International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 5, Number 1, pages 61–85, January 2015

Handling Non-determinism with Description Logics using a Fork/Join Approach

Jocelyne Faddoul
58 Peachtree Hill
Dartmouth, NS, B2W 0H8, Canada

Wendy MacCaull
St. Francis Xavier University
Antigonish, NS, B2G 2W5, Canada

Received: August 2, 2014
Revised: November 5, 2014
Accepted: December 3, 2014
Communicated by Akihiro Fujiwara

Abstract

The increasing use of Ontologies, formulated using expressive Description Logics, for time sensitive applications necessitates the development of fast (near realtime) reasoning tools. Multicore processors are nowadays widespread across desktop, laptop, server, and even smartphone and tablets devices. The rise of such powerful execution environments calls for new parallel and distributed Description Logics (DLs) reasoning algorithms. Many sophisticated optimizations have been explored and have considerably enhanced DL reasoning with light ontologies. Non-determinism remains a main source of complexity for implemented systems handling ontologies relying on more expressive logics.

In this work, we explore handling non-determinism with DL languages enabling qualified cardinality restrictions. We implement a fork/join parallel framework into our tableau-based algebraic reasoner, which handles qualified cardinality restrictions and nominals using in-equation solving. The preliminary results are encouraging and show that using a parallel framework with algebraic reasoning is worth investigating and more promising than parallelizing standard tableau-based reasoning.

Keywords: Description Logics, Non-determinism, Fork/Join

1 Motivation

Simulating a human reasoning process requires that an extensive knowledge about the world be represented and stored in a knowledge base. Among the things that need to be represented are: objects, properties, and relations between objects. A complete representation of "what exists" in a given domain forms an Ontology. Ontologies have now been adopted by many disciplines, such as biology and e-science, as part of their integration into the Semantic Web, which is defined as a "web of data" allowing machines to understand the meaning of and process the information on the World Wide Web. Applications of the semantic web are numerous, wide ranging and have tremendous potential for adding value in a vast array of situations which can take advantage of intelligence, i.e.,

the capacity to reason over knowledge stored in a knowledge base such as an ontology. However, if the application is time sensitive, the time required for reasoning can be prohibitive.

Description logics (DL) have gained a lot of attention in the research community as they provide a logical foundation for the Web Ontology Language (OWL), defined by the World Wide Web Consortium (W3C) as a standard for representing semantic links and knowledge on the Semantic Web. An important feature of DL is that they allow one to represent knowledge not only for the sake of representing it, but also to reason about it and make implicit knowledge explicit through the use of reasoning services. For instance, consider a medical Ontology where Penicillins is described as a sub-class of Antibiotics, and a patient X is asserted as taking Penicillins. DL reasoning services infer that X is taking Antibiotics. Standard DL inference services, e.g., TBox classification, concept satisfiability checking, instance checking, etc., have been extended with query answering in order to extract information and drive applications such as web services and workflow management systems [27]. The effectiveness of these reasoning services (correctness and speed) is subject to the complexity of the Ontology (e.g, size, expressivity) used and the implementation of the reasoning strategy. For many applications (e.g., associated with health services delivery or large scale mergence protocols) these services are time sensitive, but require time consuming reasoning over complex and often large ontologies.

There has been a great deal of research into optimizing DL reasoning strategies and in carving out fragments over which reasoning can proceed at a reasonable pace — but reasoning using these strategies or over these fragments often does not scale to allow the use of large ontologies. Reasoning for time sensitive tasks (e.g., those required in a clinical decision support system) still requires severe restrictions on the expressivity, the complexity and/or the size of the ontology used. The expressivity of the domain knowledge is often sacrificed in order to meet practical reasoning performance. For example, the Lipid ontology used in [6] relies heavily on Qualified Cardinality Restrictions: expressions of the form $\geq nR.C$ and $\leq nR.C$. It is expressed using the DL $\mathcal{ALCHIQ}$ using 715 concepts and 46 properties. For example, axiom (1) describes a complex lipid structure, $LC\_Hexaacylaminosugar$, that has exactly 6 primary acyl chain.

$$LC\_Hexaacylaminosugar \sqsubseteq \geq 6 hasProperFat.Primary\_Acyl\_Chain \sqcap \leq 6 hasProperFat.Primary\_Acyl\_Chain \tag{1}$$

The comprehensive classification of Lipids [15] is expressed using the DL $\mathcal{SHQ}$ using 729 concepts and 3 properties. Qualified Cardinality Restrictions (QCRs) are used heavily, and the full Lipids ontology cannot be classified by existing reasoners within hours of CPU time. Hence the recent popularity of lightweight ontologies, i.e., expressed using the extensions of the tractable DL $\mathcal{EL}$. Sacrificing the expressivity of the knowledge modelled is a limiting (and sometimes unacceptable) compromise.

When it comes to handling ontologies that rely on the use of QCRs, algebraic reasoning, to the best of our knowledge, remains the most promising approach. This has been shown in fragments of DL using QCRs [17, 16], inverse roles [8, 32], and nominals [12, 11]. Practical implementations of such algebraic tableau algorithms require carefully chosen optimizations in order to outperform the highly optimized existing state of the art reasoners. Most algebraic tableau-based algorithms proposed so far are double exponential in the worst case; their optimized implementations have been tested on a suite of artificial or often adapted subsets of ontologies. The scalability of the algebraic approach with real world and often large ontologies remains open.

The high performance computing (HPC) paradigm would seem to offer a solution to these problems, but progress using high performance computing methodologies has been challenging and slow [26, 3, 39]. The techniques that have offered speedy solutions in other domains (e.g., for "number crunching" in the physical sciences) do not suffice to crack the time bottleneck of reasoning tasks required for effective use of ontologies. Work is needed to find techniques for this kind of computing. Recently, there has been encouraging results [28, 2, 39, 36, 37]. The work considered so far, considers parallelizing the TBox classification task [2], the Abox querying task [1], or the concept satisfiability checking task [26, 28] using ontologies relying on the least expressive fragments of DLs. Parallelizing algebraic reasoning to allow the handling of large ontologies using number restrictions needs further investigation.

Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Atomic negation	$(\neg A)$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
Conjunction	$(C \sqcap D)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$(C \sqcup D)$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Value restriction	$(\forall R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \forall t \in \Delta^{\mathcal{I}} : \langle s,t\rangle \in R^{\mathcal{I}} \Rightarrow t \in C^{\mathcal{I}}\}$
Existential restriction	$(\exists R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \exists t \in \Delta^{\mathcal{I}} : \langle s,t\rangle \in R^{\mathcal{I}} \text{ and } t \in C^{\mathcal{I}}\}$
Qualified at-least restriction	$(\geq nR.C)$	$\{s \in \Delta^{\mathcal{I}} \mid\mid \{s \mid \langle s,t\rangle \in R^{\mathcal{I}}\} \mid \geq n\}$
Qualified at-most restriction	$(\leq nR.C)$	$\{s \in \Delta^{\mathcal{I}} \mid\mid \{s \mid (\langle s,t\rangle \in R^{\mathcal{I}}\} \mid \leq n\}$

Figure 1: Syntax and Semantics of the DL $\mathcal{ALCQ}$

In this work, we expand our ideas presented in [13, 14] and report a preliminary implementation. Our approach consists of handling non-determinism inherent in tableau-based DL reasoning using a parallel framework. We implement our framework into the prototype tableau-based algebraic reasoner, HARD [9]. Even though tableau-based DL reasoning for less expressive fragments has been explored in a parallel framework [28, 26], algebraic tableau-based DL reasoning has only been explored sequentially. The paper is organized as follows: Section 2 introduces the basics of Description Logics syntax, semantics, and tableau-based reasoning; Section 3 illustrates the non-determinism in tableau-based reasoning, introduces algebraic tableau reasoning, and argues why algebraic reasoning is more suitable for parallelization; Section 4 illustrates the parallel framework used; Section 5 reports on experimental results; Section 6 reviews related work; and Section 7 concludes the work.

2 Preliminaries

Description Logic (DL) [19] is a family of knowledge representation languages used to represent and reason about an application's domain elements. DLs stem from Semantic Networks [33] and Frames [29]. They are distinguished by their terminological orientation, their well defined logic-based semantics, their decidable inference services, and available tools. The parallel framework discussed in this paper has been designed and implemented to handle non-determinism arising in the so called "algebraic " tableau reasoning. In this section, we introduce the syntax and semantics of the DL $\mathcal{ALCQ}$ as well as *tableau-based* reasoning.

2.1 Syntax and Semantics of $\mathcal{ALCQ}$

To define the terminology of a certain knowledge base (KB), DL relies on the notions of "concepts", "roles", and "individuals" combined using a set of operators (DL constructors) into structured and formally well defined descriptions. A *concept* is used to denote a set of domain elements with common characteristics. A *role* is used to denote a binary relationship between domain elements. An *individual* is used to name elements within the represented domain.

The basic DL $\mathcal{ALC}$ is a syntactic variant of the modal logic K, where all roles are atomic and complex concepts can be built using boolean operators ($\sqcap$, $\sqcup$, $\neg$), universal restriction ($\forall$), and existential ($\exists$) value restriction on atomic concepts. The DL $\mathcal{ALCQ}$ extends $\mathcal{ALC}$ with qualified cardinality restriction ($\mathcal{Q}$). Let N_{C}, N_{R}, and N_{I} be non-empty and pair-wise disjoint sets of concept names, role names, and individual names, respectively. A is used to denote an atomic concept ($A \in N_{\mathrm{C}}$), R is used to denote an atomic role ($R \in N_{\mathrm{R}}$). $\mathcal{ALCQ}$ concept expressions are defined using the syntax rule shown in Figure1 where $\top$ and $\bot$ are used to abbreviate $(C \sqcup \neg C)$ and $(C \sqcap \neg C)$, respectively. Figure 2 shows a simple DL knowledge base describing the concepts Man, Child, Male, and Father respectively.

TBox axioms	
Man	$\sqsubseteq$ Person $\sqcap$ Male
Child	$\sqsubseteq$ Person $\sqcap$ (Male $\sqcup$ Female) $\sqcap$ Offspring
Male	$\sqsubseteq \neg$Female
Father	$\sqsubseteq$ Man $\sqcap \exists$hasChild.$\top \sqcap \forall$hasChild.Child
ABox assertions	
$\langle$Joseph, Jonas$\rangle$: hasChild	
Joseph: Man	

Figure 2: Basic DL knowledge base consisting of a TBox and an ABox.

DLs differ from their predecessors in that they are equipped with formal logic-based semantics. An interpretation $\mathcal{I}$ is defined to give formal semantics. An interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set, called the domain of the interpretation, and $.^{\mathcal{I}}$ is the interpretation function. The interpretation function maps each atomic concept $A \in N_{\mathrm{C}}$ to a subset of $\Delta^{\mathcal{I}}$, each atomic role $R \in N_{\mathrm{R}}$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual $a \in N_{\mathrm{I}}$ to an element of $\Delta^{\mathcal{I}}$. The interpretation function is extended to satisfy $\mathcal{ALCQ}$ concept expressions as shown in Figure1, where n is a natural number ≥ 1.

Concepts, roles, and individuals represented using a DL language can be interrelated in such a way that implicit knowledge can be derived from explicitly represented knowledge. Every standard DL reasoning task can be reduced to a concept satisfiability check. For example, checking whether a concept C subsumes a concept D ($D \sqsubseteq C$?) can be reduced to checking the satisfiability of the concept expression $(D \sqcap \neg C)$. An interpretation $\mathcal{I}$ is said to be a model of C, if $C^{\mathcal{I}} \neq \emptyset$, we say that $\mathcal{I}$ satisfies C. The satisfiability of a concept C can therefore be decided by finding a model $\mathcal{I}$ for C. If no such model exists, then the concept C is unsatisfiable.

Many reasoning methods were investigated to handle Dl inference services such as *structural subsumption* (early 80s), *tableau-based* (1991), *automata-based* (2003) [4], *semantic binary tree* (2005), and *resolution-based* (2006). In the following, we introduce *tableau-based*, which is most widely used, and give an example which shows how the satisfiability of the concept Father as defined in Figure 2 can be decided using such methods.

2.2 Tableau Algorithms

The first DL tableau algorithm was designed for the DL $\mathcal{ALC}$ in 1991 [35] and later extended for more expressive logics [5, 18, 21, 22]. In general, tableau algorithms are considered as goal-directed decision procedures. They try to decide the satisfiability of a concept expression by constructing a corresponding model. The idea is that a concept C is satisfiable if a model exists that corresponds to an interpretation of C such that $C^{\mathcal{I}} \neq \emptyset$. Tableau algorithms work on concepts in *Negation Normal Form* [20], they are characterized by an underlying *data structure*, a set of *expansion rules*, a number of so-called *clash-triggers*, and sometimes a set of *blocking strategies*.

Definition 2.1 *Negation Normal Form (NNF) A concept expression is said to be in NNF if the negation ($\neg$) appears only in front of concept names. NNF can be obtained by pushing negations inwards [22] using DeMorgan's law (1-2) and the following equivalences:*

(1) $\neg(C \sqcup D)$	$\Longleftrightarrow$	$\neg C \sqcap \neg D$	*(4)* $\neg(\forall R.C)$	$\Longleftrightarrow$	$\exists R.\neg C$
(2) $\neg(C \sqcap D)$	$\Longleftrightarrow$	$\neg C \sqcup \neg D$	*(5)* $\neg(\geq nR.C)$	$\Longleftrightarrow$	$\leq (n-1)R.C$
(3) $\neg(\neg C)$	$\Longleftrightarrow$	C	*(6)* $\neg(\leq nR.C)$	$\Longleftrightarrow$	$\geq (n+1)R.C$

The Data Structure The data structure used to describe the model of a given concept C is usually a directed graph $G(V, E)$ referred to as a "completion graph". V is a set of vertices representing individuals in the domain, and E is a set of edges representing relations between individuals. Every node $x \in V$ is labeled by a concept expression $\mathcal{L}(x)$ that is satisfied by the represented individual.

Every edge between two nodes, x and y, is labeled by a set, $\mathcal{L}\langle x, y\rangle$, of role names satisfying the dependencies (Role successor-ship) between the two nodes. A symmetric binary relation $\neq$ is used to keep track of inequalities between two nodes. For most DLs, the construction of a model starts by initializing a root node (x_0) in G such that x_0 must satisfy the concept expression C. This is ensured by setting the label of x_0 to C ($\mathcal{L}(x_o) = \{C\}$).

Definition 2.2 *Role-Successor A node y is said to be a Role-Successor of a node x if there exists an edge $\langle x, y\rangle$ with R in its label ($R \in \mathcal{L}\langle x, y\rangle$) for some $R \in N_{\mathrm{R}}$. The node y is said to be an R-successor of x which is then said to be ancestor of y.*

$\sqcap$-Rule	**If** $C \sqcap D \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \nsubseteq \mathcal{L}(x)$ **Then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$
$\sqcup$-Rule	**If** $C \sqcup D \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ **Then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ with $E \in \{C, D\}$
$\exists$-Rule	**If** $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and there exists no y such that: y is a R-successor of x with $C \in \mathcal{L}(y)$ **Then** set $\mathcal{L}(\langle x, y\rangle) = \mathcal{L}(\langle x, y\rangle) \cup \{R\}$, and $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\forall$-Rule	**If** $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there exists y such that: y is an R-successor of x, and $C \notin \mathcal{L}(y)$ **Then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
choose-Rule	**If** $\leq nR.C \in \mathcal{L}(x)$, x is not blocked, and there exists y such that: y is an R-successor of x with $\mathcal{L}(y) \cap \{C, \dot{\neg}C\} = \emptyset$ **Then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ with $E \in \{C, \dot{\neg}C\}$
$\geq$-Rule	**If** $\geq nR.C \in \mathcal{L}(x)$, x is not blocked, and there are no $y_1, \ldots, y_n$ R-successors of x with $C \in \mathcal{L}(y_i)$, and $y_i \neq y_j$ for $1 \leq i \leq j \leq n$ **Then** create n new nodes $y_1, \ldots, y_n$ as R-successors of x such that for $1 \leq i \leq j \leq n$ set $y_i \neq y_j$, and $\mathcal{L}(y_i) = \{C\}$
$\leq$-Rule	**If** $\leq nR.C \in \mathcal{L}(x)$, x is not blocked, and there are $y_1, \ldots, y_m$ R-successors of x with $C \in \mathcal{L}(y_i)$, and $m \geq n + 1$ **Then** select y_j and y_i such that y_j such that not $y_j \neq y_i$, and Merge (y_i, y_j) and remove y_i

Figure 3: Tableau expansion rules for $\mathcal{ALCQ}$.

The Expansion Rules The graph G is gradually expanded according to some expansion rules designed to preserve and construct the logical dependencies encoded in C. These expansion rules (also known as tableau completion rules), correspond to constructors in the logic, they expand the initial graph by describing sub-graphs of the completion graph before and after rule application. In many cases, they only operate on a node and its direct neighbours. Figure 3 shows the tableau expansion rules corresponding to the DL $\mathcal{ALCQ}$.

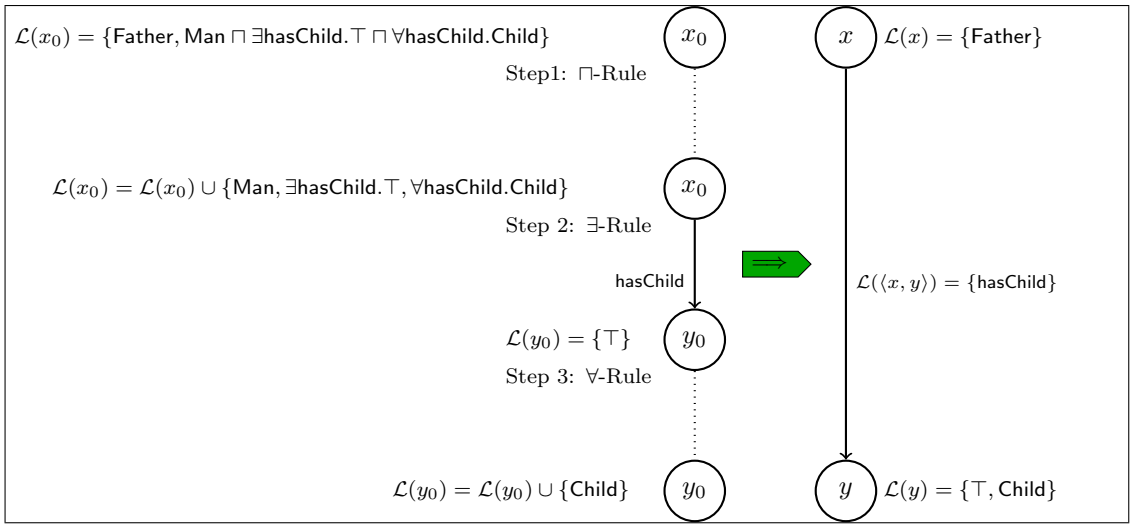


Figure 4: Tableau expansion during a satisfiability test of the concept Father as defined in Figure 2. The expansion on the left shows the step by step application of the expansion rules. The expansion on the right, shows the final completion graph representing a model for the concept Father.

Some rules add new nodes (e.g., $\exists$-Rule), and others yield more than one possible outcome (e.g., $\sqcup$-Rule). The latter ones are known as non-deterministic rules. In order to ensure termination in the case when cyclic descriptions are encountered (e.g., $C \sqsubseteq \exists R.C$), the rules employ the notion of blocking (see Definition 2.4). In practice, non-determinism means search and it is dealt with by exploring the various possible models. For an un-satisfiable concept, all possible expansions will lead to the discovery of an obvious contradiction known as a clash (see Definition 2.3), however, if at least one expansion leads to a complete and clash-free completion graph, then the concept is satisfiable.

Treating QCRs is done using the $\leq -$Rule, $\geq$-Rule, and the *choose*-Rule to ensure that individuals satisfy the at-least and at-most restrictions expressed using this constructor. The main idea is that at-least restrictions are treated by generating the required role-successors as new distinct nodes, with additionally asserting that the newly created role-successors are also members of the qualifying concept (C). At most restrictions are treated by non-deterministically merging role-successors whenever the number or role-successors exceeds the number allowed by the restrictions. A create and merge cycle is avoided using the inequality relations between nodes created to satisfy an at-least restriction such that two nodes x and y cannot be merged if $x \neq y$. Also, in order to detect unsatisfiability of concepts such as $(\geq nR \sqcap \leq mR.C \sqcap \leq mR.\neg C)$ with $n > m$, the non-deterministic *choose*-rule is used such that all R-successors are non-deterministically distributed over C or $\neg C$.

Some expansion rules require the merging of two nodes in order to satisfy an at-most restriction. When a node x is merged with another node y (`Merge` (x,y)), y inherits all of x's properties including its label, inequalities, ancestors (incoming edges) and successors (outgoing edges). Therefore, the label of x needs to be added to the label of y ($\mathcal{L}(y) = \mathcal{L}(y) \cup \mathcal{L}(x)$), all edges that lead to x are removed so that they lead to y. The completion graph is then pruned by removing x and, recursively, all *blockable* role-successors of x.

When no more rules are applicable, it means that all implicit knowledge has been made explicit and the completion graph is said to be complete. In the case of a satisfiable concept C, a complete and clash free completion graph is found and is said to be a completion model of C. Figure 4 shows the step by step expansion of the completion model for the satisfiability of the concept Father as defined in Figure 2.

Clash Triggers The tableau expansion algorithm stops either when no more rules are applicable (i.e., the completion graph is complete) or when a clash is detected.

Definition 2.3 *Clash A node x is said to contain a clash when a logical dependency is violated such as having x satisfy C and $\neg C$ ($\{C, \neg C\} \subseteq \mathcal{L}(x)$), or when a node x that must satisfy an at-most restriction ($\leq nR$) on its R-successors, and it already has m distinct R-successors with $m > n$.*

Blocking In some cases, expanding the completion graph does not lead to a complete graph. This can happen if the TBox axioms include cycles. For example, if a TBox $\mathcal{T}$ contains the axiom: $C \sqsubseteq \exists R.C$, then the satisfiability of C w.r.t. $\mathcal{T}$ will never stop because the tableau algorithm can go on creating new individuals with repeating structure. These situations are handled using *blocking* [5]; the idea is to block a node from applying rules if it needs to satisfy a concept expression that is satisfied by one of its ancestors.

Definition 2.4 *Blocked Node A node x is said to be directly blocked by a node y if it has a direct ancestor node y such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$. The node x is said to be blocked if it is directly blocked or one of its ancestors is blocked.*

Strategy of Rule Application The implementations of tableau algorithms have shown that expansion rules often need to be applied according to a certain strategy in order to ensure termination of the procedure. The general idea of the strategy is to apply shrinking rules (i.e., rules that result in merging nodes) before any other rule, and to apply these rules to lower level nodes before applying them to higher level nodes.

2.3 Complexity of DL Reasoning

Analyzing the complexity of DL reasoning is part of studying the inherent difficulty of its reasoning services. Usually, a distinction is made between analyzing the computational complexity of an inference service (e.g., checking the satisfiability of a certain concept), and analyzing the complexity of the underlying reasoning algorithms to solve an inference service.

The computational complexity of DL inference services is usually determined based on worst-case analysis of the size of a completion model, of a given knowledge base, and the time needed to construct such a model. Clearly DLs that enable QCRs enjoy additional expressive power with high computational complexity. QCRs are known to interact with other DL constructors, such as nominals ($\mathcal{O}$), and inverse role ($\mathcal{I}$). Table 1 shows the complexity of different DLs as can be found at the DL Complexity Navigator [1]. The high worst-case complexity initially led to the conjecture that expressive DLs might be of limited practical applicabilitiy.

DL Language	Satisfiability Checking
$\mathcal{ALC}$	ExpTime-complete
$\mathcal{ALCQ}$	ExpTime-complete
$\mathcal{ALCOQ}$	ExpTime-complete
$\mathcal{ALCOIQ}$	NExpTime-complete
$\mathcal{SROIQ}$	NExpTime-hard

Table 1: Computational complexity of DL satisfiability checking using a general TBox.

Analyzing the efficiency of a DL reasoning algorithms consists of analyzing its soundness, completeness, and termination. While soundness is evaluated by making sure that the algorithm will always find the correct answer, completeness means that the algorithm will explore every possible case before returning an answer, and termination means that the algorithm will always terminate. When studying the practical implication of a reasoning algorithm, termination is usually of a great importance; a correct reasoning algorithm is of limited use if it is not guaranteed to terminate. While worst-case complexity analysis serve as a theoretical estimate for the termination of a reasoning algorithm, the practical estimate is usually done through performance analysis on average cases.

[1] http://www.cs.man.ac.uk/ezolin/dl/.

A considerable gap was often observed between the theoretical presentation of a reasoning algorithm and that of its practical implementation. When analyzing the practical implication of a reasoning algorithm, one needs to distinguish between the theoretical efficiency of the algorithm, and its practical efficiency. The theoretical efficiency is usually measured as the theoretical worst-case complexity compared to the worst-case complexity of the inference problem. The practical efficiency is usually measured as a typical case practical performance. Early experiments with DL systems indicated that practical performance is a serious problem if the systems were not equipped with suited optimizations. Most modern DL reasoners implement tableau-based algorithms together with a set of sophisticated optimization techniques. In the remaining sections, we analyze non-determinism as a major source of complexity with DL reasoning.

3 Non-Determinism with Expressive DLs

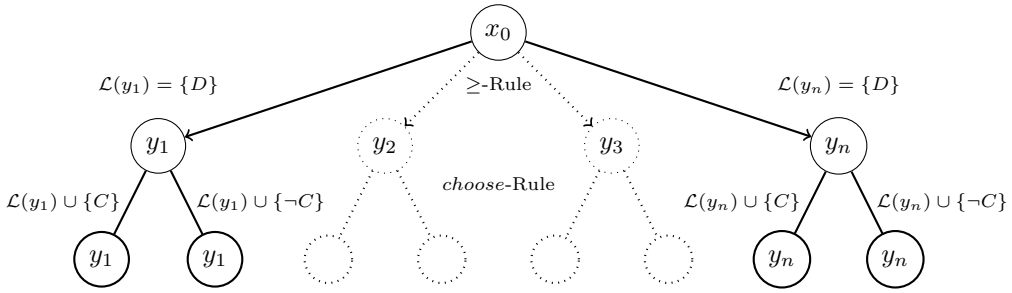
In practice, the poor performance of tableau algorithms is due to non-determinism in the expansion rules, which results in search of different possible expansions of the completion graph. We illustrate how Qualified Cardinality Restrictions (QCRs) aggravate non-determinism in *tableau-based* DL reasoning. We also outline how algebraic tableau reasoning extends *tableau-based* reasoning to better handle QCRs and argue why parallelizing algebraic tableau reasoning is more promising. In the following sections we outline a Fork/Join framework for handling non-determinism arising from algebraic tableau reasoning.

QCRs introduce non-determinism in choosing a distribution (*choose*-Rule) for each role-successor created to satisfy at-least restrictions, and when merging those role-successors is necessary to satisfy at-most restrictions ($\leq$-Rule). Such non-determinism can be aggravated when the number of nodes created in a completion graph increases (either due to a large number of at-least restrictions $\geq nR.C$, or due to large numbers (n) used in these restrictions), possibly the number of rules applied to these nodes increases and some of these rules might also be non-deterministic (See Examples 3.1, 3.2). Moreover, in cases where an at-most restriction is violated, the non-determinism introduced when merging two nodes can cause a blow up of the search space especially when the qualification used with the at-most restriction also contains a disjunction (See Examples 3.2, 3.3).

3.0.1 Examples

The following examples illustrates the effect of non-determinism with QCRs on the tableau-based reasoning algorithm introduced in Section 2.

Example 3.1 ***(Non-Determinism Due to The** choose**-Rule)*** *When testing the satisfiability of* $(\geq nR.D \sqcap \leq mR.C)$*, the tableau algorithm starts with a node* x_0 *such that* $\mathcal{L}(x_0) = \{(\geq nR.D \sqcap \leq mR.C)\}$*. After applying the* $\sqcap$*-Rule to* x_0*, the label is extended to* $\mathcal{L}(x_0) = \{(\geq nR.D \sqcap \leq mR.C), \geq nR.D, \leq mR.C\}$*. Applying the* $\geq$*-Rule, creates* n *distinct* R*-successors,* $y_1, \ldots, y_n$*, of* x_o *such that* $\mathcal{L}(y_i) = \{D\}$ *for each* $1 \leq i \leq n$*. The choose-Rule non-deterministically assigns* C*, or* $\neg C$ *to the label of each* R*-successor of* x_0*.*



$\geq$-Rule: create n R-successors of x_0 such that $\mathcal{L}(\langle x_0, y_i \rangle) = \{R\}$ for $1 \leq i \leq n$.
choose-Rule: extend the label of each R-successor of x_0 such that $\mathcal{L}(y_i) = \mathcal{L}(y_i) \cup E \in \{C, \neg C\}$.

Figure 5: Completion graph showing one source of non-determinism: the *choose*-Rule.

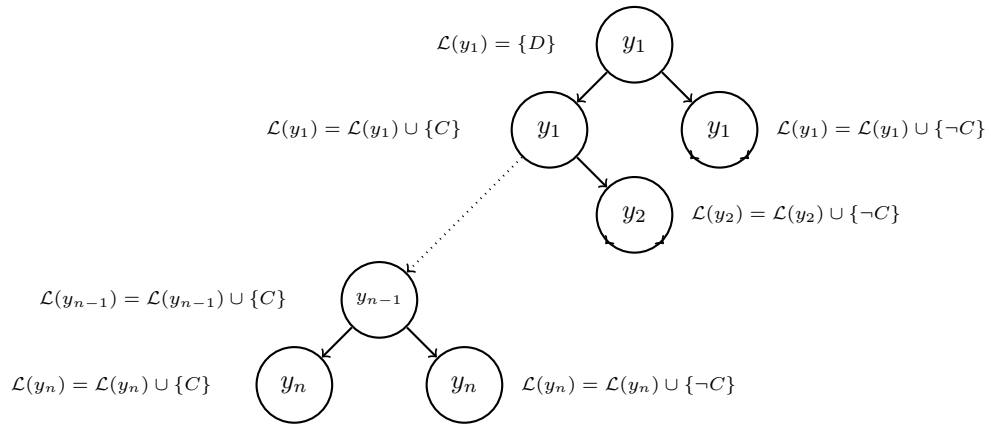


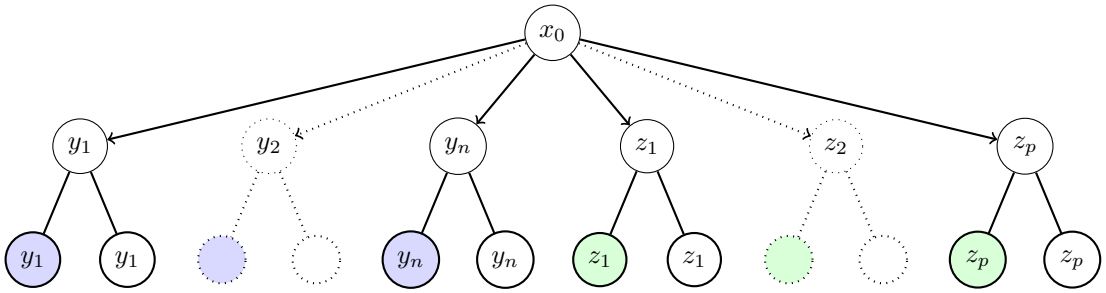
Figure 6: Completion graph expansion tree due to the *choose*-Rule.

The completion graph can therefore be expanded in 2^n different ways, as shown in Figure 6, based on the distribution of these R-successors. Having more than one at-most restriction, also affects the expansion such that if there are q at most restrictions, then the total number of branches becomes equal to $2^{n \times q}$.

Example 3.2 ***(Non-determinism due to the** choose**-Rule and the $\leq$-Rule)*** *This example shows the effect of non-determinism due to the $\leq$-Rule when the number of R-successors, created to satisfy at-least restrictions ($\geq nR.D$, $\geq pR.E$), exceeds the number allowed by an at-most restriction ($\leq mR.C$), $(n+p) > m$. As shown in Figure 7, the $\geq$-Rule creates $(n+p)$ R-successors of x_0 such that $y_1, \ldots y_n$ are mutually disjoint and $z_1, \ldots, z_p$ are mutually disjoint. This means y-nodes cannot be merged together, and z-nodes cannot be merged together either (to make sure the $\geq$ is always satisfied and avoid a create and merge cycle known as a "yo-yo" effect). The choose-Rule non-deterministically expands the completion graph by creating two branching points in the search space for each R-successor of x_0. Having $(n+p)$ R-successors, means that the search space is expanded with 2^{n+p} branches. Since $(n+p) > m$, the $\leq$-Rule non-deterministically merges a y_i node such that $C \in \mathcal{L}(y_i)$ with a z_j node such that $C \in \mathcal{L}(z_i)$ until $(n+p) \leq m$. Assuming that the exceeding number of R-successors is given as $k = \mid m-(n+p) \mid$, this means that k y_i nodes need to be merged with k z_j nodes. The number of ways that the n R-successors $(y_1, \ldots, y_n)$ can be grouped into k R-successor is defined as $K_y = \frac{(n!)}{(n-k)!}$ and the number of ways that the p R-successors, $z_1, \ldots, z_p$, can be grouped*

into k elements is defined as $K_z = \frac{(p!)}{(p-k)!}$. *This means that the number of ways to merge k* y_i *nodes with k* z_j *nodes is given as* $K_{yz} = \frac{(n!)}{(n-k)!)} \times \frac{(p!)}{(p-k)!}$. *The nodes that can be merged are highlighted. Due to non-deterministic rules, the completion graph is expanded in* 2^{n+p} *ways (choose-Rule), and* K_{yz} *ways (≤-Rule). Note also that having more at-least restrictions, not only affects the expansion of the completion graphs by introducing more expansions due to the choose-Rule, but also introduces more possible ways to merge excess R-successors.*

$$\mathcal{L}(x_0) = \{\geq nR.D, \geq pR.E, \leq mR.C\}$$



$\geq$-Rule: create n R-successors of x_0 such that $\mathcal{L}(\langle x_0, y_i\rangle) = \{R\}$ for $1 \leq i \leq n$.

$\geq$-Rule: create p R-successors of x_0 such that $\mathcal{L}(\langle x_0, z_j\rangle) = \{R\}$ for $1 \leq j \leq p$.

choose-Rule: extend the label of each R-successor of $x_0$with $E \in \{C, \neg C\}$.

$\leq$-Rule: non-deterministically merge y_i with z_j until $\leq mR.C$ is satisfied.

Figure 7: Completion graph expansion showing two sources of non-determinism: the *choose*-Rule and the ≤-Rule.

Example 3.3 ***(Non-determinism due to the** choose**-Rule, ⊔-Rule, and ≤-Rule)*** *This example shows a source of non-determinism when handling QCRs using disjunctive descriptions to qualify R-successors such as in the following concept description:* $(\geq nR.D \sqcap \geq pR.E \sqcap \leq mR.(A \sqcup B))$. *As illustrated in Figure 8, in addition to the sources of non-determinism illustrated in Examples 3.1 and 3.2, the ⊔-Rule introduces non-deterministic expansions of the labels of R-successors of* x_o *and doubles the number of nodes to be considered by the ≤-Rule.*

3.1 Tableau-based Algebraic Reasoning

Tableau-based algebraic reasoning has been introduced to better handle the complexity of reasoning with QCRs. The algorithm is hybrid; it relies on tableau expansion rules working together with an integer programming solver (e.g., simplex solver). We outline how tableau-based algebraic reasoning extends tableau-based reasoning introduced in Section 2 in terms of the underlying *data structure*, *expansion rules*, *clash-triggers*. We also show the non-determinism introduced.

The Data Structure A concept model is represented using a *compressed completion graph*, which is a directed graph $G = (V, E, \mathcal{L}, \mathcal{L}_E)$ where each node $x \in V$ is labeled with $\mathcal{L}$ and $\mathcal{L}_E$ where $\mathcal{L}(x)$ denotes a set of concept expressions, and $\mathcal{L}_E(x)$ denotes a set of inequations. Each edge $\langle x, y\rangle \in E$ is labeled with a set, $\mathcal{L}(\langle x, y\rangle)$, of role names. We denote by ξ_x the set of inequations in $\mathcal{L}_E(x)$ encoding the QCRs in $\mathcal{L}(x)$. An integer solution σ for ξ_x maps each variable v occurring in ξ_x to a non-negative integer p such that σ is a distribution of role successors of x. The distribution is consistent with the lower and upper bounds expressed in at-least $(\geq nR.D)$ and at-most $(\leq nR.C)$ restrictions.

To decide the satisfiability of C, the algorithm starts with the *compressed completion graph* $G = (\{x_0\},\emptyset,\{C\},\emptyset)$. Where the root node $x_0 \in V$ is such that $C \in \mathcal{L}(x_0)$. Note that x_0 represents an arbitrary individual and all concepts in C are assumed to be in *negation normal form*.

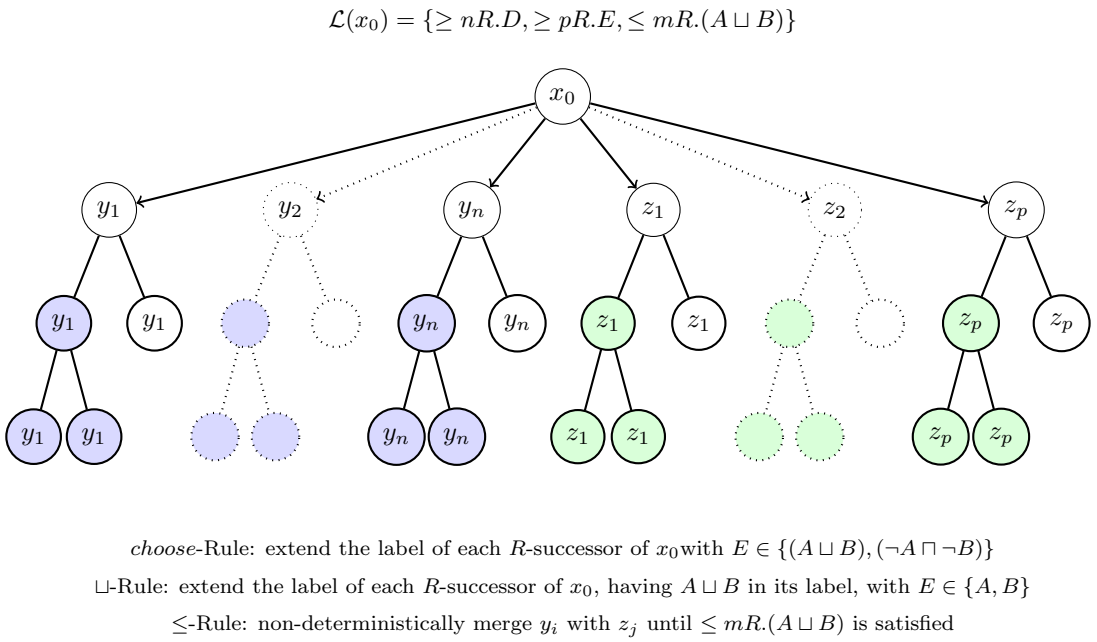


Figure 8: Completion graph expansion showing three sources of non-determinism: the *choose*-Rule, the $\sqcup$-Rule, and the $\leq$-Rule.

The Expansion Rules The compressed completion graph G is then expanded by applying the expansion rules given in Figure9 until no more rules are applicable or a clash occurs. When G is complete and there is no clash, we have a pre-model and the algorithm returns that C is satisfiable.

The $\leq$-*Rule* and the $\geq$-*Rule* are responsible for encoding the qualified cardinality restrictions in the label $\mathcal{L}(x)$ of a node x into a set (ξ_x) of inequations maintained in $\mathcal{L}_{\mathrm{E}}(x)$. An inequation solver is always active and is responsible for finding a non-negative integer solution σ for ξ_x or for triggering a clash if no solution is possible.

Each of the variables used in inequations represents the cardinality of a certain set P of role successors. The set P is derived by applying the atomic decomposition technique on the set of role names used with QCRs. For a complete overview about this technique and how it is used to decompose the set of domain elements into disjoint partitions refer to [12].

In order to preserve completeness, the *ch-Rule* is used to check for empty partitions. Given a set of inequations in the label ($\mathcal{L}_{\mathrm{E}}$) of a node x and a variable v corresponding to a partition $\alpha(v)$ in $\mathcal{P}$, we distinguish between two cases:

- (i) The case when a partition must be empty; this can happen when restrictions of individuals assigned to this partition trigger a clash.
- (ii) The case when a partition can have at least one individual; if a partition can have one individual without causing any clash, this means that we can have n $(n \geq 1)$ [2] individuals also in this partition without a clash.

Since the inequation solver is unaware of logical restrictions of filler (role successor) domains we allow an explicit distinction between cases (i) and (ii). We do this by non-deterministically assigning ≤ 0 or ≥ 1 for each variable v occurring in $\mathcal{L}_{\mathrm{E}}(x)$.

The *fil-Rule* is used to generate role successors for a node x depending on the distribution (solution) returned by the inequation solver. For a proof of correctness we refer the reader to [10].

Clash Triggers A node x in V is said to contain a clash if either:

[2]The value of n is decided by the inequation solver.

$\sqcap$**-Rule**	**If** $C \sqcap D \in \mathcal{L}(x)$, and $\{C, D\} \nsubseteq \mathcal{L}(x)$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$.
$\sqcup$**-Rule**	**If** $C \sqcup D \in \mathcal{L}(x)$, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$ **then** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ **or** set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{D\}$.
$\forall$**-Rule**	**If** $\forall R.C \in \mathcal{L}(x)$ and $R \in \mathcal{L}(\langle x, y\rangle)$ with $C \notin \mathcal{L}(y)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$.
$\forall_{(\backslash)}$**-Rule**	**If** $\forall(R \backslash S).C \in \mathcal{L}(x)$, and there exists y such that $R \in \mathcal{L}(\langle x, y\rangle)$ and $S \notin \mathcal{L}(\langle x, y\rangle)$ **then** set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$.
$\leq$**-Rule**	**If** $(\leq nR) \in \mathcal{L}(x)$ and $\xi(R, \leq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \leq, n)\}$.
***ch*-Rule**	**If** there exists v occurring in $\mathcal{L}_E(x)$ with $\{v \geq 1, v \leq 0\} \cap \mathcal{L}_E(x) = \emptyset$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v \geq 1\}$ **or** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v \leq 0\}$.
$\geq$**-Rule**	**If** $(\geq nR) \in \mathcal{L}(x)$ and $\xi(R, \geq, n) \notin \mathcal{L}_E(x)$ **then** set $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{\xi(R, \geq, n)\}$.
***fil*-Rule**	**If** there exists v occurring in $\mathcal{L}_E(x)$ such that (i) $\sigma(v) = m$ with $m > 0$, and (ii) there are no m nodes $y_1 \dots y_m$ with $\mathcal{L}(\langle x, y_i\rangle) = \alpha(v)$ for $1 \leq i \leq m$ **then** create m new nodes $y_1 \dots y_m$ and set $\mathcal{L}(\langle x, y_i\rangle) = \alpha(v)$ for $1 \leq i \leq m$.

Figure 9: Tableau expansion rules for algebraic reasoning with the DL $\mathcal{ALCQ}$

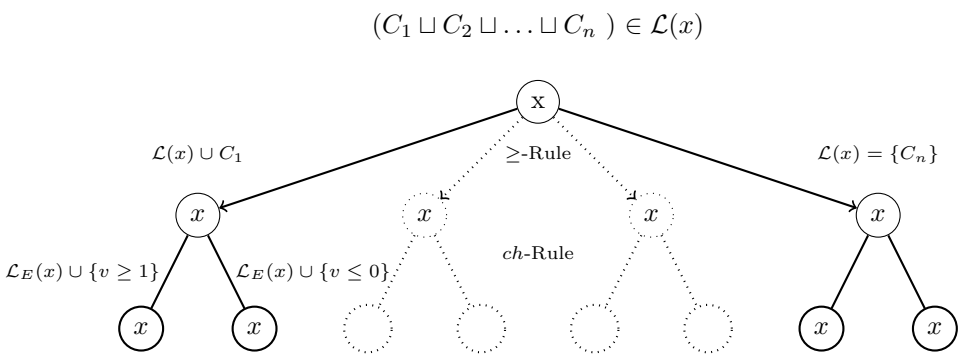
- (i) $\{C, \neg C\} \subseteq L(x)$, or
- (ii) the set of inequations $\xi_x \subseteq \mathcal{L}_E(x)$ does not admit a non-negative integer solution.

Case (ii) is decided by the inequation solver. When no rules are applicable or there is a clash, a *compressed completion graph* is said to be *complete*.

Strategy of Rule Application In order to preserve the correctness of the algorithm, we assign the *fil-Rule* the lowest priority; all other rules can be fired in arbitrary order. Complete proofs of soundness, completeness, and termination of the algebraic reasoning tableau can be found in [12].

3.1.1 Non-determinism with QCRs

In the case of the algebraic tableau algorithm for the DL $\mathcal{ALCQ}$, non-determinism due to disjunctive descriptions ($\sqcup$) is handled similar to the case of standard tableau. However, handling QCRs relies on the use of the atomic decomposition technique [31], which computes disjoint partitions by considering all possible interactions between domain elements. As shown in Figure10, applying the *ch*-Rule and the $\sqcup$-Rule leads to independently constructing completion graphs for each rule choice. Even though the number of choices for the *ch*-Rule is fixed to 2, the number of times this rule is fired depends on the number of variables used for the inequations. This means that the number of qualified cardinality restrictions within a concept expression affects the number of times this rule is applied. In [9], it was reported that the order in which the variables are selected for this rule affects performance. Applying this rule for all the variables before deciding a satisfiability check is a major source of performance degradation. Similarly to the $\sqcup$-Rule, exploring all choices can significantly exhaust the memory and time required to perform a satisfiability check. Let N_v denote the total number of variables used by the atomic decomposition technique. N_v is calculated as 2^r, where r denotes the number of qualified cardinality restrictions. In the worst case, when all possible completion graphs



$\sqcup$-Rule: extend the label of the node x such that $\mathcal{L}(x) = \mathcal{L}(x) \cup E \in \{C_1, C_2, \ldots, C_n\}$.

ch-Rule: extend the equations label of the node x such that $\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup E \in \{v \geq 1, v \leq 0\}$.

Figure 10: Non-deterministic expansions with *ch*-Rule and $\sqcup$-Rule

are explored, the *ch*-Rule is applied $2^{2^r} - 1$. This means that the *ch*-Rule is responsible for a double exponential blow up of the search space of the algorithm. In theory, the worst-case complexity due to non-determinism is aggrevated compared to a standard tableau-based reasoning algorithm.

On the other hand, this handling of domain elements, as shown in Figure 10, results in only one additional source of non-determinism rather than the two sources for handling qualified cardinality restrictions with the standard tableau. We argue that tableau-based algebraic reasoning appears to have a better potential for parallelization than standard tableau for the following reasons:

- having less sources of non-determinism (2 instead of 3) means less overhead in managing concurrent execution of non-deterministic rules. This also means that adopting optimizations such as dependency directed backtracking becomes more fine grained and less complicated.

- the *ch*-rule always fires for two choices. This means that the search trees resulting from the distinct branches have similar structure which facilitates load balancing between parallel expansions of the search tree. Load balancing is a common goal in parallel computing where unequal thread workloads can easily diminish the performance gain of parallelization.

- satisfying qualified cardinality restriction is delegated to an inequation solver and can be done in isolation from tableau expansion. This means that the task of satisfying the inequations can be delegated to the use of separate threads, or even FPGAs [37] and GPUs (Graphical Processing Units).

- the use of "compressed completion graph" consisting of proxy nodes representing sets of domain elements instead of completion graphs consisting of a node representing each domain element allows the use of a smaller data structure representing the completion model. This means that a smaller amount of data needs to shared among and communicated between parallel tasks thus reducing the communication overhead between threads.

In the following section we illustrate our parallel framework for handling non-determinism with algebraic tableau-based reasoning.

4 Parallelizing Non-determinism

Tableau-based reasoning has been the most common choice for Description Logics tools. In the previous sections, we introduced the core elements of a tableau-based reasoning algorithm and those of a tableau-based algebraic reasoning algorithm for satisfiability checking. A satisfiability check is at the core of most DL reasoning services. As illustrated in Section 2, a satisfiability check works

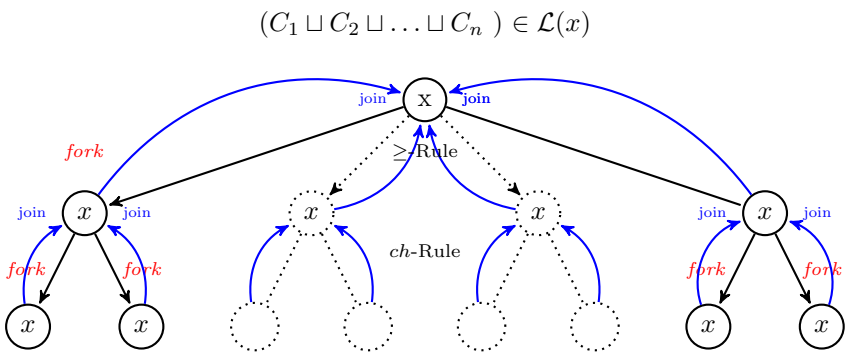


Figure 11: Fork/Join handling of non-deterministic completion graph expansions.

by constructing a model represented using a completion graph. The search for a model requires completion graph expansion using tableau rules, some of which are non-deterministic. Given a sequential execution, non-deterministic choices are explored upon clash-detection.

The previous section showed how non-determinism in tableau expansion rules can become a major source of complexity for such reasoning algorithms. On the other hand, nodes belonging to graph expansions due to non-deterministic choices do not exchange information. We use this feature and extend completion graph expansion by handling non-deterministic choices in parallel using the Fork/Join framework. In this framework, non-deterministic choices can be expanded concurrently and the search for a model halts if any of the expansions returns a clash-fee model. We choose to explore this framework in tableau-based algebraic reasoning because we believe that it is more amenable for parallelization.

4.1 The Fork/Join Framework

The Fork/Join parallelism framework is based on Doug Lea's work [25] and supports recursive, divide-and-conquer parallelism. This parallel framework was implemented as part of the Java Specification request JSR166 to provide concurrency utilities and has been integrated as part of the standard Java Concurrency library since Java 7. This framework abstracts the handling of tasks and thread management allowing developers to focus more on the development of the parallel algorithm and its implementation.

The Fork/Join framework expresses parallelism using two primitives: *fork* and *join*. A *fork* operation starts the execution of a task in parallel with its parent creator. The *join* operation merges control upon completion of tasks by both the parent (invoking the a *fork* operation) and child tasks.

We implement our *Thread Pool* design [13] using a pool, the *ForkJoinPool*, of worker threads. Using a *thread Pool* allows controlling the continuation of tasks. With the *ForkJoinPool*, each worker thread has its own dequeue. When a task generates a new subtask using the *fork* operation, the thread (parent) pushes the subtask onto the head of its dequeue. The tasks that are currently being executed can also generate new tasks using *fork*. The newly generated tasks are also added to the dequeue. This task creation is ideal for handling non-deterministic rule application as shown in Figure11 because one cannot determine beforehand the number of graph expansion tasks until a non-deterministic rule becomes applicable. Using the number of variables generated by the atomic decomposition technique, one could possibly estimate the number of tasks needed to handle the *ch*-rule. However, in the case of the $\sqcup$-Rule this is not possible. The Fork/Join pool of tasks is usually constructed such that the level of parallelism depends on the number of available processors.

Instead of implementing a *Leader/Followers* handling of thread management as suggested in [13], we rely on the *work-stealing* behaviour of threads used by the Fork/Join framework. When a worker thread has completed execution of a certain task, it can fetch the next task from its dequeue. If the thread's dequeue is empty, it steals another task from the tail of another thread's dequeue. This

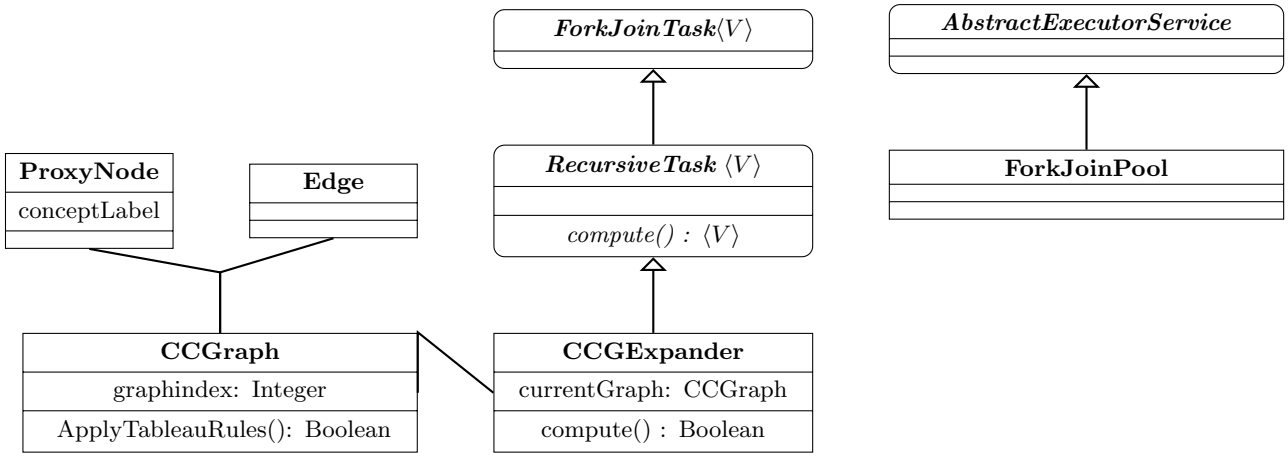


Figure 12: Diagram depicting the relation between the CCGraph and the ForkJoinTask classes.

means no thread is waiting in the pool to be assigned a new task. Such behaviour can maximize throughput while minimizing contention.

4.2 Parallel HARD

The parallel framework described in the previous section is implemented in HARD (Hybrid Algebraic Reasoner for Description Logics), which is a prototype DL reasoner implementing the algebraic tableau reasoning algorithm for the DL $\mathcal{SHOQ}$[3], and is equipped with the optimizations discussed in [9].

Figure12 shows the relations between the classes responsible to implement the Fork/Join framework. When a satisfiability check for a concept expression E is initiated, a CCGraph object is created such that it includes a ProxyNode, *pNode*, with E added to *pNode.conceptLabel*. A ForkJoinPool object is created and initiated with a new CCGExpander task, which is also a RecursiveTask. This means that when the CCGExpander task is invoked, the compute() method is executed. Algorithm 1 illustrates how this method is used to decide the satisfiability of E using ForkJoinTasks.

The CCGExpander distinguishes between deterministic ($\sqcap$-Rule, $\forall$-Rule, $\forall_{\backslash}$-Rule, $\leq$-Rule, $\geq$-Rule, and fil-Rule) and non-deterministic rules (ch-Rule and $\sqcup$-Rule) such that the tasks of applying tableau rules are split recursively until satisfiability of the original concept expression, E, can be decided. The following task splitting strategy is adopted:

- In the case where the applicable rule at a given node is deterministic. This means that the graph expansion task is simple enough to be executed sequentially using *ApplyTableauRules()* method.
- In the case where the applicable rule at a given node is non-deterministic, and must explore n choices. This means that the graph expansion task can be split into n CCGExpander subtasks that can run concurrently. A CCGExpander task is created for each possible choice of the non-deterministic rule. For example, in the case of the ch-Rule the CCGExpander task is split into 2 tasks using the *fork()* operation. Each *task* is added to the list and a *fork* operation is invoked. These tasks are distributed across threads and are run in parallel. If any of the tasks succeeds in creating a complete and clash-free model, then the satisfiability can be decided and the original concept expression is satisfiable. Otherwise, (if none of the tasks returns a positive answer) the original concept expression is unsatisfiable.

The ability to dynamically handle and generate new tasks is crucial because the tasks cannot be anticipated until a rule becomes applicable. The work stealing algorithm is a key feature anticipated to enable dynamic load balancing.

[3]The DL $\mathcal{SHOQ}$ extends the DL $\mathcal{ALCQ}$ with transitive roles ($\mathcal{S}$) and nominals ($\mathcal{O}$).

Algorithm 1 `CCGExpander.COMPUTE()`

```
if a deterministic rule is applicable then
  return currentGraph.ApplyTableauRules();
else if a non-deterministic rule is applicable then
  List⟨RecursiveTask⟨Boolean⟩⟩ forks = new LinkedList⟨⟩();
  for each applicable choice do
    CCGExpander task = new CCGExpander(currentGraph, nextDDBGraph);
    forks.add(task);
    task.fork();
  end for
  for each task in forks do
    if task.join() == true then
      return true
    end if
  end for
  return false
end if
```

4.3 Optimizations

The sequential version of the HARD reasoner outperformed most existing DL reasoners in handling complex satisfiability tests, where the concept expressions used QCRs. Such results were possible due to key optimization techniques discussed in [11, 9] such as *lazy partitioning* and the use of *noGood* variables. We analyze whether these optimizations can still be adopted in the parallel framework.

Preprocessing Optimizations The optimizations used before applying tableau expansion rules and which aim at bounding the size of the search space using role hierarchy relations, and disjointness relations do not interfere with the task splitting adopted by our Fork/Join framework. Therefore, such optimizations can still be used.

Look Ahead Optimizations for Backtracking The optimizations used to reduce the size of the search space by discarding choice points as soon as a non-deterministic rule is applicable can also be adopted. For example, the $\sqcup$-*Rule Look Ahead* optimization discards a choice point when the $\sqcup$-Rule is applied to a node x if this choice points is known to lead to a clash.

Look Back Optimizations for Backtracking *Back-jumping* or *conflict-directed backtracking* are improved backtracking methods adapted to DL reasoning as dependency directed backtracking (DDB)[23]. Adopting existing DDB techniques for DL reasoning helps prune the search space due to the $\sqcup$-Rule but not the choice points due to the *ch*-Rule, which is the rule responsible for the double exponential blow up of the search space (as shown in Section 3.1.1). The sequential version of the algebraic tableau reasoner implements backtracking during two phases:

- *Back-jumping* phase: during this phase, an adapted form of DDB analyzes the source of a clash and decides how far to backtrack.
- *Learning* phase: after consulting the source of a clash, the algorithm can learn that a certain partition must be empty and assigns the variable for this partition as a *noGood* variable. This learned information is recorded as a new constraint in form of global variables.

Adopting these optimizations in our parallel framework requires the use of shared data among threads. For example, in order to enable the *Learning* phase optimizations, multiple threads require access (read/write) to the partition variables either to set the learned constraint, or to check if a certain variable is a *noGood*. We use the `AtomicBoolean`, from the java concurrency library, to manage the use of *noGood* variables. Once a partition is learned to be empty, the corresponding variable is updated atomically using the `compareAndSet` operation. This handling of variables allows a better

$$\mathsf{C}_1 \quad \sqsubseteq \geq 4\mathsf{nr}.\top \sqcap \geq 2\mathsf{nr}.\mathsf{C}_1 \sqcap \ldots \sqcap \geq 2\mathsf{nr}.\mathsf{C}_\mathsf{i} \\ \sqcap \leq \mathsf{nr}.(\neg\mathsf{C}_1 \sqcup \neg\mathsf{C}_2) \sqcap \ldots \sqcap \leq \mathsf{nr}.(\neg\mathsf{C}_\mathsf{i} \sqcup \neg\mathsf{C}_{\mathsf{i}+1})$$

$$\mathsf{C}_2 \quad \sqsubseteq (\geq 4\mathsf{nr}.\top \sqcup \geq 2\mathsf{nr}.\mathsf{C}_1 \sqcup \cdots \sqcup \geq 2\mathsf{nr}.\mathsf{C}_\mathsf{i}) \\ \sqcap \leq \mathsf{nr}.(\neg\mathsf{C}_1 \sqcup \neg\mathsf{C}_2) \sqcap \cdots \sqcap \leq \mathsf{nr}.(\neg\mathsf{C}_\mathsf{i} \sqcup \neg\mathsf{C}_{\mathsf{i}+1})$$

$$\mathsf{C}_3 \quad \sqsubseteq \geq 4\mathsf{nr}.\top \sqcap \geq 2\mathsf{nr}.\mathsf{C}_1 \sqcap \cdots \sqcap \geq 2\mathsf{nr}.\mathsf{C}_\mathsf{i} \\ \sqcap(\leq \mathsf{nr}.(\neg\mathsf{C}_1 \sqcup \neg\mathsf{C}_2) \sqcup \cdots \sqcup \leq \mathsf{nr}.(\neg\mathsf{C}_\mathsf{i} \sqcup \neg\mathsf{C}_{\mathsf{i}+1}))$$

Figure 13: Concept descriptions relying on the use of QCRs.

throughput, and a resistance to deadlocks and synchronization problems between threads. Designing an efficient solution to the shared data required for the applicability of the optimizations used for a *Back-jumping* phase needs further investigation. Also, we do not consider the optimizations aimed at handling the nominals constructor such as *Lazy Partitioning, Lazy Nominal Generation*. As a preliminary experiments, we only consider parallel HARD for the DL $\mathcal{ALCQ}$.

5 Preliminary Experiments

Before implementing the Fork/Join parallel framework to our HARD reasoner, we converted HARD to rely on the latest OWL API[4] using JAVA7, which is required to enable the Fork/Join framework. Parallel HARD is implemented as a shared-memory program using the `java.concurrency library`. HARD preferences allow users to select a sequential execution or a parallel execution of the tableau-rules, while also specifying the maximum number of threads to be used by the *ForkJoinPool*.

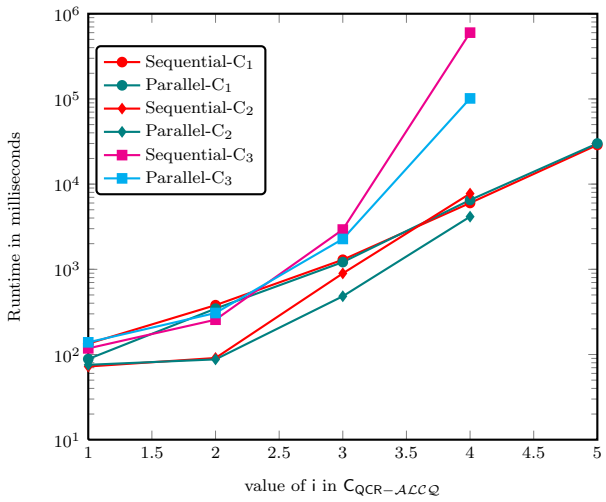
We ran preliminary experiments on a standard MacBook Pro with 2.66GhZ Intel Core 2 Duo and 4GB main memory running OS X version 10.9.1. We set the number of worker threads in the Fork/Join pool to default. We set the maximum java heap size to 2GB and we use the default settings for the garbage collector. The test cases used were mainly those used to evaluate the sequential version of HARD in [9]. Some of the results of these test case are shown in Figure14a. We also ran preliminary experiments on Glooscap, a parallel HP AMD Opteron Linux cluster provided by ACEnet. Glooscap consists of 180 nodes with 220 cores in total, the processors on these nodes range between 2.3-2.7 GHz Dual-Core AMD Opteron; Gigabit Ethernet. Some of the results of these test case are shown in Figure14b and in Figure15.

Each test case consists of an .owl file representing an OWL ontology, in the RDF/XML format, modelled using Protégé 4.1.[5] The run-time of each test is reported in milliseconds as the average run-time of three to five separate executions of the test using HARD. Each run-time represents the time needed for HARD to perform a concept satisfiabilty test. The reported time is in milliseconds, and the TimeOut is set to 10 minute (600000 ms). As a preliminary evaluation, we test parallel HARD against *The number of qualified cardinality restrictions used*, and *The satisfiability versus the unsatisfiability of the given concept expression.*

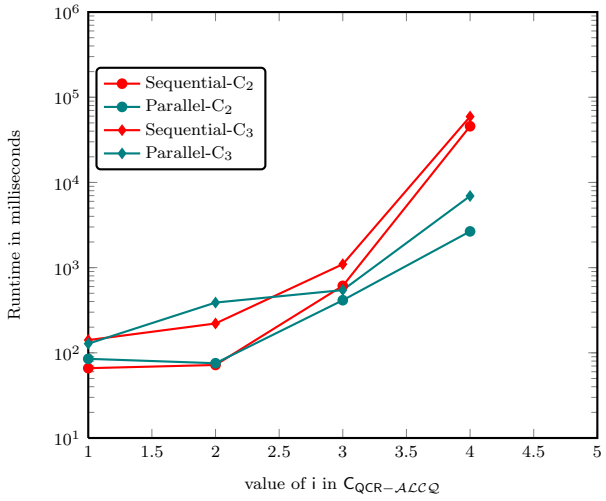
5.0.1 Testing The Number of Qualified Cardinality Restrictions

It was shown in [9] that the complexity of the tableau-based algebraic algorithm implemented by HARD is characterized by a double exponential function of the number of QCRs, q. It is therefore expected that as the number of QCRs increases, the performance of HARD degrades. Also, q affects the number of variables needed to solve the total number of inequations encoded. This means that q affects the number of times the *ch*-Rule is applicable. The effect of increased QCRs in descriptions

[4]Last modified 2014-01-18 and designed to support OWL 2.0.
[5]http://protege.stanford.edu/



(a) The maximum number of threads is fixed to 2.



(b) The maximum number of threads is fixed to 12.

Figure 14: Effect of increasing the number of QCRs in a satisfiable concept expression.

is tested using the concepts C_1, C_2, C_3 as described in Figure13. With C_1, the number of QCRs is increased within a conjunctive description, whereas with C_2 and C_3, the number of QCRs is increased within disjunctions.

The effect of increasing the number of QCRs on reasoning performance is shown in Figure14. As the number i increases, the number of QCRs increases as well as the number of disjuncts for the $\sqcup$-Rule. The results clearly show that as the number of disjuncts increases, a parallel framework can decide the satisfiability more quickly than a sequential framework. The results also show that as the number of disjuncts increases within a disjunctive expression, the parallel framework might even achieve a super linear speedup, as shown in Figure15. A super linear speedup might be due to the cumulative effect of handling the *ch*-Rule and the $\sqcup$-Rule concurrently. The speedup factor is calculated based on the following formula $S = \frac{T}{T_n}$ where T is the time reported sequentially, and T_n is the time reported using n threads. We also noticed that with the test cases where QCRs were increased in a conjunctive expression (i.e., case of C_1), the parallel framework was as quick as a sequential one. This might be an indication that the performance is gained due to the parallelization of the $\sqcup$-Rule.

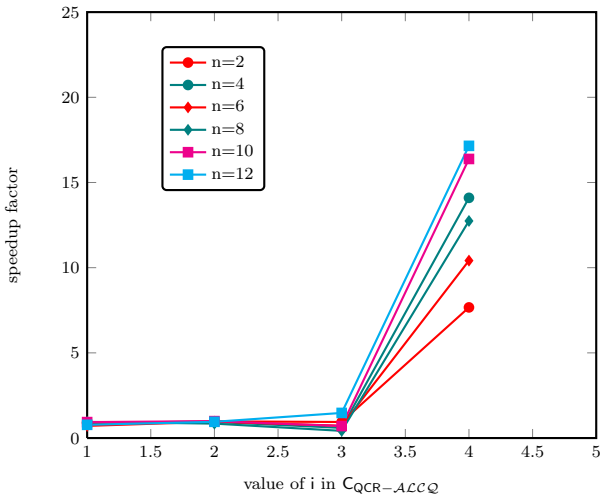


Figure 15: Effect of increasing the number of QCRs on the speedup factor achieved. n represents the maximum number of threads used in the thread pool.

5.0.2 Satisfiable Versus Unsatisfiable Concepts

The effect of satisfiable and unsatisfiable test cases is checked using a series of test cases which decide the satisfiability of the following concept:

$$\begin{array}{ll} \mathsf{C}_{\mathsf{Back}-\mathcal{ALCQ}} & \sqsubseteq \geq 3\mathsf{r}.\mathsf{D}_1 \sqcap \ldots \sqcap \geq 3\mathsf{r}.\mathsf{D}_\mathsf{i} \sqcap \leq 3\mathsf{i}-1\mathsf{r}.\top \\ \mathsf{D}_\mathsf{q} & \sqsubseteq \neg \mathsf{D}_\mathsf{p} \text{ for all } \mathsf{q} < \mathsf{p} \end{array}$$

$\mathsf{C}_{\mathsf{Back}-\mathcal{ALCQ}}$ is satisfiable if a completion graph can be constructed where a node a has $3\mathsf{i}$ r-successors such that, for a given value of i, 3 r-successors satisfy each $\geq 3\mathsf{r}.\mathsf{D}_\mathsf{i}$ and the total number of these r-successors cannot exceed $3\mathsf{i}-1$. This means that some r-successors must satisfy $(\mathsf{D}_\mathsf{q} \sqcap \mathsf{D}_\mathsf{p})$ for some $1 \leq \mathsf{p}, \mathsf{q} \leq \mathsf{i}$, however this is not possible because all D_p and D_q are disjoint. This renders the concept $\mathsf{C}_{\mathsf{Back}-\mathcal{ALCQ}}$ unsatisfiable. The complexity of this concept is due to the effect of non-determinism which requires backtracking in order to explore all possibilities. In the case of a tableau algorithm, backtracking is involved each time an r-successors satisfying D_p is merged with an r-successor satisfying D_q. With algebraic reasoning, backtracking is involved each time a distribution of individuals puts r_i-successors and r_j-successors in the same partition (i.e the *ch*-Rule assigns the corresponding variable ≥ 1). Notice however that $\mathsf{C}_{\mathsf{Back}-\mathcal{ALCQ}}$ does not rely on any disjunctive descriptions. This means that non-determinism is due to the *ch*-Rule for algebraic tableau reasoning, and the non-determinism in merging individuals for standard tableau reasoning. Notice that the numbers used are very small, this means that in the case where individuals are distributed over distinct partitions, the *ch*-Rule might perform as many non-deterministic choices as the $\leq$-Rule for merging individuals. Even though handling this source of non-determinism using algebraic tableau reasoning was shown to be more efficient than standard tableau reasoning[9], a parallel framework did not improve the effect of non-determinism due to the *ch*-Rule in this case.

A more complicated test case is considered where disjunctive QCRs are used with disjunctive qualifications.

$$\begin{array}{ll} \mathsf{C}_{\mathsf{Back}-\mathsf{disjunctive}-\mathcal{ALCQ}} & \sqsubseteq \geq (\mathsf{j}+1)\mathsf{r}.\mathsf{D}_1 \sqcap \ldots \sqcap \geq (\mathsf{j}+1)\mathsf{r}.\mathsf{D}_\mathsf{i} \\ & \sqcap \leq \mathsf{jr}.(\mathsf{D}_1 \sqcup \mathsf{D}_2) \sqcup \leq \mathsf{jr}.(\mathsf{D}_2 \sqcup \mathsf{D}_3) \sqcup \ldots \sqcup \leq \mathsf{jr}.(\mathsf{D}_{\mathsf{i}-1} \sqcup \mathsf{D}_\mathsf{i}) \\ \mathsf{D}_\mathsf{q} & \sqsubseteq \neg \mathsf{D}_\mathsf{p} \text{ for all } \mathsf{q} < \mathsf{p} \end{array}$$

Non-determinism in tableau reasoning now has three sources: the *choose*-Rule (or *ch*-Rule for algebraic reasoning), the $\sqcup$-Rule, and non-determinism in merging individuals exceeding the number in the at-most restriction. Since each one of $\sqcup$-Rule and *choose*-Rule rules is applicable to each

created individual, the greater the size of j, the less efficient the reasoning; Figure 16 shows how increasing j with just one number affects performance of the three well known tableau-based DL reasoners; Fact++[6], Hermit[7], and Pellet[8].

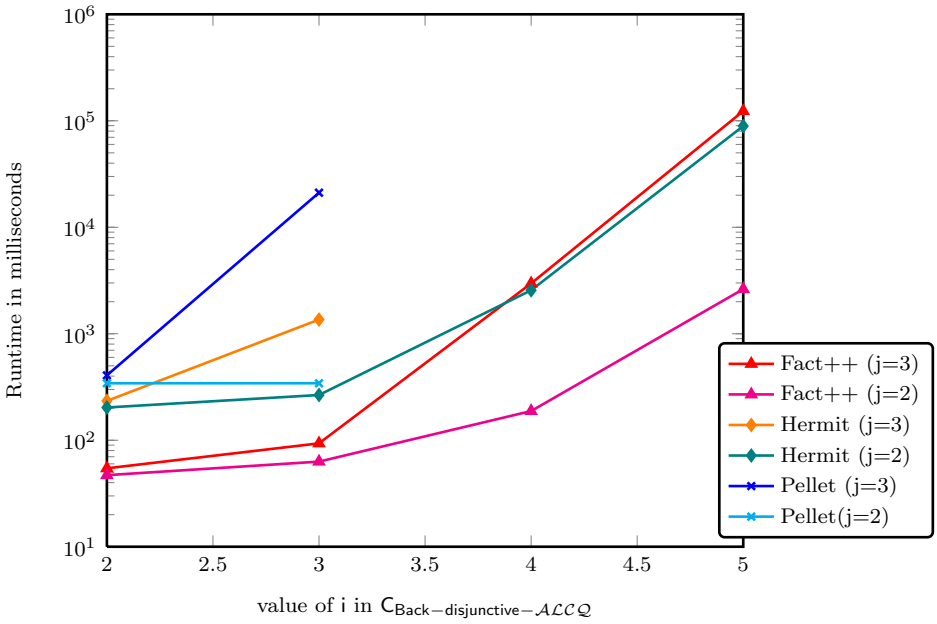


Figure 16: Effect of backtracking with $\mathsf{C}_{\mathsf{Back-disjunctive-}\mathcal{ALCQ}}$.

We focus on the effect of non-determinism and backtracking and run the test with $\mathsf{j} = 3$. The interaction between non-determinism in the *choose*-Rule and the $\sqcup$-Rule blows up the search space of tableau reasoning and the standard DL reasoners time out for $i \geq 2$ (Hermit, Pellet) and $i \geq 4$ (Fact++). Whereas, the algebraic approach allows a better scalability because of two main things: (1) the *ch*-Rule is applied for every variable, which could represent n individuals, which means that the semantic split over the qualifications used in at-most restrictions is done over groups of individuals rather than for each individual as is the case with the *choose*-Rule. Non-Determinism in concept descriptions does not interact with non-determinism in distributing individuals among the restrictions used as qualifications in at-most restriction.

[6]http://owl.man.ac.uk/factplusplus/
[7]http://hermit-reasoner.com
[8]http://clarkparsia.com/pellet

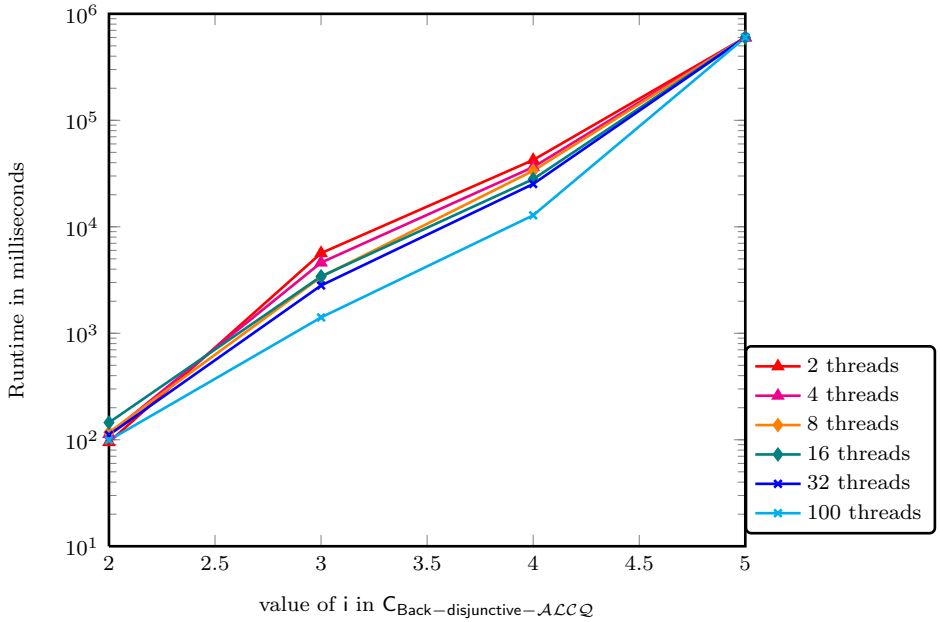


Figure 17: Effect of Non-determinism with $\mathsf{C}_{\mathsf{Back-disjunctive-}\mathcal{ALCQ}}$.

The results are shown in Figure17, due to a Java limitation and memory problems we could only test the cases for $i < 5$. Although HARD performs better than other non-algebraic reasoners, we noticed that the order in which the *ch*-Rule is applied has a dramatic effect on performance even in a parallel execution. Even though the parallel execution always outperformed a sequential execution, we observed runs where HARD was able to return results much faster even for the same number of threads. The difference in run-times was associated with a different order in which the *ch*-Rule was applied because it interacts with the effect of the optimizations responsible for pruning the search space after analyzing clash sources. In fact, the sooner clashes are discovered the more efficient is the pruning of the search space. A similar observation was reported with the sequential version of HARD. It seems as if designing some heuristics to guess the order in which to apply the *ch*-Rule is worth investigation and comparison with respect to a parallel execution.

We also performed test cases where the concept expression is unsatisfiable, either because the set of inequations does not admit a solution or because the logical expressions do not result in a clash free model. The test cases that we used were also from [9]. On average, both versions of HARD (sequential, and parallel) decided the satisfiability of these test cases in less than a second. We plan to extend these test cases in order to test the scalability of the framework illustrated in this paper. A thorough investigation on the speedup, efficiency and overhead of the parallel framework is work in progress. The preliminary results are encouraging and show that using a parallel framework with algebraic reasoning is worth investigating and more promising than parallelizing standard tableau-based reasoning.

6 Related Work

A parallel algorithm for description logics reasoning was first considered in 1995 [7], limited scalability results were possible due to hardware limitations. Further results and research activity have been reported in this area since the work presented in [26], where non-deterministic choices in core satisfiability test were explored concurrently. The DL considered ($\mathcal{SHN}$) did not handle qualified cardinality restrictions and non-determinism was mainly due to disjunctions. A thread pool design was implemented to handle thread management, and tasks were stored in a priority queue. Encouraging results were reported even though no work stealing algorithm between threads was implemented. In [28], a parallel search engine was used to parallelize a lean DL reasoner implementing

tableau rules for the DL $\mathcal{ALC}$. Even though some speedup was reported, it is not clear whether the results could be adopted in a full DL reasoner for $\mathcal{ALC}$. Parallelizing conjunctive expansions ($\sqcap$-Rule) was recently explored in [40], where moderate speedup is reported.

More work can be found on exploring parallel programming in the context of the classification task. Techniques using the MapReduce algorithm to classify $\mathcal{EL}^+$ ontologies [30] and fuzzy $\mathcal{EL}^+$ ontologies [42] have been proposed with no empirical evaluation. Concurrent classification of lightweight ontologies has also been considered in the context of consequence-based reasoning [24] where an impressive speedup was achieved. Tableau-based concurrent classification of more expressive ontologies has been recently reported in [3, 2, 41], where lock-free algorithms with limited synchronization have been used in a multi-core environment, and in [39] where specialized data structures have been proposed to optimize the use of a shared memory environment. However, only less expressive fragments of DL have been considered. Parallel reasoning has also been investigated in the context of distributed resolution reasoning [34] about interlinked ontologies as an alternative to centralized tableau-based reasoning (DL $\mathcal{ALCHIQ}$).

Parallelizing has been considered in [36] by examining a data partitioning approach and a rule partitioning approach. WebPIE [38] is another parallel inference engine which has shown considerable performance results. WebPIE reasoning is based on the MapReduce programming model and is implemented using the Hadoop framework. One might argue that the MapReduce programming model also allows dividing a tasks into subtasks and could be explored in our reasoner. However, MapReduce does not allow task divisions to be done dynamically. This means that the number of tasks must be decided a priori. Another difference between using Fork/Join and MapReduce is related to inter-task communication. Fork/Join allows such communication, in contrast to Map tasks in Hadoop, which are completely independent until the output of Map tasks are reduced. This feature might be crucial to adopt some optimization techniques like dependency directed backtracking.

7 Outlook

We have described our experience in implementing a parallel handling of non-determinism in the algebraic tableau algorithm for the DL $\mathcal{ALCQ}$. We build on the ideas presented in [13] and adopt the Fork/Join parallel execution paradigm to allow the execution of non-deterministic rules on independent cores. This parallel framework allows splitting the tasks of completion graph expansion into parts that are partially evaluated and then combined together to return the final result. Fork/Join allows a divide and conquer style of dividing tasks. This means that the number of tasks created is not determined a priori, it is computed dynamically and recursively as each tasks unfolds.

For this implementation, we support the DL $\mathcal{ALCQ}$. An initial evaluation reveals encouraging results particularly with test cases where the concept expression relies on disjunctive restrictions. We plan to extend this work to handle the full expressivity of the DL $\mathcal{SHOQ}$, where the complexity of reasoning due to the interaction between nominals ($\mathcal{O}$) and qualified cardinality restrictions ($\mathcal{Q}$) might require extra efforts to parallelize the algebraic tableau reasoner. Even though non-determinism remains due to the same tableau expansion rules, the solutions provided by the inequation solver have a global scope within the satisfiability check, whereas, with $\mathcal{ALCQ}$ the solutions provided by the inequation solver are local to a certain node within the completion graph for a given satisfiability check. This might require the handling of communications between tasks running disjunctive expansions, and a more complex implementation.

8 Acknowledgments

We greatly acknowledge the thoughtful feedback by Professor Volker Haarslev on an earlier version of the manuscript. Computational facilities are provided by ACEnet, the regional high performance computing consortium for universities in Atlantic Canada. ACEnet is funded by the Canada Foundation for Innovation (CFI), the Atlantic Canada Opportunities Agency (ACOA), and the provinces of Newfoundland and Labrador, Nova Scotia, and New Brunswick. The second author acknowledges support from NSERC.

References

[1] Juan Esteban Maya Alvarez. Query engine for massive distributed ontologies using mapreduce. Master's thesis, Technische Universitat Hamburg-Harburg, 2010.

[2] Mina Aslani and Volker Haarslev. Parallel tbox classification in description logics - first experimental results. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI 2010 - 19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 485–490, Lisbon, Portugal, August 16-20, 2010. IOS Press.

[3] Mina Aslani and Volker Haarslev. Concurrent classification of owl ontologies - an empirical evaluation. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proceedings of the 2012 International Workshop on Description Logics, DL-2012*, volume 846, Rome, Italy, June 7-10, 2012.

[4] Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From tableaux to automata for description logics. volume 57, pages 247–279, Amsterdam, The Netherlands, The Netherlands, 2003. IOS Press.

[5] Franz Baader and Ulrike Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.

[6] Christopher JO Baker, Rajaraman Kanagasabai, Wee Tiong Ang, Anitha Veeramani, Hong Low, and Markus R Wenk. Towards ontology-driven navigation of the lipid bibliosphere. In *Sixth International Conference on Bioinformatics*, volume 9. BMC Bioinformatics, 2008.

[7] Frank W. Bergmann and J. Joachim Quantz. Parallelizing description logics. In *19th Ann. German Conference on Artificial Intelligence*, LNCS, pages 137–148. Springer-Verlag, 1995.

[8] Yu Ding. *Tableau-based Reasoning for Description Logics with Inverse Roles and Number Restrictions*. PhD thesis, Concordia University, 2008.

[9] Jocelyne Faddoul. *Reasoning Algebraïcally with Description Logics*. PhD thesis, Concordia University, Montreal, Canada, 2011.

[10] Jocelyne Faddoul, Nasim Farsinia, Volker Haarslev, and Ralf Möller. A hybrid tableau algorithm for $\mathcal{ALCQ}$,. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the 21st International Workshop on Description Logics (DL 2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[11] Jocelyne Faddoul and Volker Haarslev. Optimizing algebraic reasoning for description logics with qualified cardinality restrictions and nominals. *Submitted for review*, page 50 pages.

[12] Jocelyne Faddoul and Volker Haarslev. Algebraic tableau reasoning for the description logic $\mathcal{SHOQ}$. *Journal of Applied Logic*, 8(4):334–355, 2010.

[13] Jocelyne Faddoul and Wendy MacCaull. Parallelizing algebraic reasoning for the description logic $\mathcal{SHOQ}$. In *Proceedings of the 4th Canadian Semantic Web Symposium*, pages 20–23. CEUR, 2013.

[14] Jocelyne Faddoul and Wendy MacCaull. A parallel framework for handling non-determinism with expressive description logics. In *28th International Parallel and Distributed Processing Symposium Workshops*. IEEE, 2014.

[15] Eoin Fahy, Shankar Subramaniam, H. Alex Brown, Christopher K. Glass, Alfred H. Merril Jr., Robert C. Murphy, Christian R. H. Raetz, David W. Russell, Yousuke Seyama, Walter Shaw, Takao Shimizu, Friedrich Spener, Gerrit van Meer, Michael S. VanNieuwenhze, Stephen H. White, Joseph L. Witztum, and Edward A. Dennis. A comprehensive classification system for lipids. In *Journal of LIPID research*, volume 46, pages 839–62, 2005.

[16] Nasim Farsiniamarj and Volker Haarslev. Practical reasoning with qualified number restrictions: A hybrid abox calculus for the description logic. *AI Communications*, 23(2-3):205–240, 2010.

[17] Volker Haarslev, Martina Timmann, and Ralf Möller. Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Proceedings of the International Workshop on Description Logics (DL'2001), Aug. 1-3, Stanford, USA*, volume 49 of *CEUR Workshop Proceedings*, pages 152–161, 2001.

[18] Jan Hladik and Jörg Model. Tableau systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, 2004.

[19] Ian Horrocks. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter Basic Description Logics, pages 47–100. Cambridge University Press, 2003.

[20] Ian Horrocks. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter Implementation and optimisation techniques, pages 306–346. Cambridge University Press, 2003.

[21] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.

[22] Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.

[23] Ian Horrocks and Stephan Tobies. Optimisation of terminological reasoning. LTCS-Report LTCS-99-14, LuFG Theoretical Computer Science, RWTH Aachen, 1999.

[24] KazaKov and Simancik. Concurrrent classification of el ontologies. In *10th International Semantic Web Conference*, 2011.

[25] Doug Lea. A java fork/join framework. 2000.

[26] Thorsten Liebig and Felix Müller. Parallelizing tableaux-based description logic reasoning. In *The 2007 OTM Confederated International Conference on the Move to Meaningful Internet Systems*, pages 1135–1144, Berlin, 2007. Springer-Verlag.

[27] Wendy MacCaull and Fazle Rabbi. NOVA Workflow: A Workflow Management Tool Targeting Health Service Delivery. In *International Symposium on Foundations of Health Information Engineering and Systems (FHIES - 2011)*, volume 7151 of *Lecture Notes in Computer Science*, pages 75–92. Springer, 2012.

[28] Adam Meissner. Experimental analysis of some computation rules in a simple parallel reasoning system for the alc description logic. *International Journal of Applied Mathematics and Computer Science - Semantic Knowledge Engineering*, 21(1):83–95, March 2011.

[29] Marvin Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA, 1981.

[30] Raghava Mutharaju, Frederick Maier, and Pascal Hitzler. A MapReduce algorithm for el+. In *23rd International Workshop on Description Logics*, pages 464–474, 2010.

[31] Hans Jürgen Ohlbach and Jana Koehler. Modal logics description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1-2):1–31, 1999.

[32] Laleh Roosta Pour. Algebraïc reasoning with the description logic $\mathcal{SHIQ}$. Master's thesis, Concordia University.

[33] Ross Quillian. *Word concepts: A theory and simulation of some basic semantic capabilities*, volume 12 of *Behavioral Science*, pages 410–430. September 1967.

[34] Anne Schlicht and Heiner Stuckenschmidt. Distributed resolution for expressive ontology networks. In *Web Reasoning and Rule Systems, 3rd International Conference (RR-2009)*, pages 87–101, Chantilly, VA, USA, October 2009.

[35] Manfred Schmidt-Schaub and Gert Smolka. Attributive concept descriptions with complement. *Artificial Intelligence*, 48(1):1–26, 1991.

[36] Ramakrishna Soma and V.K. Prasanna. Parallel inferencing for owl knowledge bases. In *37th International Conference on Parallel Processing - ICPP'08*, pages 75–82, 2008.

[37] Prasad Subramanian. A field programmable gate array based finite-domain constraint solver. Master's thesis, School of Graduate Studies, Utah State University, 2008.

[38] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, and Frank van Harmelen. Webpie: a web-scale parallel inference engine using mapreduce. *Web Semantics: Science, Services and Agents on the World Wide Web*, 10, 2012.

[39] Kejia Wu and Volker Haarslev. A parallel reasoner for the description logic $\mathcal{ALC}$. In Yevgeny Kazakov, Domenico Lembo, and Frank Wolter, editors, *Proceedings of the 2012 International Workshop on Description Logics, DL-2012*, volume 846 of *CEUR Workshop Proceedings*, Rome, Italy, June 7-10, 2012. CEUR-WS.org.

[40] Kejia Wu and Volker Haarslev. Exploring parallelization of conjunctive branches in tableau-based description logic reasoning. In *Proceedings of the 2013 International Workshop on Description Logics*, 2013.

[41] Kejia Wu and Volker Haarslev. Parallel owl reasoning: Merge classification. In *The Third Joint International Semantic Technology Conference*, 2013.

[42] Zhangquan Zhou, Guilin Qi, Chang Liu, Pascal Hitzler, and Raghava Mutharaju. Reasoning with fuzzy-$\mathcal{EL}^+$ ontologies using mapreduce. In Luc De Raedt et al., editor, *ECAI 2012 - 21st European Conference on Artificial Intelligence*, pages 933–934. IOS Press, 2012.