Distributed Resolution for Expressive Ontology Networks

Anne Schlicht, Heiner Stuckenschmidt

Knowledge Representation and Knowledge Management Research Group
Computer Science Institute
University of Mannheim
{anne, heiner}@informatik.uni-mannheim.de

Abstract. The Semantic Web is commonly perceived as a web of partially interlinked machine readable data. This data is inherently distributed and resembles the structure of the web in terms of resources being provided by different parties at different physical locations. A number of infrastructures for storing and querying distributed semantic web data, primarily encoded in RDF have been developed but almost all the work on description logic reasoning as a basis for implementing inference in the Web Ontology Language OWL still assumes a centralized approach where the complete terminology has to be present on a single system and all inference steps are carried out on this system.
We propose a distributed reasoning method that preserves soundness and completeness of reasoning under the original OWL import semantics. The method is based on resolution methods for $\mathcal{ALCHIQ}$ ontologies that we modify to work in a distributed setting. Results show a promising runtime decrease compared to centralized reasoning and indicate that benefits from parallel computation trade off the overhead caused by communication between the local reasoners.

1 Introduction

Almost all the work on description logic reasoning as a basis for implementing inference in the Web Ontology Language OWL still assumes a centralized approach where the complete terminology has to be present on a single system and all inference steps are carried out on this system. This approach has a number of severe drawbacks. First of all, the complete, possibly very large data sets have to be transferred to the central reasoning system creating a lot of network traffic. Furthermore, transferring the complete models to a single reasoner also makes this a major bottleneck in the system. This can go as far as reaching the limit of processable data of the reasoning system. A number of approaches for distributed reasoning about interlinked ontologies have been proposed that do not require the models to be sent to a central reasoner [5, 7, 10]. All these approaches rely on strong restrictions on the types of links between ontologies or the way concepts defined in another ontology may be used and refined and thus introduce special kinds of links between data sets stored in different locations. In particular, none of these approaches supports the standard definition of logical import from the OWL specification, limiting their usefulness on real data sets. We illustrate these problems using a small example.

Example 1. We assume the two small ontologies depicted below are connected by an owl:imports statement in ontology B.

Ontology A	Ontology B
$A{:}Car \sqsubseteq A{:}Vehicle$	$B{:}HybridCar \sqsubseteq A{:}Vehicle$
$A{:}Car \sqsubseteq \exists_{\leq 1} A{:}hasEngine$	$B{:}HybridCar \sqsubseteq \exists_{\geq 2} A{:}hasEngine$

As we can easily see, $A{:}Car \sqsubseteq \neg B{:}HybridCar$, and hence adding the assertion "$A{:}Car \sqcap B{:}HybridCar(a)$" would yield an inconsistency.

Our goal is to have a distributed reasoning method that performs local reasoning on the two ontologies and that is still able to detect the inconsistency. Looking at the previous proposals for distributed reasoning mentioned above, we notice that none of them meets these requirements. The framework of ϵ-connections[7] does not apply in this scenario as it does not allow the specification of subsumption relationships between interlinked ontologies. Using the framework of conservative extensions[10] does not provide any advantages in terms of local reasoning. In particular, the overall model is neither a conservative extension of ontology A nor of ontology B as in both cases, the additional information in the other parts can be used to derive new information concerning the signature of ontology A or ontology B, respectively. Encoding the ontology network in distributed description logics, finally, the domains of A and B are disjoint by definition and the assertion "$A{:}Car \sqcap B{:}HybridCar(a)$" is not expressible.

Note that a set of ontologies linked by mapping axioms can also be represented in terms of OWL imports. In this case, the mapping axioms would be part of any of the two ontologies and and the other ontology would be imported by the one containing the mapping axioms.
Our aim is to develop a method for reasoning about description logic ontologies that overcomes the disadvantages of existing methods. We have designed and implemented a distributed reasoning method that 1) preserves soundness and completeness of reasoning under the original OWL import semantics 2) avoids restrictions on the use of definitions from remote models in local definitions or on the way knowledge is distributed a priori 3) decreases runtime by parallel computation, trading off the overhead caused by communication.

In previous work [15] we proposed a distributed resolution method for $\mathcal{ALC}$. The extension to $\mathcal{ALCHIQ}$ is complex because the ordered resolution calculus we used for $\mathcal{ALC}$ cannot handle the equality literals introduced by number restrictions. A resolution calculus that decides $\mathcal{ALCHIQ}$ is much more sophisticated and requires a more involved strategy for exchanging axioms between reasoning peers. The paper is structured as follows: In the next section we address the distribution principles and distributed resolution in general. Section 3 reviews the idea proposed in [15] and presents the details of our distributed reasoning method. In Section 4 we investigate the properties of the method with respect to number of derivations, communication effort and degree of parallelization and show that these parameters are promising.

2 Distributing Logical Resolution

2.1 Distribution Principles

There are various options for distributing the process of logical reasoning. Many of these options have been investigated in the field of automated theorem proving for first-order logics [4, 3]. In the following we discuss these options and their pros and cons with respect to the requirements and goals defined in the introduction. In particular, we have to make two choices:

1. We have to choose a reasoning method that is sound and complete for description logics and permits distribution.
2. We have to choose a distribution principle that supports local reasoning and minimizes reasoning and communication costs.

Concerning the reasoning method, analytic tableaux are the dominant method for implementing sound and complete inference systems for description logics [8]. It has been shown, however, that sound and complete resolution methods for expressive description logics can be defined [16, 9]. We exclude other existing methods such as a reduction of DL reasoning to logic programming from our investigation because these approaches are not sound and complete for the languages we are interested in. Because tableaux-based as well as resolution-based methods meet our requirements with respect to language coverage and completeness, the decisive factor is their suitability for distributed reasoning.
The survey [3] discusses different strategies for parallelizing logical inference. In particular, the authors distinguish between parallelism at the term-, clause and the search level where paralellism at the search level is further distinguished into multi-search and distributed search approaches. Parallelism at the term- and clause level is not suitable for our purposes as it speeds up basic reasoning functions such as matching or unification using a shared memory. The idea of multi-search approaches is to try different heuristics or starting points in parallel and require the complete logical model to be available to all reasoners. The distributed search paradigm naturally fits the distributed storage of parts of the model and therefore represents a paradigm that fits the goals of our research as it allows to assign the part of the search space relevant for a specific model to a local reasoner instance that interacts with other local reasoners if necessary. The choice of the distributed search paradigm has consequences for the choice of the reasoning method. In particular, it has been shown that distributed search can be used in combination with ordering-based methods [6, 2] to support parallel execution of logical reasoning. We build on top of these results by proposing distributed reasoning methods based on the principles of resolution. Our proposal extends beyond the state of the art in distributed theorem proving as it addresses specific decidable subsets of first-order logics that have not yet been investigated in the context of distributed theorem proving. Furthermore, existing strategies for assigning inference steps to reasoners such as the ancestor-graph criterion [2] cannot avoid redundancy. We propose a method based on ordered resolution that takes advantage of the special structure of clauses in the description logic $\mathcal{ALCHIQ}$ for efficiently deciding satisfiability in a distributed setting.

2.2 Resolution Theorem Proving

Before describing our distributed resolution method for ontologies, we first briefly review standard resolution reasoning and present the basic idea for distributed resolution. Resolution is a very popular reasoning method for first order logic (FOL) provers. As description logics are a strict subset of first order logic, resolution can be applied to description logic ontologies as well [17]. For this purpose the DL ontology is transformed into a set of first order clauses as defined in Section 3.2. This translation can be done on a per axiom basis independently of other parts of the model. It can be shown that the ontology is satisfiable if and only if the set of clauses is satisfiable. The set of clauses is satisfiable iff exhaustive application of the rule standard resolution with factoring does not derive an empty clause.

Definition 1 (Standard Resolution). *For clauses C and D and literals A and $\neg B$, standard resolution with factoring is defined by the rule*

$$\textit{Standard Resolution with Factoring}\quad \frac{C \vee A_1 \vee \cdots \vee A_n \qquad D \vee \neg B}{C\sigma \vee D\sigma}$$

where the substitution σ is the most general unifier of $A_1, \ldots, A_n$ and B.

2.3 Distributed Resolution

The implementation of a resolution algorithm is described in [18]. For an input set of clauses, it systematically applies resolution rules to appropriate pairs of clauses and adds the derived new clauses to the clause set. If an empty clause is derived or no new and non-redundant clause can be derived, the algorithm terminates. An essential part of a resolution prover and the most time consuming component [18] are reduction rules that delete clauses that are not necessary for the decision process. Without reduction, the number of clauses generally increases infinitely and it may be impossible to saturate even a small set of clauses.
As described in [15], a standard resolution algorithm can be modified to support distributed reasoning. In particular, the inferences can be distributed across different reasoners by separating the set of input clauses and running provers on separate parts of the set:

- Every reasoner separately saturates the clause set assigned to it.
- Newly derived clauses are propagated to other reasoners if necessary.

Instead of adding every clause that is derived to the local set of clauses, some new clauses are propagated to other reasoners and deleted locally.

Definition 2 (Allocation). *An* allocation *for a set $\mathcal{C}$ of clauses and a set of ontology modules M is a relation $a \in (\mathcal{C} \times M)$ such that*

$$\forall c \in \mathcal{C} \colon \exists m \in M \colon a(c, m)$$

The set of modules a clause c is allocated to by the allocation a is defined by

$$a(c) := \{m \in M \mid a(c, m)\}$$

If the allocation relation is functional we may omit the parenthesis and write $a(c) = m$.

In addition to the propagation of clauses we have to add a second modification to the algorithm to turn it into a distributed resolution algorithm. In contrast to the centralized case, a reasoner that has saturated the local clause set may have to continue reasoning once a new clause is received from another reasoner. The whole system of connected reasoners stops if the empty clause is derived by one of the reasoners or all are saturated. After this intuitive description of a distributed resolution algorithm, we define distributed resolution formally:

Definition 3 (Distributed Resolution Calculus). *A* distributed resolution calculus $R(a)$ *is a resolution calculus that depends on an allocation relation* $a\colon \mathcal{C} \to M$ *such that each rule* r *of* $R(a)$ *is restricted to premises* $P \subset \mathcal{C}$ *with*

$$\exists m \in M\colon \forall c \in P\colon a(c, m)$$

We call this restriction allocation restriction*.*

Hence, the rules of a distributed resolution calculus are restricted to premises allocated to the same module. A distributed calculus can be obtained from any resolution calculus by defining an allocation relation and adding the allocation restriction to each rule of the calculus.
Obviously, termination of the underlying calculus is preserved by distribution if it does not depend on reduction rules. In the worst case, each inference of the original calculus is performed once in every module of the distributed calculus. The results presented in Section 4 also indicate that local reduction (i.e. deleting clauses that are redundant with respect to the reasoner they are processed by) is sufficient in practice.
Preserving completeness without allocating each clause to every reasoner is more difficult, we have to make sure the allocation restriction never excludes inferences that are possible in the original calculus. For standard resolution, a given clause C has to be propagated to any reasoner whose clause set contains a clause with a literal that matches (i.e. is unifiable and of opposite polarity) any of the literals in C. This would lead to a substantial communication overhead and potentially redundant inference steps.
To avoid redundancy, we aim at allocating every clause to only a single reasoner. A functional allocation guarantees that the same resolution step is never carried out twice, because equivalent clauses are always assigned to the same unique reasoner which takes care of avoiding local redundancy.

3 Distributed Resolution for Description Logic

As we have seen above, the ability to define a sound and complete distributed reasoning method relies on two requirements: (1) the existence of a sound and complete resolution calculus and (2) the ability to find a corresponding allocation that satisfies the allocation restriction. In this section, we show that for the case of ontologies defined in $\mathcal{ALCHIQ}$ both of these requirements can be satisfied leading to a sound and complete distributed resolution method. We do not address reduction rules in this section because reduction is not necessary to guarantee the theoretical properties of the proposed calculus. However, for efficient reasoning reduction is essential and hence the practical effects distribution has on reduction are discussed in the experimental section.

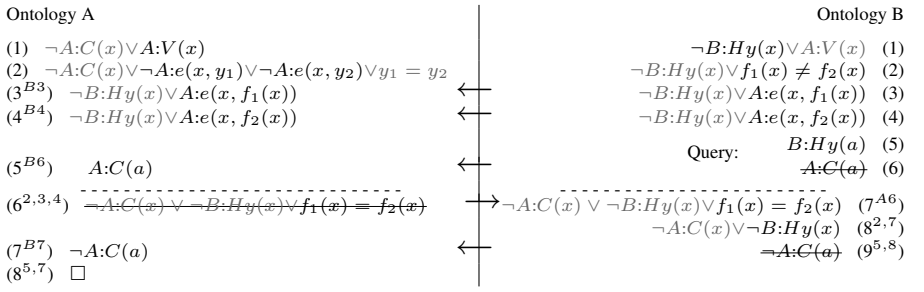

Fig. 1. Distributed refutation example. The designer of ontology B from Example 1 wants to check satisfiability of the concept "$B{:}HybridCar \sqcap A{:}Car$" and adds the the appropriate query. Since the concept is unsatisfiable, an empty clause is derived. Predicates are abbreviated to simplify presentation, derived clauses are noted below the dashed line, arrows denote propagation of a clause. Literals that are not resolvable literals are grayed out (assuming predicates from B precede predicates from A and $A{:}V > A{:}C$). Clauses that are striked out are locally deleted on propagation.

3.1 Distribution Principle

The idea for our distributed reasoning approach is to take advantage of the restrictions description logic imposes on first order logic. In particular, we identify a property that holds for many efficient resolution calculi and use it for defining a distribution principle. The important property of a resolution calculus is, that each clause contains only one resolvable literal. I.e. for every possible inference the resolvable literal (or a subterm of it) is unified with a literal of another premise, the other literals are (possible with substituted variables) passed to the conclusion. Formally, the resolvable literal and its uniqueness are defined as follows:

Definition 4 (Resolvable Literal). *A literal* lit *of a clause $C \vee lit$ is a* resolvable literal *of $C \vee lit$ with respect to a calculus R and logical language L iff there is a clause $D \vee lit' \in L$, such that R can be applied to the premises $C \vee lit$ and $D \vee lit'$ deriving the clause $(C \vee D \vee lit'')\sigma$ with appropriate substitution σ and literal lit''*[1].

In standard resolution all literals of a clause are resolvable literals, but more advanced calculi restrict the applicability of resolution rules such that there is only one resolvable literal in each clause. In particular, for the ordered resolution calculus defined in [11] for $\mathcal{ALC}$ description logic, each clause contains an unique resolvable literal [15]. Based on the uniqueness of the resolvable literal and an allocation of symbols to reasoners we can define an allocation function that allocates every clause to one module of the networked ontology. Note that for ontologies linked by import statements, the namespaces define the allocation of symbols. For distributed reasoning on a single ontology, the ontology is first partitioned into linked modules as described in [14].

[1] For ordered resolution lit'' is false and may be omitted.

Definition 5 (Allocation $\mathbf{a(c)}$).

$$a(c) := \{alloc(topSymbol(lit)) \mid lit \textit{ is resolvable literal of } c\}$$

where topSymbol of a possibly negated predicate literal $(\neg)P(t_1, ..., t_n)$ *is* P.

If all clauses have unique resolvable literals, then the allocation a is functional, too. For determining where (and if) a derived clauses is propagated, we first pick the unique resolvable literal of the clause, then the top symbol of this literal and finally the reasoner this symbol is allocated to. If a symbol s is allocated to a module m we say that module m is *responsible* for s. Figure 1 illustrates distributed resolution on the ontologies from Example 1. In [15] we showed that ordered resolution with this allocation function is a complete distributed method for deciding $\mathcal{ALC}$ satisfiability because the inferences are the same for distributed and centralized resolution. However, for supporting more expressive ontologies, a more complex calculus is required and the allocation function defined above is not sufficient to guarantee completeness.

3.2 Preliminaries

Before presenting the calculus our distributed method is based on, we define the description logic we use and the translation of description logic axioms to first order clauses.

The Description Logic $\mathcal{SHIQ}$ A $\mathcal{SHIQ}$ ontology is a set $\mathcal{O}$ of axioms α of the following syntax in BNF:

$$\begin{aligned}
\alpha ::= & C \sqsubseteq C | C \equiv C | C(x) | R(x,x) & n ::= & number \\
& | Trans(R) | R \sqsubseteq R & A ::= & concept_name \\
C ::= & \top | \bot | A | \neg C | C \sqcap C | \exists R.C | \forall R.C & R ::= & role_name | \text{Inv}(role_name) \\
& | \exists_{\leq n} R.C | \exists_{\geq n} R.C & x ::= & individual_name
\end{aligned}$$

The *signature* of an ontology $Sig(\mathcal{O})$ is the disjoint union of concept names, role names and individual names.

Normalization The resolution calculus we apply requires first order clauses as input, hence the first order formulas obtained from an ontology are translated to clauses. To guarantee termination of the applied resolution calculus, the ontology has to be normalized prior to clausification. This ensures that only certain types of axioms and corresponding clauses occur in the reasoning procedure. For simplicity, we assume the ontology contains only subsumption axioms $A \sqsubseteq C$ where A is not a complex concept and no equivalence axioms. Complex subsumptions $C \sqsubseteq D$ are equivalent to $\top \sqsubseteq \neg C \sqcup D$ and equivalences $C \equiv D$ can be replaced by two subsumptions $C \sqsubseteq D$ and $D \sqsubseteq C$. The definitorial form normalization we use replaces complex concepts C in the right hand side of an axiom by a new concept name A and adds the axiom $A \sqsubseteq C$ to the ontology. Thus, it splits up nested axioms into simple ones by introducing new concepts.

Definition 6 (Definitorial Form). *For simple subsumptions $A \sqsubseteq D$ with atomic concept A the Definitorial Form is defined by*

$$\mathit{Def}(A \sqsubseteq D) := \begin{cases} \{A \sqsubseteq D\} & \text{if all subterms of } D \text{ are literal concepts} \\ \{Q \sqsubseteq D|_p\} \cup \mathit{Def}(A \sqsubseteq D[Q]_p) & \text{if } D|_p \text{ is not a literal concept} \end{cases}$$

where $D|_p$ denotes a certain[2] *subterm of D and $D[Q]_p$ is the term obtained by replacing this subterm with Q.*

Clausification After normalization, the ontology contains only simple axioms that can be translated to first order clauses as follows:

$A \sqsubseteq B$	$\neg A(x) \vee B(x)$	$A \sqsubseteq \exists_{\leq n} r.B$	$\neg A(x) \vee \neg r(x, y_i) \vee y_i = y_j \vee \neg B(y_i)$ $i = 1..n+1 \quad j = 1..i-1$
$A \sqsubseteq B \sqcap C$	$\neg A(x) \vee B(x)$		
	$\neg A(x) \vee C(x)$	$A \sqsubseteq \exists_{\geq n} r.B$	$\neg A(x) \vee r(x, f_i(x)) \quad i = 1..n$
$A \sqsubseteq B \sqcup C$	$\neg A(x) \vee B(x) \vee C(x)$		$\neg A(x) \vee f_i(x) \neq f_j(x) \quad j = 1..i-1$
$A \sqsubseteq \exists r.B$	$\neg A(x) \vee r(x, f(x))$		$\neg A(x) \vee B(f_i(x))$
	$\neg A(x) \vee B(f(x))$	$r \sqsubseteq s$	$\neg r(x, y) \vee s(x, y)$
$A \sqsubseteq \forall r.B$	$\neg A(x) \vee \neg r(x, y) \vee B(y)$	$r \equiv Inv(s)$	$\neg r(x, y) \vee s(y, x)$
			$\neg s(x, y) \vee r(y, x)$

The clauses resulting from the ontologies of Example 1 are depicted in Figure 1.

3.3 Distributed Resolution for $\mathcal{ALCHIQ}$

When an ontology contains qualified($\mathcal{Q}$) or unqualified($\mathcal{N}$) number restrictions or functional properties ($\mathcal{F}$), the translation to clauses contains equalities. To deal with these equalities, a much more sophisticated calculus than ordered resolution is required which in turn requires a more involved allocation of clauses to ontology modules. Before we present the necessary adaptions and extensions to the distributed resolution method, we briefly describe the calculus our method is based on. The DL expressivity that can be covered with this calculus is $\mathcal{ALCHIQ}^-$ which is $\mathcal{SHIQ}$ without transitive properties with the additional restriction that number restrictions are only allowed on roles that do not have subroles. Extension of the method for supporting $\mathcal{SHOIQ}(\mathcal{D})$ is discussed in the next subsection.

Resolution Calculus for $\mathcal{ALCHIQ}^-$ A complete calculus that terminates on clauses obtained from ontologies that contain number restrictions is *basic superposition* [1, 9], an extension of ordered resolution. Like ordered resolution, basic superposition uses two parameters, a *selection function* and *ordering of literals* that restrict applicability of the resolution rules.
As usual for theories containing equalities, we assume a translation of predicates to general function symbols such that all literals are equalities (e.g. the literal $P(x)$ translates

[2] The exact definition of $|_p$ (*position*) is not relevant in this paper, please refer to [11] for detail.

to $P(x) \approx \top$), we may still write $P(x)$ for readability purpose and call these literals *predicate literals*. Clauses are split into skeleton clause C and substitution σ representing all substitutions introduced by previous unifications. The clause $C\sigma$ is denoted as closure $C \cdot \sigma$ or alternatively a closure is denoted by enclosing non-variable subterms of $C\sigma$ that correspond to variables in C in brackets (e.g. $P([f(y)])$ for $P(x) \cdot \{x \mapsto f(y)\}$). For distributing basic superposition, the rules we have to take care of are positive and negative superposition, the other rules contain only one premise and hence distribution of the input clauses into separate sets does not restrict application of these rules.

Definition 7 (Superposition).

$$\textit{Positive superposition} \quad \frac{(C \vee s \approx t) \cdot \rho \quad D \vee (w \approx v) \cdot \rho}{(C \vee D \vee w[t]_p \approx v) \cdot \theta}$$

where

1. σ *is the most general unifier of* $s\rho$ *and* $w\rho|_p$ *and* $\theta = \rho\sigma$
2. $t\theta \not\succeq s\theta$ *and* $v\theta \not\succeq w\theta$
3. *in* $(C \vee s \approx t) \cdot \theta$ *nothing is selected and* $(s \approx t) \cdot \theta$ *is strictly maximal*
4. *in* $D \vee (w \approx v) \cdot \theta$ *nothing is selected and* $(w \approx v) \cdot \theta$ *is strictly maximal*
5. $w|_p$ *is not a variable.*
6. $s\theta \approx t\theta \not\succeq w\theta \approx v\theta$

$$\textit{Negative superposition} \quad \frac{(C \vee s \approx t) \cdot \rho \quad D \vee (w \not\approx v) \cdot \rho}{(C \vee D \vee w[t]_p \not\approx v) \cdot \theta}$$

where

1. σ *is the most general unifier of* $s\rho$ *and* $w\rho|_p$ *and* $\theta = \rho\sigma$
2. $t\theta \not\succeq s\theta$ *and* $v\theta \not\succeq w\theta$
3. *in* $(C \vee s \approx t) \cdot \theta$ *nothing is selected and* $(s \approx t) \cdot \theta$ *is strictly maximal*
4. $(w \not\approx v) \cdot \theta$ *is selected or maximal and no other literal is selected in* $D \vee (w \not\approx v) \cdot \theta$
5. $w|_p$ *is not a variable.*

In addition to the superposition rules, two rules with only one premise are necessary to deal with equalities (see [9] for details). Ordered resolution is a special case of positive superposition, where $w|_p = w$, i.e. p is the root position. A sequence of ordered resolution inferences can be combined into a *ordered hyperresolution* inference by deleting intermediate conclusions. We assume that ordered hyperresolution and not ordered resolution is applied to clauses containing multiple resolvable literals (see derivation of clause A6 in Figure 1).

Definition 8 (Resolution Calculus $R_\mathcal{Q}$ [9]).
$R_\mathcal{Q}$ is the calculus with 1) rules positive and negative superposition, reflexivity resolution and equality factoring, 2) selection of every negative binary literal, 3) the term ordering $\succ_Q$ is a lexicographic path ordering (LPO, [12]) based on a total precedence $>$ of function, constant and predicate symbols with $f > c > P > \top$ for every function f constant c and predicate P.

Literals containing different variables are $\succ_Q$-incomparable because otherwise the ordering would depend on the substitution. Literals that contain a function symbol are ordered first to avoid substituting the arguments of functions with function terms. Limited nesting depth of literal terms is necessary to guarantee termination of the calculus, it makes sure only the types of clauses depicted in Table 1 occur when basic superposition is applied to clauses obtained from an $\mathcal{ALCHIQ}$ ontology (i.e. the set of $\mathcal{ALCHIQ}$ closures is closed under basic superposition).

1	$\neg R(x,y) \vee Inv(R)(x,y)$
2	$\neg R(x,y) \vee S(x,y)$
3	$\mathbf{P^f}(x) \vee R(x, \langle f(x) \rangle)$
4	$\mathbf{P^f}(x) \vee R([f(x)], x)$
5	$\mathbf{P_1}(x) \vee \mathbf{P_2}(\langle \mathbf{f}(x) \rangle) \vee \bigvee \langle f_i(x) \rangle \approx / \not\approx \langle f_j(x) \rangle$
6	$\mathbf{P_1}(x) \vee \mathbf{P_2}([g(x)]) \vee \mathbf{P_3}(\langle \mathbf{f}[g(x)] \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$
7	$\mathbf{P_1}(x) \vee \bigvee_{i=1}^{n} \neg R(x, y_i) \bigvee_{i=1}^{n} \mathbf{P_2}(y_i) \vee \bigvee_{i=1}^{n} \overset{n}{}_{j=i+1} y_i \approx y_j$
8	$\mathbf{R}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle) \vee \mathbf{P}(\langle \mathbf{t} \rangle) \vee \bigvee \langle t_i \rangle \approx / \not\approx \langle t_j \rangle$

$\mathbf{P}(t)$, where t is a term, denotes a possibly empty disjunction of the form $(\neg)P_1(t) \vee \cdots \vee (\neg)P_n(t)$. $\mathbf{P}(\mathbf{f}(x))$ denotes a disjunction of the form $\mathbf{P_1}(f_1(x)) \vee \cdots \vee \mathbf{P_m}(f_m(x))$. Note that this definition allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals. $\langle t \rangle$ denotes that term t may but need not be marked (i.e. has been introduced by a previous unification), $\approx / \not\approx$ denotes a positive or negative equality predicate. For clauses of type 6 t_i and t_j are either of the form $f([g(x)])$ or of the form x and the clause contains at least one term $f(g(x))$.

Table 1. The 8 types of $\mathcal{ALCHIQ}$ closures [9]

Because the set of clause types is finite and the set of symbols is finite for every given ontology, the number of clauses that can be derived is finite, too and hence basic superposition terminates for $\mathcal{ALCHIQ}$ input[9].

Allocation for $\mathcal{ALCHIQ}$ The first consideration for distributing basic superposition is the number of resolvable literals. A close look to Definition 7 reveals that the resolvable literals $s \approx t \cdot \rho$ and $w \approx / \not\approx v \cdot \rho$ are necessarily either selected or maximal. Furthermore, the $\mathcal{ALCHIQ}$ closures of types 3-6 and 8 are totally ordered and types 1 and 2 contain exactly one selected literal. Only closures of type 7 may contain multiple resolvable literals, but since all resolvable literals have the same top symbol, the allocation from Definition 5 is still functional. Before allocating $\mathcal{ALCHIQ}$ closures, we have to extend the definition of *topSymbol* to equality literals:

Definition 9 (Top Symbol).
The top Symbol of an equality literal $f(t_1) \approx / \not\approx g(t_2)$ is f if $f(t_1) \succ g(t_2)$.

Note that arguments of equalities that are resolvable literal of an $\mathcal{ALCHIQ}$ clause are always comparable. With this extended definition, we could use the allocation function a for distributed resolution on $\mathcal{ALCHIQ}$. Unfortunately, this calculus would not be complete, because some inferences of basic superposition are prevented in the dis-

tributed setting. We illustrate this problem on an example inference:

$$\text{Positive superposition} \quad \frac{C \vee f(x) \approx g(x) \qquad D \vee P(f(y))}{C \vee D \vee P(g(x))}$$

Here, $f(x)$ is unified with $f(y)$ but f is not the top symbol of the resolvable literal $P(f(y))$. Hence, if f and P are allocated to different ontology modules, the rule is not applied and the clause $C \vee D \vee P(g(x))$ is missing in the reasoning process. Due to superposition of equalities into predicate literals, we have to extend the allocation to guarantee completeness of the decision procedure.

Definition 10 (Allocation for $\mathcal{ALCHIQ}$). *The clause allocation $a_+(c)$ for the distributed calculus $R_\mathcal{Q}(a_+)$ is defined by $a_+(c) := a(c) \cup a_f(c)$ with*

$$a_f(c) := \{alloc(funSymbol(lit)) \mid lit \text{ is resolvable literal of } c\}$$

where

- *$funSymbol(lit) := f$ for every literal $lit = (\neg)P(f(t))$ or $lit = (\neg)P(f(x), x)$ or $lit = (\neg)P(x, f(x))$ with unary or binary predicate symbol P. For other literals $funSymbol(lit)$ is null.*
- *$alloc\colon Sig(\mathcal{O}) \rightarrow M$ is an allocation of the signature symbols of the input ontology $\mathcal{O}$, including concepts introduced by the definitorial form transformation. $alloc(null) := \emptyset$*

Note that resolvable literals of closures of type 7 never contain function symbols, hence for all $\mathcal{ALCHIQ}$ closures c the allocation $a_f(c)$ is a function and the set $a_+(c)$ consists of at most two modules. The allocation a_+ solves the problem of the example depicted above. But, it remains to be proved that no other pair of premises that could be resolved in a basic superposition inference is allocated to different ontology modules and hence completeness of the calculus for $\mathcal{ALCHIQ}$ is preserved by distribution.

Theorem 1 (Completeness of Distributed Resolution for $\mathcal{ALCHIQ}$). *The distributed resolution calculus $R_\mathcal{Q}(a_+)$ decides $\mathcal{ALCHIQ}$ satisfiability.*

Since $R_\mathcal{Q}$ decides $\mathcal{ALCHIQ}$, it remains to be shown that every inference in the original calculus is performed in the distributed calculus, too. Let us first consider superposition into root position (i.e. $w|_p = w$). In this case, basic superposition is equivalent to ordered resolution, both premises are allocated to the same module because the two resolvable literals have the same top symbol. Superposition at other positions is only possible for function equalities into predicate literals i.e. $s \approx t$ is an equality literal $f(x) \approx/\not\approx g(x)$ and $w \approx/\not\approx v$ is a predicate literal $(\neg)P(f(x))$ or $(\neg)R(x, f(x))$. Variable equations are never selected or maximal and hence no resolvable literals. If $s \approx t$ is a predicate literal or $w \approx/\not\approx v$ is an equality, superposition is only possible at root position, otherwise unification is impossible or w would be a variable which is not allowed according to Definition 7.

Hence, for every application of a rule in R_Q, the allocation a_+ ensures all premises meet in one module. A clause is allocated to at most two modules, local saturation of

the local clause sets is enough to guarantee completeness of the method. Note that only clauses are duplicated, duplication of inferences can be avoided by restricting basic superposition such that only the module responsible for the top symbol of $s \approx t$ in Definition 7 performs the inference.

3.4 Extension to $\mathcal{SHOIQ}(\mathcal{D})$

The expressivity that can be handled by basic superposition while guaranteeing termination is $\mathcal{ALCHIQ}^-$ which is $\mathcal{ALCQ}$ plus role hierarchies and inverse roles, with the restriction that number restrictions are only allowed on roles that do not have subroles. For extending the expressivity to $\mathcal{ALCHIQ}$ the *decomposition rule* has to be added to the calculus[9]. Decomposition is an reduction rule that is applied to newly derived clauses eagerly. Since the decomposition rule has only one premise, it can be added to our approach without restricting the possible inferences.
Transitivity axioms contained in a $\mathcal{SHIQ}$ ontology can be eliminated by a well known transformation, reducing the expressivity to $\mathcal{ALCHIQ}$. Hence, with some preprocessing we can decide satisfiability of a $\mathcal{SHIQ}$ ontology. The transformation is polynomial in the size of the input, but the adaption to the distributed setting is not trivial. In contrast to the transformation of description logic axioms to first order clauses mentioned so far, the translation of transitivity depends on the whole ontology and not only on the transitivity axioms. Hence, the linked ontologies cannot be transformed independently.
Nominals are concepts with a single instance, e.g. the concept $\{$*Erdös*$\}$ with instance *Erdös* is used in the concept description $\exists coauthorOf\{$*Erdös*$\}$. Nominals are replaced by common concepts for many applications: Each nominal $\{nom\}$ is replaced by a new concept Nom and the axiom $Nom(nom)$ is added to the ontology. The restriction that Nom may not contain another instance is not expressible in description logic without nominals. However, it can be expressed by the first order clause $\neg Nom(x) \vee x = nom$.
Datatypes $(\mathcal{D})$ can be eliminated without changing the semantics by moving the datatypes into the abstract domain. In practice, sorts (datatype and abstract) are handled different from the other predicates to speed up reasoning. Built-in datatype predicates can be added to support e.g. the *greater* relation between integers.

4 Experiments

Our distributed resolution implementation is based on the first order prover SPASS[3] [19]. A number of different resolution strategies including ordered resolution and basic superposition are supported, precedence and selection are specified in the input file. We implemented definitorial form normalization and clausification in a separate tool. Clauses are stored in separate files for each ontology and include precedence and selection in every input file. Apart from compliance to the requirements of $R_{\mathcal{Q}}$ (Definition 8) the precedence was random. The applied reduction rules include forward and backward subsumption reduction[4]. For turning SPASS into a distributed reasoner (i.e. adding the

[3] http://www.spass-prover.org

[4] The complete configuration for Spass is: Distributed=1 Auto=0 Splits=0 Ordering=1 Sorts=0 Select=3 FullRed=1 IORe=1 IOFc=1 IEmS=0 ISoR=0 IOHy=0 RFSub=1 RBSub=1 RInput=0 RSSi=0 RObv=1 RCon=1 RTaut=1 RUnC=1 RSST=0 RBMRR=1 RFMRR=1

Query	# Parts	Runtime/ms	# Derivations	# Propagations	Busy Factor
Satisfiability	1	230	608	-	100%
	13	146	610	432	20%
Subsumption	1	132	154	-	100%
(positive)	13	36	578	252	60%

Table 2. Results of tests on the chem ontology from SWEET project.

"Distributed" option) we added support for sending and receiving clauses. A set of received clauses is treated like a set derived from a given clause, i.e. it is forward and backward reduced with respect to the local worked off clause list before adding the non redundant received clauses to the usable list. All reasoners are connected at startup, clause communication is performed in separate processes to avoid the local reasoning being blocked on sending a clause. The priority of the reasoning and communication processes is adjusted such that while not saturated locally, a clause is only send to another reasoner if this destination reasoner messages that it is idle. New clauses are only received when the local clause set is completely saturated. Startup and shutdown of the system is initialized by a central control process. In a fully decentralized P2P system this job is performed by the peer that receives a query. The control process starts the separate machines on their respective input clauses files. Apart from passing clauses between each other, the reasoners send status messages whenever they are locally saturated, when they continue reasoning on newly received clauses and when they derive an empty clause. When one reasoner finds a proof or all reasoners are saturated for an interval longer than the maximal time necessary for clause propagation the query is answered and the reasoners are shut down.

Dataset Our implementation is tested on the Semantic Web for Earth and Environmental Terminology (SWEET [13]), a set of linked ontologies published by the NASA Jet Propulsion Laboratory. We used the chemical ontology chem[5] and the ontologies that are directly or indirectly imported by chem. In total, our dataset consists of 13 ontologies liked by 34 import statements. The ontology network describes 480 classes and 99 individuals, translation to first order logic yields 930 clauses. We replaced datatype properties by object properties and nominals by common concepts because the current version of our system does not support them. The expressivity of the obtained test ontology network is $\mathcal{SHIN}$.

Results For comparing the runtimes in the centralized and distributed setting, we ran a satisfiability check and tested all 456 positive subsumption queries (i.e. axioms $A_1 \sqsubseteq A_2$ derivable from the ontology network). The runtime of negative queries is in general similar to the runtime of a satisfiability check [15]. We used standard 1.6GHz desktop machines with 3GB RAM, the denoted runtimes do not include the time needed for establishing the TCP connections. For simplicity, we connect all reasoners prior to the distributed reasoning process, it would be much more efficient to connect the peers only on demand. The connection time is not relevant for our investigations because it can be

[5] http://sweet.jpl.nasa.gov/2.0/chem.owl

expected to increase only linear with the number of ontologies if the network is sparsely connected. The source code and original and preprocessed dataset is available online[6].
The most important result is that distributed resolution is considerably faster than conventional resolution. The runtime for checking satisfiability of the knowledge base is decreased by one third. Answering a positive subsumption query took only about a quarter of the runtime when computed in the distributed setting.
The number of derived clauses shows the effect of distribution on application of reduction rules. For the satisfiability query, almost all redundant clauses are detected and deleted also in the distributed setting. However, positive queries cause much more derivations than necessary. Runtime is not affected by the redundant derivations because they are performed by reasoners that would have been idle otherwise. The number of propagation is important when the network connection is slow e.g. due to physical distance between the reasoning peers.
The most important factor for scalability of our approach is the amount of computation that is actually performed in parallel. In the worst case, only one reasoning peer is active at the same time while the others are idle and waiting for new input. For technical reasons, the timer for computing the busy factor starts when all reasoners are connected. The busy factor depicted in table 2 is the average (weighted by runtime) percentage of the runtime each reasoner is active. The first couple of milliseconds all reasoners are busy, but after local saturation only those that received new clauses continue reasoning. For some positive queries the busy factor reached 100% because the empty clause is derived in one reasoner before one of the others is locally saturated.

5 Conclusions

In this paper, we have shown that the principle of distributed resolution as a basis for reasoning about interlinked ontologies that has been proposed in previous work [15] can be extended to expressive ontology languages. This result is non-trivial, because ordered resolution, that has been used as a basis for previous work cannot be applied to expressive ontology languages due to the existence of equality induced by number restrictions. Our work extends previous results both on a theoretical and a practical level. On the theoretical level, we have developed a distributed version of the basic superposition calculus presented by [9] and have shown that the distributed version of the calculus decides satisfiability for $\mathcal{ALCHIQ}$. On the practical level, we have extended the implementation of our distributed reasoning engine with this new calculus and have tested it on a set of expressive real world ontologies. By conducting experiments, we have shown that the distributed version of the algorithm significantly outperforms centralized reasoning. Further, we have investigated the ability of the method to support parallelization with promising results. In summary, we have shown that the principle of distributed resolution can be applied to expressive ontologies and that distributed resolution is a real alternative to tableaux-based methods when it comes to distributed reasoning in the presence of OWL ontologies. In the future, we will investigate optimizations of the methods, primarily in terms of advanced redundancy checking. Further,

[6] http://ki.informatik.uni-mannheim.de/dire.html

we plan to exploit the advantages of resolution being a bottom-up reasoning method by investigating the use of our method for supporting incremental reasoning.

References

1. Leo Bachmair, Harald Ganzinger, Christopher Lynch, and Wayne Snyder. Basic paramodulation. *Inf. Comput.*, 121(2):172–192, 1995.
2. Maria Paola Bonacina. The clause-diffusion theorem prover peers-mcd (system description). In *Proceedings of the 14th International Conference on Automated Deduction*, volume 1249 of *Lecture Notes In Computer Science*, pages 53 – 56, London, UK, 1997. Springer Verlag.
3. Maria Paola Bonacina. A taxonomy of parallel strategies for deduction. *Annals of Mathematics and Artificial Intelligence*, 29(1–4):223–257, 2000. Published in February 2001.
4. Maria Paola Bonacina and Jieh Hsiang. Parallelization of deduction strategies: An analytical study. *J. Autom. Reasoning*, 13(1):1–33, 1994.
5. Alex Borgida and Luciano Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
6. Susan E. Conry, Douglas J. MacIntosh, and Robert A. Meyer. Dares: A distributed automated reasoning system. In *Proc. AAAI-90*, pages 78–85, 1990.
7. Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using e-connections. *Journal Of Web Semantics*, 4(1), 2005.
8. F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CLSI Publications, 1996.
9. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Reasoning in Description Logics by a Reduction to Disjunctive Datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.
10. C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*, 2007.
11. Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
12. Robert Nieuwenhuis and Albert Rubio. Theorem proving with ordering and equality constrained clauses. *Journal of Symbolic Computation*, 19:321–351, 1995.
13. Rob Raskin. Knowledge representation in the semantic web for earth and environmental terminology (SWEET), 2005.
14. Anne Schlicht and Heiner Stuckenschmidt. A flexible partitioning tool for large ontologies. In *International Conference on Web Intelligence and Intelligent Agent Technology (WI/IAT)*, 2008.
15. Anne Schlicht and Heiner Stuckenschmidt. Peer-to-peer reasoning for interlinked ontologies. *International Journal of Semantic Computing, Special Issue on Web Scale Reasoning*, 2010 (to be published).
16. Tanel Tammet. *Resolution methods for Decision Problems and Finite Model Building*. PhD thesis, Chalmers University of Technology and University of Göteborg, 1992.
17. D. Tsarkov, A. Riazanov, S. Bechhofer, and I Horrocks. Using vampire to reason with owl. In *International Semantic Web Conference (ISWC)*, 2004.
18. Christoph Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27. Elsevier, 2001.
19. Christoph Weidenbach, Uwe Brahm, Thomas Hillenbrand, Enno Keen, Christian Theobalt, and Dalibor Topić. SPASS version 2.0. In Andrei Voronkov, editor, *Automated deduction - 18th International Conference on Automated Deduction*. Springer, 2002.