-substitution). Simple term unification without Skolem functions is used to calculate θ-substitutions. Similarly to unification, the application of a θ-substitution to a literal is an application to its variables, i.e. $\theta(E) = E\big(\theta(x)\big)$ and $\theta(r) = r\big(\theta(x), \theta(y)\big)$, E is a concept and r is a role. Freitas and Otten (2016) show that using θ-substitution (over formulae without any Skolem functions) under the $\mathcal{ALC}$ θ-CM calculus (see next definition) is equivalent to using unification (with formulae possibly containing Skolem functions) under the classical connection calculus for $\mathcal{ALC}$ formulae in FOL, since $\mathcal{ALC}$ is a subset of FOL and both calculi embedding their respective procedures (θ-substitution, unification) return the same results w.r.t. validity, when given the same inputs.

Definition 7 ($\mathcal{ALC}$ connection calculus). Figure 3 shows the $\mathcal{ALC}$ θ-CM calculus, adapted from the classical CM (Otten, 2010). The calculus' rules are applied bottom-up. The basic structure is the tuple

<*C, M, Path*>, clause C being the open sub-goal, M the matrix corresponding to $O \models \alpha$, $Path$ the *active path*, i.e. the (sub-)path currently checked. A clause C^{μ} is the μ-th copy of clause C, $\mu \in \mathbb{N}$. μ is the indexing function, increased when the *Copy rule* is applied for that clause (the variable x in C^{μ} is denoted by x_{μ}). When the *Copy rule* is used, it has to be followed by an application of the *Extension* or *Reduction rule*, to avoid non-determinism. The *Blocking Condition* is stated as follows: for a new individual x_{μ} in the cycle, its set of concepts should be larger than/distinct from the set of concepts of the previous copied individual, i.e., $\tau(x_{\mu}) \not\subseteq \tau(x_{\mu-1})$ (Schmidt and Tishkovsky, 2007, Freitas and Otten, 2016).

$$Axiom\ (A) \quad \frac{}{\{\}, M, Path}$$

$$Start\ Rule\ (S) \quad \frac{C_1, M, \{\}}{\varepsilon, M, \varepsilon} \quad with\ C_1 \in \alpha$$

$$Reduction\ Rule\ (R) \quad \frac{C, M, Path \cup \{L_2\}}{C \cup \{L_1\}, M, Path \cup \{L_2\}}$$
$$with\ \theta(L_1) = \theta(\overline{L_2})\ \ and\ the\ Skolem\ condition\ holds$$

$$Extension\ Rule\ (E) \quad \frac{C_2 \backslash \{L_2\}, M, Path \cup \{L_1\} \quad C, M, Path}{C \cup \{L_1\}, M, Path}$$
$$with\ C_2 \in M,\ L_2 \in C_2, \theta(L_1) = \theta(\overline{L_2})\ \ and\ the\ Skolem\ condition\ holds$$

$$Copy\ Rule\ (C) \quad \frac{C \cup \{L_1\}, M \cup \{C_2^{\mu}\}, Path}{C \cup \{L_1\}, M, Path}$$
$$with\ C_2^{\mu}\ is\ a\ copy\ of\ C_1, L_2 \in C_2^{\mu}, C_1 \in M, \theta(L_1) = \theta(\overline{L_2})\ and\ the\ blocking\ condition\ holds$$

Figure 3: The connection calculus $\mathcal{ALC}$ θ-CM.

Remark 3 (Copy rule, blocking Skolem condition, $\mathcal{ALC}$ connection calculus). The CM for FOL already copies clauses, using the indexing function μ; however, since it does not need blocking an explicit rule is not needed. Here, the *Copy rule* implements blocking (Baader et al., 2003), portraying the cases when no alternative connection is available and cyclic ontologies occur. It restricts clause copies and the creation of new individuals, thus preventing non-termination due to the presence of an infinite cycle. Note, however, that, instead of working with the original matrix *M*, we may be using *M'*, which is *M* with some clauses copied. The *Skolem condition* avoids the situation where, in classical logic, unification fails for two distinct Skolem functions.

Example 5 ($\mathcal{ALC}$ θ-CM). Figures 4 and 5 show the connection proof of Example 4 by a graphical matrix and the formal calculus. For a step-by-step proof example, see (Freitas and Otten 2016).

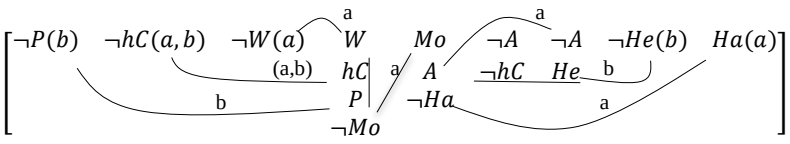

Figure 4: The matrix proof for Example 4. Labels over the arcs indicate individuals involved in the connection. The order of columns was changed to increase readability.

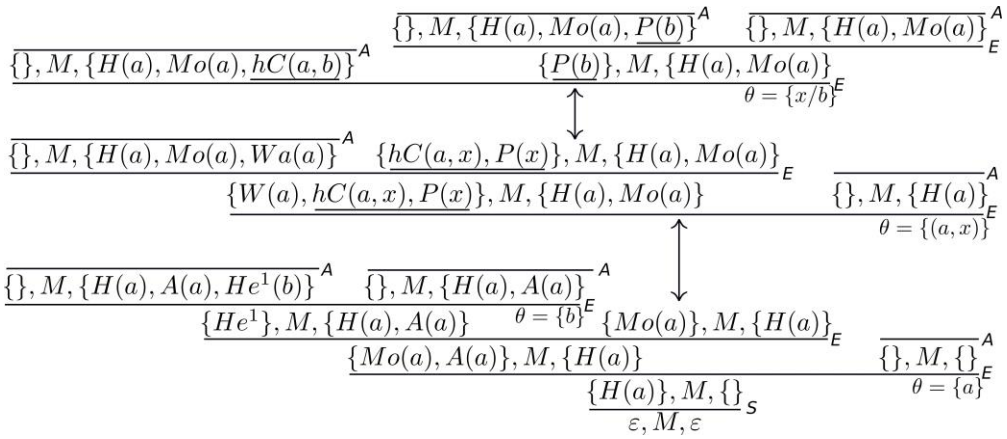

Figure 5: The connection proof using the formal $\mathcal{ALC}$ θ-CM connection calculus. M is an abbreviation for M ={{W, $\underline{hC, P}$, ¬Mo}, {Mo, ¬Ha, A} {¬A, $\underline{\neg hC^1}$}, {¬A, $\underline{\neg He^1}$}, {¬$W(a)$}, {¬$hC(a,b)$}, {¬$P(b)$}, {¬$He(b)$}, {$Ha(a)$}}.

3 RACCOON: A Reasoner based in $\mathcal{ALC}$ θ-CM

We developed RACCOON (ReAsoner based on the Connection Calculus Over ONtologies), a reasoner based on the $\mathcal{ALC}$ θ-CM calculus. It is coded in C++ and is publicly available at http://github.com/dmfilho/raccoon . Ontologies in the OWL functional format written in $\mathcal{ALC}$ or more restricted DLs can be verified for consistency with the reasoner online at http://raccoon.bitroot.com.br/test . RACCOON consists of three modules: Parser, Preprocessor and Reasoner, as displayed in Figure 6.

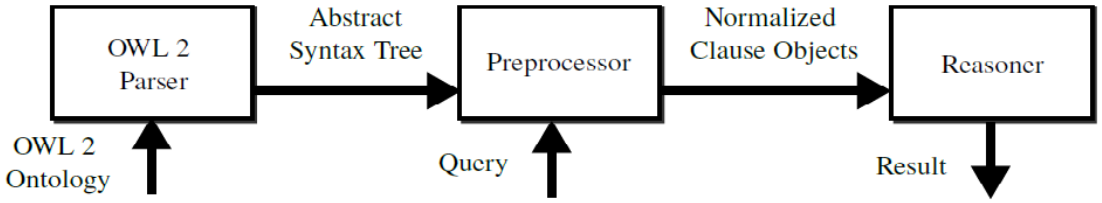

Figure 6: RACCOON's modules, inputs and outputs.

The parser loads the OWL ontology and converts it into an Abstract Syntax Tree which optionally can be reduced by the Preprocessor. The next step, normalization, is applied by the latter to create a set of $\mathcal{ALC}$ DNF two-lined normalized clauses. The normalization results in clauses following the forms described in Definition 3. Therefore, the input for the reasoner is an Abstract Tree in $\mathcal{ALC}$ DNF two-lined normal form as described above, containing normalized clauses.

3.1 Regularity

As for optimizations, we did not implement yet any typical DL optimization, such as backjumping (Baader et al 2013), among others. In the preprocessing phase, clauses with *pure literals*, i.e., the ones without complements in the matrix, are eliminated (Brachman and Levesque, 2004).

The only optimization integrated is regularity (Letz et al, 1994), which is already successfully used in the CM for FOL. It avoids redundant literals by restricting the *Extension rule* to clauses which do not contain any of the literals already present on the active path. For the classical FOL connection method, the regularity condition requires that the active *Path* contains no literals from the extension clause C_2, under unification. For the $\mathcal{ALC}\theta$-CM, the same condition holds, using θ-substitution instead of unification. The *regularity condition* holds only if

$$\nexists L_c \in C_2, L_p \in Path \mid \theta(L_c) = \theta(L_p).$$

Then, the *Extension rule* is restricted to clauses C_2, such that the regularity condition holds.

3.2 Normalization, Data Structures and Indexing

Although normalized, clauses are represented in a form slightly different from the two-lined DNF. The implemented DNF form represents clauses in the normal forms $E \sqsubseteq \hat{E}, \check{D} \sqsubseteq E$ (E is concept name, $\hat{E}$ *is* a pure conjunction, and $\check{D}$ is a pure disjunction), as an one-lined submatrix. This is a small memory optimization in the representation, as, in many cases, it prevents literals to be repeated in the matrix and, thus, connected more than once (such as $\neg A$ in Example 4).

As for indexing, each clause contains a list of instances, where each instance is associated to one of its variables. Both literals and instances are uniquely identified by integers. Each literal has two lists of clauses containing the literal's complements, i.e. clauses to which connections are possible. The *Path* is implemented as a stack.

3.3 The Reasoning Algorithm

RACCOON reasons by applying $\mathcal{ALC}\theta$-CM rules, as displayed in the pseudo-code of Figure 7. For the sake of clarity, the pseudo-code was simplified, i.e., some lower-level steps were omitted.

```
01: func proveLiteral (clause, literalIndex, path) {
02:     if literalIndex >= clause.size()                  \\ check if clause literals exhausted
03:         return true;
04:     if path.contains (literalIndex)                   \\ regularity: if literal is already in the path
05:         return false;
06:     if path.containsNeg (literalIndex)                \\ Reduction rule
07:         return proveLiteral (clause, literalIndex+1, path)   \\ try next literal
08:     path.push (literalIndex);                         \\ put literal on path
09:     for each connection in literal.connections {      \\ try each literal's possible connection
10:         if connection.valid (path) {                  \\ if we prove with the current connection
11:             path.pop ();                              \\ remove literal from path to try next literal
12:             if proveLiteral (clause, literalIndex+1, path)     \\ and try the clause's next literal
13:                 return true;
14:             path.push (literalIndex)                  \\ if failed, push the literal back again
16:         }                                             \\ and try next connection (backtrack)
17:     }
18:     path.pop ();
19:     return false;                                     \\ no connection worked; not proven
20:}
```

Figure 7: RACCOON's simplified pseudo-code.

The *Start Rule* consists of calling the method *proveLiteral* with the first literal from the start clause, the literal with literalIndex of 0 (each clause is a literal array, with indices starting from 0). Any clause from consequent α, not created by the normalization, can be a start clause. Initially, the *Path* is empty.

Since it is a recursive method, the algorithm first checks the base case: if there are no more literals to be tested in the clause (line 2) then the method has proven this part, and the *Axiom rule* is reached (line 3). Next, to speed up processing, regularity is checked, by verifying whether the literal is already in the path (line 4). If this is the case, this search branch is abandoned.

After these tests, RACCOON tries to apply the *Reduction rule*: function *containsNeg* (line 6) verifies if there is a literal negation (the complement) already in the *Path.* If so, the current branch is proven, and the next literal of the clause will be checked.

The last, most expensive and main option to solve the current literal, the *Extension Rule,* is tested against each possible connection within the loop starting at line 8. The literal is put in the *Path*, and, in case the current connection worked, the literal is popped from the *Path* stack, and the next literal will be verified (line 12). If the connection did not work, the literal is reinserted in the *Path* stack for the algorithm to backtrack and try the next possible, available connection, if any.

The Skolem condition validation is performed inside the *valid* method from the connection class, called in line 10. This condition was easy to implement: it suffices to creates a flag for each variable/ individual in the *Path* being checked, denoting if the *Path* already contains an underlined concept for that variable/individual. In this case, a new connection cannot be set.

The *Copy Rule* is also located within the same method. It does not actually copy a clause object into the matrix; instead, it represents the copy by allocating memory for a new set of instances for the clause, when the blocking condition holds. This implementation was inspired by the connection structure calculus (Eder 1992).

A last remark on the algorithm refers to the backtracking possibilities. For connection calculi, the possibilities reside in varying (i) initial clauses, (ii) unifiers and (iii) connections. For $\mathcal{ALC}$ θ-CM, regarding initial clauses, the method *proveLiteral* should be invoked for all clauses of the consequent α, if necessary. As for unifiers, $\mathcal{ALC}$ θ-CM does not include functions; thus, the θ-substitutions are never complex, as they are either variable changes or a substitution of a variable by an individual, that can occur only once in each path. In any case, θ-substitutions never include term decompositions; consequently, it is clearly a one-step procedure, without the need of further backtracking. The third possibility, the backtracking over possible connections, is carried out inside the algorithm. After having exhausted all possibilities without closing all paths, the ontology is deemed consistent, as the algorithm has failed in proving its inconsistency.

4 Experiments

We conducted a practical evaluation of RACCOON and compared it to other reasoners that participated in ORE (Ontology Reasoning Evaluation), the yearly DL reasoner's competition (Bail et al 2014). The other reasoners are Konclude (Steigmiller et al, 2014), the current ORE champion, Hermit (Glimm et al, 2014) and FACT++ (Tsarkov and Horrocks, 2006).

In the experiments, only the ORE consistency task was considered. Each reasoner was tested on all ontologies, which had $\mathcal{AL}$, $\mathcal{ALE}$ and $\mathcal{ALC}$ expressivity from the 2014 and 2015 ORE's datasets. The former contains 1,621 ontologies of this kind, and the latter 401. Tests were run on a machine with 8 GB RAM and Intel core i5 @ 2.3 GHz, with two cores. Only Konclude is multithreaded, i.e., it is the only reasoner that takes advantage of the two cores. Timeout for each problem was set to 250 seconds. Times were measured in microseconds summing up parsing, normalization and reasoning times; in the tables they are presented in seconds. The detailed results from each ontology, as well as consolidated results under many perspectives, are available at goo.gl/V9Ewkv .

Results of the performance of RACCOON and other solvers on the ORE 2014, 2015 problem ontologies are displayed in the next two tables. We first analyzed, for each DL expressivity ($\mathcal{AL}$, $\mathcal{ALE}$

and $\mathcal{ALC}$), the number of solved ontologies that each reasoner solved in the fastest time. These data are presented in Tables 2 and 3.

#Solved Ontologies by Time Interval	$\mathcal{AL}$ – 825 ontologies				$\mathcal{ALE}$ – 740 ontologies				$\mathcal{ALC}$ – 56 ontologies			
	Raccoon	**Hermit**	**Fact++**	**Konclude**	**Raccoon**	**Hermit**	**Fact++**	**Konclude**	**Raccoon**	**Konclude**	**Fact++**	**Hermit**
0 - < 0.1s	592	0	0	577	272	0	0	310	46	0	0	47
0.1 - < 1s	201	143	659	195	194	125	393	285	8	26	51	9
1 - < 10s	13	666	157	44	87	574	323	127	1	30	5	0
10 - < 100s	7	11	7	9	30	26	4	18	1	0	0	0
100 - < 250s	0	1	0	0	3	8	3	0	0	0	0	0
Timeouts	**12**	4	2	0	**154**	7	17	0	0	0	0	0
Fastest	**683**	0	0	142	363	0	6	**371**	**45**	0	0	11

Table 2: Results of RACCOON and ORE competitors for consistency on the ORE 2014 ontologies.

#Solved Ontologies by Time Interval	$\mathcal{AL}$ – 167 ontologies				$\mathcal{ALE}$ – 198 ontologies				$\mathcal{ALC}$ – 36 ontologies			
	Raccoon	**Hermit**	**Fact++**	**Konclude**	**Raccoon**	**Hermit**	**Fact++**	**Konclude**	**Raccoon**	**Hermit**	**Fact++**	**Konclude**
0 - < 0.1s	77	0	0	73	55	0	0	60	12	0	0	13
0.1 - < 1s	65	75	91	60	77	52	97	87	12	13	18	17
1 - < 10s	13	83	63	23	40	123	75	32	7	19	14	3
10 - < 100s	7	4	8	11	17	6	4	19	4	4	1	3
100 - < 250s	0	0	0	0	1	10	1	0	0	0	0	0
Timeouts	5	5	5	0	8	7	21	0	1	0	3	0
Fastest	**141**	0	0	26	**143**	0	1	54	**23**	1	0	12

Table 3: Results of RACCOON and ORE competitors for consistency on the ORE 2015 ontologies.

The first observation is the high number of 166 RACCOON timeouts in the ORE 2014 dataset (while Hermit and Fact++ had only 11 and 19, respectively). On the other hand, for the ORE 2015 dataset, RACCOON had far less timeouts than Fact++ (14 against 29), and almost as many as Hermit.

Konclude performed consistently excellent throughout all tests, not exceeding the 250s limit a single time; besides that, RACCOON and Konclude were almost always the fastest solvers in all segments. RACCOON was the fastest in most of the ontologies, in all expressivities and ontology sizes for both datasets (see detailed results at goo.gl/V9Ewkv). RACCOON's good performance is likely due to the efficient parsing, and the few, small optimizations implemented in the reasoner. Furthermore, RACCOON is restricted to $\mathcal{ALC}$, whereas the other solvers can also deal with larger DL fragments.

Tables 4 and 5 exhibit statistics on average times with respect to the number of axioms in the ontologies of ORE 2014 and 2015 baselines.

For the $\mathcal{AL}$ fragment, RACCOON exhibits the lowest average time for all ontology sizes in ORE 2014 and 2015. For $\mathcal{ALE}$ ontologies, RACCOON has average times only marginally higher than those of Koncludeand similar to those of Fact++. In $\mathcal{ALC}$, however, the lack of DL typical optimizations came into play, making RACCOON's average times raise significantly. Another aspect to be noted is that RACCOON performs better for small ontologies, as can be seen in Figure 8. On the other hand, RACCOON is the reasoner whose times increase more rapidly when ontology size increases.

Average Time by #axioms	#ontologies	$\mathcal{AL}$ – 825 ontologies				$\mathcal{ALE}$ – 740 ontologies				$\mathcal{ALC}$ – 56 ontologies			
		Raccoon	Hermit	Fact++	Konclude	Raccoon	Hermit	Fact++	Konclude	Raccoon	Hermit	Fact++	Konclude
0-10^3 axs	837	0.03	2.75	0.49	0.04	1.90	2.36	0.57	0.05	0.03	1.70	0.49	0.04
10^3-10^4	315	0.14	4.22	0.82	0.17	2.87	2.60	0.84	0.17	8.22	2.10	0.75	0.16
10^4-10^5	427	0.58	3.53	1.64	0.86	1.55	4.99	3.97	0.99	1.66	3.49	1.19	0.38
10^5-10^6	39	14.80	25.58	20.91	27.99	15.65	66.63	16.65	15.32	-	-	-	-
$\geq 10^6$axs	3	N/A	51.94	N/A	40.66	41.44	53.67	53.73	58.73	-	-	-	-
Average	-	0.30	3.34	0.94	0.55	2.79	5.60	2.45	1.21	0.79	1.92	0.58	0.08

Table 4: Comparative evaluation of RACCOON and ORE competitors for the task of consistency on the ORE 2014 ontologies. "N/A" indicates that all ontologies of the segment timed out for the reasoner.

Average Time by #axioms	#ontologies	$\mathcal{AL}$ – 167 ontologies				$\mathcal{ALE}$ – 198 ontologies				$\mathcal{ALC}$ – 36 ontologies			
		Raccoon	Hermit	Fact++	Konclude	Raccoon	Hermit	Fact++	Konclude	Raccoon	Hermit	Fact++	Konclude
0-10^3 axs	119	0.03	0.80	0.47	0.04	1.53	0.91	0.57	0.06	0.50	0.82	0.49	0.04
10^3-10^4	96	0.12	1.09	0.73	0.15	7.67	1.25	0.77	0.22	10.94	1.17	0.82	0.19
10^4-10^5	136	0.56	2.16	1.61	0.84	0.89	3.97	5.48	1.24	1.49	1.75	1.33	0.51
10^5-10^6	48	11.43	13.86	26.30	21.45	10.07	84.00	9.26	16.81	11.42	15.74	13.89	15.19
$\geq 10^6$axs	2	N/A	41.80	N/A	38.72	-	-	-	-	25.71	41.67	N/A	33.48
Average	-	1.28	2.39	2.83	2.49	4.20	11.60	2.84	2.94	3.76	4.03	2.09	2.86

Table 5: Comparative evaluation of RACCOON and ORE competitors for the task of consistency on the ORE 2015 ontologies. "N/A" indicates that all ontologies of the segment timed out for the reasoner.

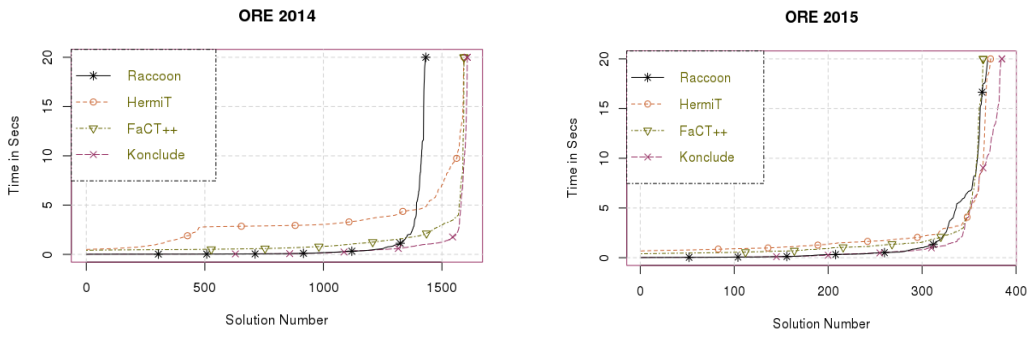

Figure 8: Comparison of time complexity behavior.

Regarding timeouts, RACCOON's performance was hampered by ontologies with higher complexity, in which cycles occur inside other cycles. Such structures were often found in the $\mathcal{ALE}$ segment of the 2014 ontologies; it seems that many of the ontologies were built by altering others from the same dataset, given their similarity. Such drawback can be addressed in one of the three ways: (i) by applying an $\mathcal{ALC}$ non-clausal calculus, which is being defined by our research group, based on the non-clausal calculus for classical FOL (Otten 2016) (ii) by using a FACTOR reduction, (Bibel, 1993, 56-58), or (iii) by trying to integrate into our connection reasoner some DL tableaux optimizations

implemented in the other reasoners that help solve these cases. Timeouts by expressivity and ontology sizes can be seen in Table 6.

Timeouts	ORE 2014									ORE 2015								
		Expressivity			Ontology Size						Expressivity			Ontology Size				
	Total	$\mathcal{AL}$	$\mathcal{ALE}$	$\mathcal{ALC}$	$\leq 10^3$	10^3-10^4	10^4-10^5	10^5–10^6	$>10^6$	Total	$\mathcal{AL}$	$\mathcal{ALE}$	$\mathcal{ALC}$	$\leq 10^3$	10^3-10^4	10^4-10^5	10^5–10^6	$>10^6$
Raccoon	**166**	12	**154**	0	1	9	**152**	2	2	14	5	8	1	1	5	6	1	1
Hermit	11	4	7	0	0	0	0	10	1	12	5	7	0	0	0	0	12	0
Fact++	19	2	17	0	0	1	2	15	1	**29**	5	**21**	3	0	0	4	**23**	2

Table 6: Reasoners' timeouts for consistency of both datasets by DL expressivity and ontology size.

5 Conclusions, Ongoing and Future Work

In the current paper, a new reasoner, RACCOON is presented. It was implemented based on the new calculus $\mathcal{ALC}\theta$-CM, and the first tests over $\mathcal{AL}$, $\mathcal{ALE}$ and $\mathcal{ALC}$ ontologies on the task of consistency were executed. Comparing its performance with other ORE reasoners has demonstrated that our implementation can be competitive. The high number of timeouts displayed in the ORE 2014 dataset will be tackled in a future version of RACCOON.

As for practical future work, in the short term, RACCOON will be improved to cope with the other two ORE tasks, realization and subsumption. Another important enhancement consists in finding good optimizations for treating ABoxes. In the medium term, other expressive constructs will be tackled, given good theoretical solutions are found to be encompassed by our current connection calculus.

Other relevant future work, that involves theoretical investigations, consists in enlarging the expressiveness for the calculus and the reasoner. For instance, we already developed the theoretical foundations for $\mathcal{ALC}\theta$-CM to deal with (in)equality, and therefore, with DL cardinality restrictions ($\geq / \leq n\, r$ for $\mathcal{ALCN}$ and $\geq / \leq n\, r.C$ for $\mathcal{SHQ}$). Hence, soon RACCOON will be capable of parsing and reasoning with $\mathcal{EL}$++ ontologies thus enabling an ORE participation, once subsumption and realization are implemented. Nonetheless, the equality issue is a sensitive one in terms of performance for connection calculi implementations, given the known drawback that substitutions are applied to the whole matrix.

Besides cardinality, in the medium term, other expressive constructs will be tackled for reasoning, once appropriate matrix representations have been developed. Indeed, an OWL 2 complete translation to matrices has been set (available at http://dl.adrianomelo.com/cm-rules.pdf). But reasoning with all this expressiveness requires solving subtle problems, which we have not addressed yet. For instance, creating good theoretical and practical solutions for our framework to encompass techniques such as sophisticated blocking schemes to emulate dynamic and double blocking for DL constructs like inverse roles (Horrocks and Sattler, 1999) and dealing with nominals, among other DL complex reasoning issues, are still on our agenda.

Acknowledgements. The authors would like to thank Pernambuco´s state sponsoring agency FACEPE for a grant to support the stay of Jens Otten at UFPE, and the anonymous reviewers for their comments. We also thank UFPE and FACEPE for funding the article presentation at the conference, and Renata Melo for helping preparing the final version.

References

Baader, F., Calvanese, D. McGuinness, D., Nardi, D., Patel-Schneider, P. (Eds.) (2003) *The Description Logic Handbook*. Cambridge University Press.

Bail, S., Glimm, B., Jiménez-Ruiz, E., Matentzoglu, N., Parsia, B., Steigmiller, A. (Eds.) (2014) *Informal Proceedings of the 3rd International .Workshop OWL Reasoner Evaluation*. CEUR Workshop Proceedings 1207.

Bibel, W. (1993) *Deduction*: *Automated Logic*. Academic Press.

Brachman, R., Levesque, H. (2004) *Knowledge representation and Reasoning*. Morgan Kaufmann.

Eder, E. (1992) *Relative Complexities of First Order Calculi.* Vieweg.

Freitas, F., Otten, J. (2016) *A Connection Calculus for the Description Logic $\mathcal{ALC}$.* Proceedings of the Canadian Conference on Artificial Intelligence (AI 2016), LNAI 9673, pp. 243-256, Springer.

Ghilardi, S, Lutz, C., Wolter, F. (2006) *Did I damage my ontology: A Case of Conservative Extensions of Description Logics*. Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR), AAAI Press.

Glimm, B., Horrocks, I., Motik, B., Stoilos, G.,Wang, Z. (2014) *HermiT: An OWL 2 Reasoner*. Journal of Automated Reasoning 53(3): 245-269.

Horrocks, I.,Sattler, U. (1999) *A Description Logic with Transitive and Inverse Roles and Role Hierarchies*. Journal of Logic and Computation 9(3):385–410.

Letz, R., Mayr, K., Goller. C. (1994) *Controlled Integration of the Cut Rule into Connection Tableau Calculi.* Journal of Automated Reasoning 13:297-337.

Otten, J. (2008) *leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic*. Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2008), LNAI 5195, pp. 283-291, Springer.

Otten, J. (2010) *Restricting backtracking in connection calculi*. AI Communications 23(2-3):159-182.

Otten, J. (2014) *MleanCoP*: *A Connection Prover for First-Order Modal Logic.* Proceedings of the Intl. Joint Conference on Automated Reasoning (IJCAR 2014), LNAI 8562, pp. 269-276, Springer.

Otten, J. (2016) *nanoCoP*: *A non-clausal Connection Prover.* Proceedings of the International Joint Conference on Automated Reasoning (IJCAR 2016), LNAI 9706, pp. 302-312, Springer.

Schmidt, R., Tishkovsky, D. (2007) *Analysis of Blocking Mechanisms for Description Logics*. Proceedings of the 14th Workshop on Automated Reasoning (ARW'07), pp. 51-52.

Sirin, E., Grau, B.C., Parsia, B. (2006) *From wine to water: optimizing description logic reasoning for nominals*. International Conference on the Principles of Knowledge Representation and Reasoning (KR-2006), AAAI Press.

Steigmiller, A. Liebig, T., Glimm, B. (2014) *Konclude: System Description*. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 27(1):78-85.

Tsarkov, D., Horrocks, I. *FaCT++ Description Logic Reasoner: System Description*. Proc. of the Int. Joint Conference on Automated Reasoning (IJCAR 2006), LNAI 4130, pp. 292-297, Springer.

Tsarkov, D., Riazanov, A., Bechhofer, S., Horrocks, I. (2004) *Using Vampire to Reason with OWL*. Proc. of the Int. Semantic Web Conference (ISWC 2004), LNCS 3298, pp. 471-485, Springer.