Distributed Resolution for $\mathcal{ALC}$ - First Results

Anne Schlicht, Heiner Stuckenschmidt

University of Mannheim, Germany
{anne,heiner}@informatik.uni-mannheim.de

Abstract. The use of description logic as the basis for semantic web languages has led to new requirements with respect to scalable and non-standard reasoning. Description logic is a decidable fragment of FOL but still, the standard reasoning tasks are of exponential complexity, satisfiability and subsumption tests are often intractable on large ontologies. Existing large ontologies have a modular structure like networks of linked ontologies, caused by the development process. However, current reasoning approaches do scarcely take advantage of this structure. The available reasoners do not exploit parallel computation and scalability improvements enabled by distributed reasoning.
In this paper, we lay the foundation for developing distributed reasoning methods by showing that the description logic fragment $\mathcal{ALC}$ can be distributed. We propose a distributed, complete and terminating algorithm that decides satisfiability of terminologies in $\mathcal{ALC}$. The algorithm is based on recent results on applying resolution to description logics. We show that the resolution procedure proposed by Tammet can be distributed amongst multiple resolution solvers by assigning unique sets of literals to individual solvers. This provides the basis for a highly scalable reasoning infrastructure for description logics.

1 Introduction

The use of description logics as one of the primary logical languages for knowledge representation on the Web has created new challenges with respect to reasoning in these logics. In order to be able to support the vision of a semantic web of interrelated ontologies, reasoning procedures have to be highly scalable and able to deal with physically distributed knowledge models. A natural way of addressing these problems is to rely on distributed inference procedures that can distribute the load between different solvers thus reducing potential bottlenecks both in terms of memory and computation time. Currently, almost all the work on description logic reasoning still assumes a centralized approach where the complete terminology has to be present on a single system and all inference steps are are carried out on this system. Exceptions to this rule are approaches like Distributed Description Logics (DDL) [2] that support local reasoning at the price of sacrificing expressiveness in the links between local models and by dropping some formal properties on the level of the overall model. In DDL for example, certain types of inconsistencies are not propagated on the global level. The goal of our work is to support local reasoning in description logics without the need to reduce the expressiveness of links between local models. Based on results in resolution reasoning

$Pair \sqsubseteq Set$	$\{\neg Pair(x_1), Set(x_1)\}$
$Pair \sqsubseteq \forall part.\neg Set$	$\{\neg Pair(x_2), \neg part(x_2, x_3), \neg Set(x_3)\}$
$Set \sqsubseteq \exists part.Set$	$\{\neg Set(x_4), part(x_4, f(x_4))\}, \{\neg Set(x_4), Set(f(x_4))\}$
$Pair(a)$	$\{Pair(a)\}$

Fig. 1. $\mathcal{ALC}$ example and equisatisfiable clauses. This example states that every pair is a set, every part of a pair is not a set and every set has a part that is a set. For testing satisfiability of the concept $Pair$ we additionally state that it has an instance a. The translation to FOL clauses requires replacing the existencial quantified variable y with a skolem function.

for description logic [8], we present an algorithm that decides satisfiability of a set of $\mathcal{ALC}$ ontologies. This algorithm allows us to provide distributed reasoning support on sets of terminologies that share non-logical symbols without merging the modules. A possible application is the provision of distributed reasoning support for ($\mathcal{ALC}$ equivalent parts of) OWL ontologies linked by import statements. Our method guarantees that the global semantics is preserved.

The contribution of this paper is threefold: (1) We identify general requirements for a resolution procedure needed to guaranteeing soundness and completeness. (2) we show that the resolution method proposed in [8] satisfies these requirements. (3) we present first results on the practical applicability of the algorithm. The paper is organized as follows: after a brief discussion of related work in the remainder of this section, we first explain the general idea of distributing standard resolution and discuss the problems that have to be solved for guaranteeing completeness. We then discuss ordered resolution as a adequate basis for distribution and its adaptation to $\mathcal{ALC}$ knowledge bases before turning to the distributed ordered resolution algorithm. Finally, we present the first results of this method and discuss possible optimizations.

1.1 Related Work

Modular DL Reasoning Current approaches to distributed reasoning on description logic mostly rely on tableaux methods. Distribution by solving the two choices of non-determinitic tableaux rules in parallel is difficult as it hampers the application of optimization and blocking strategies. Instead most distributed tableaux approaches try to identify all possible conflicts, i.e. all axioms that might follow from another module and would cause a contradiction and send these as queries to the other modules. So far, this is only done for links with rather restricted expressiveness between the modules.
The most prominent actually distributed T-Box reasoning implementation for ontologies *Distributed Description Logic* (DDL) [2] supports only a special type of links (called bridge rules) between ontologies. The local domains have to be disjoint, i.e. there is no real subsumption between elements of different modules. Like DDL, *E-Connections* [3] treat local domains as disjoint and do not support subsumption relations between modules.
Currently, a modular ontology framework based on *Conservative Extensions* [5] is in development. This framework, however, imposes severe requirements to the self-

containement of the modules. Reasoning with these modules is relatively easy but the computational difficulties are transfered to checking and maintaining the properties of the modules. We think that this approach overshoots the mark and poses too many restrictions on the way axioms might be distributed across different sites.

Distributed Resolution Methods Resolution is used by all successful FOL provers. Approaches to distributed first order reasoning are motivated by efficiency considerations, performance is improved by using multiple processors in parallel. *Roo*[4], for example, is a parallelization of the widely (re)used first order reasoner Otter.
Parallelization differs from the distribution setting, while the former usually uses a shared memory, the latter constitutes a physical separation of modules and resulting higher communication costs. *Partition-Based Reasoning* [1] is a distributed resolution strategy that requires local reasoning to be complete for consequence finding. The requirement inevitably causes derivation of much more clauses than necessary for refutation.

Resolution for Description Logics The main problem with applying FOL approaches for DL is that termination is not guaranteed. Although DL is a decidable fragment of FOL, causing a FOL reasoner to terminate on DL input requires considerable adaption of the algorithm. This paper is based on resolution methods introduced to description logics by [6] that decide satisfiability. The algorithm we present can be seen as a distributed version of the algorithm for deciding $\mathcal{ALC}$ satisfiability proposed there.

2 Distributed Standard Resolution

Analyzing the applicability of the Partition-Based Reasoning approach of [1] to description logic revealed that the only way to avoid redundant derivations is to perform the same inferences as a common resolution reasoner instead of connecting a set of black-box reasoners. Therefore, the basic idea proposed in this paper is to distribute common standard resolution by allocating each derivation step to a unique site in the distributed system.
We first define distributed resolution before detailing this essential property of a distributed resolution calculus.

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

$$\textit{Resolution}\ \frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where σ is the most general unifier of A and B.

A distributed calculus can be created from a given resolution calculus, based on an allocation relation that determines the modules.

Definition 2 (Allocation). *An* allocation function for $\mathcal{C}$ *is a total function* $a : \mathcal{C} \to M$ *that maps clauses to module identifiers.*

The allocation could also be defined by an allocation relation, thus allowing for duplication of clauses to multiple modules. However, we will show that for $\mathcal{ALC}$ duplication is not necessary and we will define an appropriate allocation function. Before we turn to distributed resolution for $\mathcal{ALC}$ we give a general notation of distributed resolution.

Definition 3 (Distributed Resolution). *A* distributed resolution method *for a set of clauses* $\mathcal{C}$ *is a tuple* (R, a) *of a resolution calculus and an allocation function such that for every inference with premises* $P_i \in \mathcal{C}, i \in I$ *the* allocation restriction $\exists m \in M, \forall i \in I : a(P_i) = m$ *is satisfied.*

For rules with only one premise this restriction is trivially true. A trivial allocation is obtained by allocating all clauses to the same module ($a(c) = m_1 \; \forall c \in \mathcal{C}$). Non-trivial allocations constitute a seperation into modules.
The allocation restriction enables implementation of distributed resolution by connecting a set of resolution reasoners, each with a local set of clauses:

- Every module is saturated seperately.
- Newly derived clauses are propagated according to the allocation relation.

Considering one of these reasoners, the difference to performing centralized resolution is, that occasionally locally derived clauses are send to another reasoner and in return, clauses are received from other reasoners and have to be integrated in the local clause list. In addition, a global initialization and termination routine has to be added, that starts the separate solvers and stops if one of them finds a proof or all are saturated.

Definition 4 (Local Resolution). *The local calculus for module* $m \in M$ *in a distributed resolution method* (R, a) *is* R *with additional restriction* $a(P_i) = m$ *for every premise* P_i *to every rule* $r \in R$*.*

In contrast to the centralized case, a module that is saturated locally may have to continue reasoning once a new clause is propagated from another module.
The allocation a was defined to be functional for guarateeing that inferences are never duplicated. Most relevant are inferences with multiple premises because of the expensive search for corresponding sets of premises. The allocation restrictions ensure that every inference of the original calculus occurs in the distributed setting.

Corollary 1. *If a calculus* R *guarantees completeness (termination respective) for input* $\subseteq \mathcal{C}$ *than every distributed reasoning method* (R, a) *for* $\mathcal{C}$ *with a finite set* M *of modules is complete (terminates).*

Considering a derivation of the empty clause in centralized resolution that is not derived in the corresponding distributed resolution method leads to a contradiction: Assume the empty clause is not derived from an unsatisfiable KB by distributed ordered resolution and consider the first inference in the corresponding centralized reasoner that does not occur in the distributed version. By the allocation restriction, all premises are allocated to some module m the inference is carried out by local resolution in that module.
Termination is obviously preserved, no inference is added by distribution.

2.1 Problems Distributing Resolution

In general, clauses might have to be allocated to more than one module to make sure that the premises of every derivation step are allocated to the same module. In order to achieve an efficient method, we have to avoid duplication and allocate every clause to as few sites as possible. Even if we assume the case of only two modules where each may use terms defined by the other, distributing standard resolution might cause most of the axioms in inferences be copied to the other module. To make sure that every pair of clauses that can be resolved with each other meets in a module, we need to send all clauses containing a foreign term (i.e. a term defined in the other module) to the other module. Unfortunately the resolvent of clauses from different modules is very likely to contain terms from both modules that were only used locally before, thus it has to be copied to both modules. If we extend the approach to multiple modules we can expect the number of duplicates to further increase.

The partition based approach [1] adresses this problem by limiting propagation to clauses that contain only shared symbols. It defines the shared language between modules as the clauses consisting of symbols shared by a pair of modules. The system can be depicted as a graph, with modules as nodes and edges for every shared language, labelled with the shared symbols. The graph is modified to form a tree by deleting edges and adding the corresponding symbols to other edge labels. Modules that share a symbol "s" remain connected, there is still some path in the tree connecting the modules with "s" contained in the label of every edge on the path. While the edges in the graph are undirected, edges in the resulting tree point towards the root.
Whenever a local reasoner derives an axiom that is contained in a shared language (i.e. it contains only symbols from an outgoing edge), the axiom is propagated to the corresponding neighbor, towards the root. In difference to our approach clauses are not allocated to a specific module but may be passed on from one to the other until they reach the root. This approach faces some efficiency problems: First, as mentioned before, the system infers much more clauses than necessary for refutation, because the local reasoning has to be complete for consequence finding to guarantee completeness. If the modules are small (i.e. the knowledge base is distributed to many modules) this effect is negligible, but then the communication cost are very high. Second, the computation and communication is not distributed equably among the modules. The root is a distinguished module, communication load boosts towards it. Not only are the edges directed towards the root, also the number of shared language symbols is larger on edges close to the root. (Recall that all modules that share some symbol remain connected by a path labelled with this symbol.) Third, the partitioning depends on the query. If the query clauses are not contained in some local language, new links have to be introduced and repartitioning might be necessary.

Nevertheless, we will show that with a different approach it is possible to define a distributed resolution method for $\mathcal{ALC}$ that is sound and complete, and terminates. It does not require duplication of axioms, every clause is allocated to exactly one module. New axioms (e.g. queries) can be easily added to the system. The approach is fairly distributed, no module stands out from the others.

3 Ordered Resolution

The problem with applying first order methods for DL is loosing decidability. An DL-ontology input to a complete first order reasoner will not terminate in general. [8] showed that using ordered resolution, decidability of some description logics can be preserved by transforming the ontology into a special normal form. Before we give the definition of the method for $\mathcal{ALC}$ from [6] we define the resolution rules it is based on. Ordered resolution depends on two parameters that can be modified to improve performance e.g. for restricted subsets of first order logic like description logic. First parameter is the *order* of literals (for defining the maximal literals) that can be defined on top of an order of predicate symbols, function symbols and constants. The second parameter is the *selection function* that maps every clause C to a subset $S(C)$ of it's negative literals.

Definition 5 (Ordered Resolution).

$$\textit{Ordered resolution} \quad \frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma}$$

where

1. *σ is the most general unifier of A and B*
2. *either B is selected in $D \vee \neg B$ or else nothing is selected in $D \vee \neg B$ and $B\sigma$ is maximal w.r.t. $D\sigma$*
3. *$A\sigma$ is strictly maximal with respect to $C\sigma$*
4. *nothing is selected in $C\sigma \vee A\sigma$*

Definition 6 (Positive Factoring).

$$\textit{Positive Factoring} \quad \frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

where

1. *σ is the most general unifier of A and B*
2. *$A\sigma$ is maximal with respect to $C\sigma \vee B\sigma$*
3. *nothing is selected in $C\sigma \vee A\sigma \vee B\sigma$*

Compared to unrestricted resolution, many derivations are skipped in ordered resolution. A literal A that is not selected in a clause C can only be resolved upon if nothing else is selected in the clause and $A\sigma$ is maximal. Intuitively, the effect of the selection function is a change in the order of literals in a clause, the selected literals are moved to the front. In addition, the combination of multiple inferences for skipping redundant intermediate results (i.e. hyperresolution) is controlled by selection.

3.1 Adaption to $\mathcal{ALC}$

We assume a $\mathcal{ALC}$ ontology transformed into first order clauses (consider Figure 1 for an example). Modules are thus sets of clauses that can be derived from the given ontology. Different ontologies that share terms (e.g. one ontologie uses a concept defined in

another ontology) can be considered as modules, too. The ontologies are translated into clauses seperately resulting in the same structure as an ontology that is first clausified and than partitioned into modules.

For preserving decidability, the ontology has to be transformed into *definitorial form* before clausification. Then, resolving the clauses with ordered resolution derives an empty clause iff the ontology is inconsistent and terminates for any ontology. Intuitively, the definitorial form can be seen as contrary of unfolding, it splits up nested axioms into simple ones by introducing new concepts.

Definition 7 (Definitorial Form). *The definitorial form of a concept C in negation normal form is*

$$\mathit{Def}(C) = \begin{cases} C & \text{if all subterms of } C \text{ are literal concepts} \\ \{\neg Q \sqcap C|_p\} \cup \mathit{Def}(C[Q]_p) & \text{else} \end{cases}$$

where $C|_p$ is a concept that occurs in C (subterm of C), but not a literal concept and all subterms of $C|_p$ are literal concepts. $C[Q]_p$ is the concept definition that results from replacing $C|_p$ with Q.

Definition 8 ($\mathcal{ALC}$ Resolution). $\mathcal{ALC}$ resolution ($R_{\mathcal{ALC}}$) *is the calculus with*

- *rules ordered resolution and factoring,*
- *selection of exactly the negative binary literals, and*
- *literal ordering $\succ$ with $R(x, f(x)) \succ \neg C(x)$ and $D(f(x)) \succ \neg C(x)$, for all function symbols f, and predicates R, C, and D.*

The requirement to the literal ordering is satisfied by every *lexicographic path ordering* based on a total precedence $>$ with $f > P > \neg$ for every function symbol f and predicate symbol P.

Definition 9 (Lexicographic Path Ordering). *A lexicographic path ordering (LPO) is a term ordering induced by a well-founded strict precedence $>$ over function, predicate and logical symbols, defined by:*
$s = f(s_1, \ldots, s_m) \succ g(t_1, \ldots, t_n) = t$ if and only if
at least one of the following holds

(i) $f > g$ *and* $s \succ t_i$ *for all* i *with* $1 \leq i \leq n$
(ii) $f = g$ *and for some* j *we have* $(s_1, \ldots, s_{j-1}) = (t_1, \ldots, t_{j-!})$, $s_j \succ t_j$ *and* $s \succ t_k$ *for all* k *with* $j < k \leq n$
(iii) $s_j \succeq t$ *for some* j *with* $1 \leq j \leq m$

Theorem 1 ($\mathbf{R_{DL}}$ complexity).
For an $\mathcal{ALC}$ knowledge base KB, clausifying its definitorial form and saturating the clauses by $R_{\mathcal{ALC}}$ decides satisfiability of KB and runs in time exponential in KB.

This theorem is the reason for using ordered resolution in our approach, we thereby guarantee termination. Note that ordered resolution is not complete for consequence finding and hence, applying it for the partition based reasoning approach of [1] results in an incomplete procedure.

4 Distributed Ordered Resolution

Inuitively, the idea for distributing resolution is to allocate clauses based on their resolvable literals.

- Every module ”hosts” a subset of the literals, i.e. it is responsible for all inferences resolving upon one of these literals.
- Every (derived or stated) clause is moved to the modules that host a resolvable literal of the clause.

Essential for distributing resolution without duplicating clauses and derivations is to reduce the number of modules that may be responsible for the next derivation to a single module i.e. to define the resolution method in such a way that there is a unique resolvable literal for every clause. Note that literals can be allocated to modules based on their *top symbol* (i.e. the predicate P of the literal $(\neg)P(t)$ where t is a term) and a partitioning of symbols that allocates every symbol to exactly one module.

4.1 Resolvable Literals of a Given Clause

For defining a resolution method that is complete for this communication strategy, we investigate the options for using a given clause in an inference. The first thing we discover from the definition of ordered resolution is that a clause can only be main premise and not side premise if some literal is selected by the selection function. By exploiting the other requirements we obtain the decision tree in Figure 2 (assuming the decision on the resolvable literal is not affected by unification).

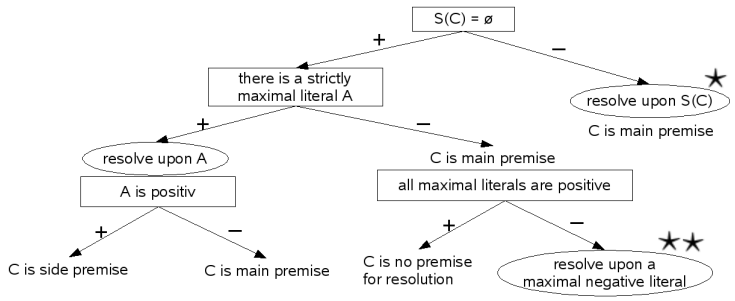

Fig. 2. Clause decision tree depicting the resolution options for a given clause.

If nothing is selected, only a maximal literal can be resolved. If there are multiple maximal literals, then the clause cannot be a side premise. Problematic are multiple negative maximal literals because any of them can be the resolved literal in the next derivation. If all multiple maximal literals are positive the clause cannot be premise for ordered resolution because it can neither be a side premise nor a main premise. A

given clause can never be side premise and main premise at same time (in different inferences). If the selection function is empty, the first branch on the right is pruned. If the order is total on all literals, the branch in the middle is pruned. By exploiting the effects of the selection and ordering on the clause tree, two problematic cases are identified:

- $\star$ Multiple selected literals: If literals from different moduls are selected, we cannot allocate the inference to one module.
- $\star\star$ Multiple negative maximal literals, none selected: We have to send the clause to every module that is responsible for derivations resolving upon one of these literals.

Note that these restrictions apply to the unified clause: The definition of ordered resolution requires each resolved atom $A\sigma$ to be (strictly) maximal with respect to $D\sigma$ ($C\sigma$) but this does in general not imply that A is (strictly) maximal w.r.t. the premise before substitution. However, the ordering defined for $R_{\mathcal{ALC}}$ is *admissible* which implies invariance under substitutions and the selection function is independent of substitutions, too. For guaranteeing termination of $R_{\mathcal{ALC}}$, ordering and selection were defined in a way that restricts the clauses to five different types [6]:

	clause type	type of resolvable literal
1	$R(x, f(x)) \vee \mathbf{P}(x)$	$R(x, f(x))$
2a	$\mathbf{P}(x)$	$(\neg)P(x)$
2b	$\mathbf{P_1}(\mathbf{f}(x)) \vee \mathbf{P_2}(x)$	$(\neg)P(f(x))$
3	$\neg R(x, y) \vee \mathbf{P_1}(x) \vee \mathbf{P_2}(y)$	$\neg R(x, y)$
4	$\mathbf{P}(\mathbf{a})$	$(\neg)P(a)$
5	$(\neg)R(a, b)$	$(\neg)R(a, b)$

Table 1. ALC clauses. For a term t, $\mathbf{P}(t)$ denotes a possibly empty disjunction of the form $(\neg)P_1(t) \ldots (\neg)P_n(t)$. $\mathbf{P}(\mathbf{f}(x))$ denotes a possibly empty disjunction of the form $\mathbf{P_1}(f_1(x)) \ldots \mathbf{P_m}(f_m(x))$. Note that this definition allows each $\mathbf{P_i}(f_i(x))$ to contain positive and negative literals. We choose a slightly different notation of the $\mathcal{ALC}$ clauses with the type 2 clauses subdivided because the resolvable literal is of type $P(f(x))$ if one exists and otherwise $P(x)$.

Obviously, $\mathcal{ALC}$ clauses never contain more than one selected literal, thus, the first problem does not occur in $R_{\mathcal{ALC}}$. The second problem can be avoided, too, by either applying a total order on literals or selecting one of the multiple maximal literals. The effect is similar because considering the possible inferences, a selected literal is treated like a strictly maximal literal. We choose the former and extend the ordering $\succ$ defined for $R_{\mathcal{ALC}}$ as follows.

Definition 10 (Ordering for Distributed Resolution).
For two literals of $\mathcal{ALC}$ clauses A and B, $A \succ_D B$ iff $A \succ B$ or A and B are incomparable wrt. $\succ$ and i) A contains a function symbol but not B or ii) A and B

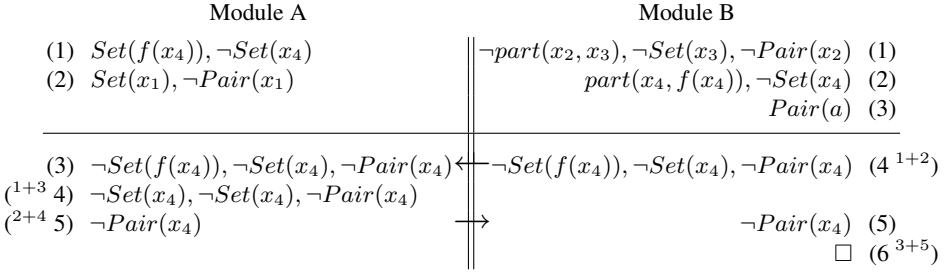

Fig. 3. Example of Distributed Ordered Resolution. The example from Figure 1 is distributed to two modules. Module A hosts literals with top predicate Set, B hosts literals with top predicate $part$ or $Pair$. Stated clauses are noted above the horizontal line, below (4 $^{1+3}$) means clause (4) was derived from clauses (1) and (2). The first literal is the resolvable literal of a clause (let $Set > Pair$). Arrows depict propagation of clauses, we ommit braces for purpose of readability. Note that in this example most clauses contain symbols hosted by the other module. In realistic examples this overlap is very small and we can expect many of the clauses to be kept in the module where they were infered.

contain the same number of function symbols and the top predicate of A precedes the top predicate of B.
The calculus $R_{\mathcal{ALC}}$ with extended ordering $\succ_D$ is denoted by $R^D_{\mathcal{ALC}}$.

The literals that remain incomparable wrt. $\succ_D$ have the same top predicate and are therefore hosted by the same module.

Corollary 2 (Distributed Ordered Resolution).
Distributed Ordered Resolution, i.e. the method $(R^D_{\mathcal{ALC}}, a)$ with allocation a defined by $a(c) = part(topSymbol(resolvableLiteral(c)))$ based on a partitioning of predicates part *that maps every predicate to a module identifier is a distributed resolution method.*

Since the resolvable literal is unique, the allocation is a function. The allocation restrictions are satisfied because for any pair of clauses that are premises of an inference the resolvable literals have the same top symbol and are thus allocated to the same module. Hence, if $R^D_{\mathcal{ALC}}$ is complete and terminates, by Corollary 1 the same holds for $(R^D_{\mathcal{ALC}}, a)$.

Theorem 2 (Completeness and Termination of Distributed Ordered Resolution).
Distributed Ordered Resolution is a complete and terminating method for deciding $\mathcal{ALC}$ satisfiability.

Proof. The theorem follows from completeness and termination of $R_{\mathcal{ALC}}$ (Theorem 1). We have to show that $R^D_{\mathcal{ALC}}$ is complete and terminates. The definition of ordered resolution allows for an arbitrary ordering of literals, hence $R^D_{\mathcal{ALC}}$ is complete. We only have to make sure that Theorem 1 is not affected by this modification and hence termination is preserved. In fact, the requirements $R(x, f(x)) \succ_D \neg C(x)$ and $D(f(x)) \succ_D \neg C(x)$ still hold for all function symbols f, and predicates R, C, and D.

Consider Figure 2 for an example refutation.

5 Experiments

The algorithm presented above is not yet implemented in an actually distributed reasoner, but first experiments were conducted simulating the distributed setting. For investigating distributed ordered resolution we implemented the definitorial form normalization and modified the resolution reasoner SPASS to perform the $R^D_{\mathcal{ALC}}$ resolution. The crucial point for determining performance of distributed ordered resolution is the number of propagated clauses. If the modules happen to be completely independent, performance would be improved by a factor propotional to the number of modules. But, in realistic settings the modules will allways share some symbols and clauses will be exchanged between them. Propagation of a derived clause to another module is considerably more expensive then just adding it to the local clause list. Apart from the distance that has to be bridged over, a received clause requires modification of the local index structures that are used to manage the clause lists. Hence, the ratio of derived clauses that have to be propagated to another module is the dicisive criterion.

5.1 Setting

For our experiments we used the FOODSWAP[1] ontology. First we transformed the ontology into its definitorial form. Then we used our partitioning tool PATO for assigning each term (i.e. concept or property name) a module number. PATO first builds a dependency graph from the ontology that is afterwards partitioned into parts that are stronger internally connected than extenally connected[2]. For minimizing the communication we configured Pato to create a link in the dependency graph for every pair of terms appearing in the same axiom. Finally we started the SPASS reasoner with appropriate configuration to check consistency of the ontology. For every ordered resolution derivation we recorded the module that performed the derivation (i.e. the module number of the resolved literal) and the module to which the derived clause is send (i.e. the module number of the resolvable literal of the derived clause).

5.2 Results

Analysing the list of source and destination of every derived clause, we found that out of the 229 derived clauses 110 were kept in the same module, the other 119 clauses were propagated to another module. The complete picture of the communication is depicted in Figure 4. The communication load of 52% of the derived clauses may already be acceptable in many applications, but still further reduction can be achieved by merging modules as depicted in the subsequent graphs. In the three module setting only 28% of the clauses are propagated and 26% for the two modules while the sizes remain balanced.

[1] http://www.mindswap.org/dav/ontologies/commonsense/food/foodswap.owl

[2] A detailed description of PATO can be found in [7]

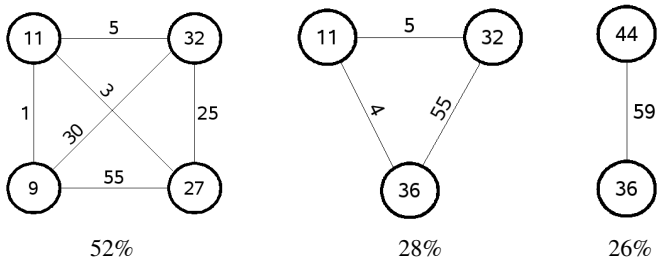

Fig. 4. Communication between modules for the satisfiability test. Modules are denoted by circles with the size (i.e. number of terms) of the module denoted inside. Edges are labelled with the number of clauses propagated between the corresponding modules, propagation direction is not distinguished for purpose of readability. The numbers below the graphs give the percentage of derived clauses that have to be propagated.

Positive tests like this consistency check require complete saturation whereas negative test are often answered faster by resolution reasoners. For our small test case the time required for reasoning is dominated by the time for loading the input, but the number of derived clauses is smaller for positive subsumption checks (i.e. derivation of the empty clause). E.g. testing the subsumption $(meat \sqsubseteq food)$ by adding its converse $(meat \wedge \neg food)$ to the clauses required only 45 derivations (with 47% propagation in the four module setting). In contrary, the negative subsumption test $(eggs \sqsubseteq dairy)$ is similar to the positive consistency check (226 derivations, 115 propagations).

6 Optimizations

The first results of distributed ordered resolution are very encouraging. However, additional tests with different larger ontologies are required and further reduction of the propagation ratio is desirable.

6.1 Partitioning

One of the next steps for improving performance will be to optimize the partitioning wrt. communication. In our experiment we started with a partitioning generated by PATO and than reduced communication by merging modules. In the same way we could start with all modules containing only a single term and find the optimal partitioning wrt. a given reasoning task. Finding the optimal partitioning is an apparently hard problem, but approximations will probably meet the needs.

6.2 Selection Function

Another topic of investigation is the selection function. Up to now, it selects at most one literal. Thus, a derived clause is very likely to be moved to another module. It would

be more efficient to select local literals first. Moreover, if we select multiple literals, we can combine multiple inferences to a hyperresolution inference and skip redundant intermediate results. By investigating the requirements necessary to make sure the set of $\mathcal{ALC}$ clauses is closed under $R_{\mathcal{ALC}}$, we can improve the selection function:

Definition 11 (Adapted Distributed Selection).

1. *Select nothing from clauses of type 1 and 5.*
2. *From type 3 clauses select the negative binary literal.*
3. *From clauses of type 2b select the maximal number of negative literals of type $P(f(x))$ hosted by a single module.*
4. *From clauses of type 2a and 4 select the maximal number of negative literals hosted by a single module.*

We may not select all local literals but selecting all literals hosted by one module has a similar effect. First the clause is moved to that module, but then all local literals are solved in the next inference.

6.3 Reduction

For efficient resolution reasoning it is not sufficient to avoid unneccessary derivations. Additionally, reduction techniques have to be applied that delete clauses if they are tautologies or subsumed by other clauses. Deleting newly derived clauses that are subsumed is forward subsumption, backward subsumption refers to the deletion of clauses that are subsumed by a newly derived clause. The naïve approach to distributed subsumption would be to propagate new clauses to all modules that host one of its literals for forward subsumption checks and send it to modules that are responsible for literals equal or smaller than the resolvable literal of the new clause for backward subsumtion. Fortunately we can again take advantage of the clause typology. Of all different clause types only type 2a can be subsumed by a clause of different type, every other clause can only be subsumed by a clause of the same type. Thus, by indexing the clauses with their types we can skip a large part of the subsumption tests.

7 Summary

We addressed the problem of improving the scalability of description logic reasoning by proposing a distributed resolution method for $\mathcal{ALC}$ terminologies. The algorithm extends ordered resolution with a method for assigning clauses to a unique location at which all possible resolution steps are executed by a local solver. We analyzed the properties of such a distributed method and identified necessary conditions for guaranteeing completeness and termination of the algorithm. Further, we have shown that the resolution method for $\mathcal{ALC}$ described by Motik satisfies these conditions and can therefore be distributed across different distributed reasoners. Our investigations lay the foundation for the implementation of a large scale reasoning infrastructure for $\mathcal{ALC}$ terminologies

and can be seen as a first step towards supporting the vision of the semantic web as a distributed system of interlinked ontologies that can be reasoned upon. First experiments on the performance of the proposed algorithm are promising but more detailed investigation is necessary. In particular, we have to find a way to control the cost of communication between local reasoners. The major task in this context is the definition of a suitable distribution strategy based on the nature of the terminologies involved. Further, we have to investigate possible optimizations of the reasoning method some of which are already mentioned in Section 6. Another major issue is the development of distributed reasoning methods for more expressive languages – the final goal is to support reasoning in OWL which is known to be equivalent to $\mathcal{SHOIN}(\mathcal{D})$. A direct extension of the approach proposed here is not possible because dealing with number restrictions requires a different resolution approach, i.e. the use of paramodulation. In this context an assignment of clauses to a certain reasoner is not feasible, duplication not be completely avoided. An adaption of our approach with limited duplication to a calculus that supports equality is the focus of future work.

Acknowledgement

This work was partially supported by the German Science Foundation in the Emmy-Noether Program under contract Stu 266/3-1.

References

1. E. Amir and S. McIlraith. Partition-based logical reasoning for first-order and propositional theories. *Artificial Intelligence*, 162(1-2):49–88, 2005.
2. A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
3. Bernardo Cuenca Grau, Bijan Parsia, and Evren Sirin. Combining owl ontologies using e-connections. *Journal Of Web Semantics*, 4(1), 2005.
4. Ewing L. Lusk, William W. McCune, and John Slaney. Roo: a parallel theorem prover. In *In Proceedings of the 11th CADE*, volume 607 of LNAI, pages 731–734. Springer Verlag, 1992.
5. C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence IJCAI-07*. AAAI Press, 2007.
6. Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
7. Heiner Stuckenschmidt and Michel Klein. Structure-based partitioning of large concept hierarchies. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proceedings of the Third International Semantic Web Conference (ISWC 2004)*, pages 289–303, Hiroshima, Japan, nov 2004.
8. Tanel Tammet. *Resolution methods for Decision Problems and Finite Model Building*. PhD thesis, Chalmers University of Technology and University of Göteborg, 1992.