Towards Distributed Ontology Reasoning for the Web

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 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 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. In this paper, we propose a distributed resolution approach that solves the problem by local resolution and propagation of derived axioms between different reasoners. The method is complete for first order logic, terminates for $\mathcal{ALC}$ ontologies and avoids duplication of axioms and inferences. The work can be seen as a building block for a large scale distributed reasoning infrastructure for the semantic web as envisioned in recent activities such as the Large Knowledge Collider (LarKC) project.

Keywords: *Semantic Web, Web Reasoning/Inference Engines*

1 Introduction

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 carried out on this system. This centralization of reasoning is in conflict with the idea of the semantic web that ontological knowledge is created and maintained in a distributed fashion. OWL - the de facto standard for representing ontological knowledge on the web explicitly allows the unrestricted use of terms and - via the `owl:import` statements - their definitions from remote ontologies. Researchers in Knowledge Representation and reasoning have addressed this problem by proposing logics for representing import relations and providing distributed reasoning methods that perform reasoning mostly local and communicate when necessary to ensure completeness. Examples are DDL [1], ϵ-connections [2], P-DL [3] and Semantic Import [5]. All of these approaches rely on the fact, that knowledge is already distributed in a specific way[1]. Our goal is to exploit the benefits of distributed reasoning – independently of whether a specific distribution of knowledge is given – by distributing necessary inference steps over multiple available reasoners.

2 Distributing Standard Resolution

Before describing our distributed resolution method for ontologies, we first briefly review standard resolution reasoning and present the basic idea for distributing resolution. Resolution is used by most successful first order logic (FOL) provers. As Description logics are a strict subset of first order logic, resolution can be applied to ontologies as well by transforming the DL ontology into a set of first order clauses. For example, the input clauses in Figure 1 are obtained from the DL axioms $Pair \sqsubseteq Set$, $Pair \sqsubseteq \forall part.\neg Set$, $Set \sqsubseteq \exists part.Set$, $Pair(a)$. The function f is a skolem function that substitutes the existencial quantified variable y. Translation to FOL can be done on a per axiom basis independently of other parts of the model. An ontology is unsatisfiable if and only if the set of clauses is satisfiable which can be decided by an exhaustive application of the standard resolution rule.

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

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

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

[1] In particular, the expressiveness of links between distributed parts of the model is often restricted

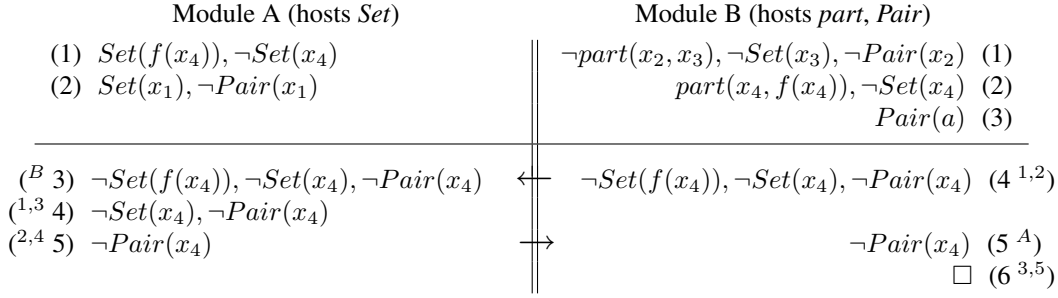


Figure 1. Example of Distributed Ordered Resolution.

Algorithm 1 Resolution Prover

ISSATISFIABLE(KB)

```
1:  Wo ← ∅
2:  Us ← KB
3:  while Us ≠ ∅ and □ ∉ Us do
4:     Given ← CHOOSE(Us)
5:     Us ← Us \ {Given}
6:     Wo ← Wo ∪ {Given}
7:     New ← RESOLVE(Given,Wo)
8:     (Given, Us, New) ← REDUCE(Given, Us, New)
9:     Us ← Us ∪ New
10: end while
11: if □ ∈ Us then return false
12: else return true
13: end if
```

Algorithm 1 (adapted from [8]) controls the application of the resolution rule on a set of input clauses. It iteratively picks a clause, searches for clauses that can be resolved with this given clause, applies resolution to derive new clauses and adds the newly derived clauses to its clause set. Before continuing with a new given clause, reduction rules are applied to delete redundant clauses and avoid redundant inferences. For recording which clauses have already been resolved with each other, the clause set is split into two lists, the *usable (Us)* list of clauses that have to be resolved and the *worked off (Wo)* list. Clauses in the *Wo* list are saturated, i.e. every clause that could be derived from *Wo* is either already contained in one of the lists or redundant.

This basic algorithm can bedistributed 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.

The propagation of clauses is added into line 9 of Algorithm 1: instead of directly adding the new clauses to the local *Us* list, some of the new clauses may be added to the *Us* lists of other reasoners and new clauses may be received from other reasoners. The system stops if the empty clause is derived in one of the reasoners or all are saturated. 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.

Obviously, an arbitrary distribution of resolution can lead to inefficiency. For guaranteeing the completeness of the resolution algorithm in a distributed setting, we have to ensure a given clause is resolved with every matching clause in any of the *Wo* lists. Hence a given clause C would have to be propagated to any reasoner whose *Wo* list contains a clause with a literal that matches (i.e. is unifiable and of opposite polarity) any of the literals in C leading to a substantial communication overhead and potentially redundant inference steps. To avoid redundancy we define distributed resolution based on an allocation function, that allocates every clause to only a single reasoner.

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

The allocation function defines the propagation necessary for distributing Algorithm 1. If a newly derived clause is allocated to the reasoner where it was derived, no propagation is necessary. Otherwise the derived clause is sent to the reasoner specified by the allocation and the local copy is deleted[2]. Based on the notion of an allocation function, we can now define a distributed resolution method as a combination of a resolution calculus and an allocation function that satisfies a certain condition.

Definition 3 (Distributed Resolution) *A* distributed resolution method *for a set of clauses* $\mathcal{C}$ *is a tuple* (R, a) *where* R *is a resolution calculus and* a *is an allocation function for* $\mathcal{C}$ *such that for every set of clauses* $CS \subset \mathcal{C}$ *that can be the premises in an inference of* R *the* allocation restriction $\exists m \forall c \in CS \;\; a(c) = m$ *is satisfied.*

The restriction of the allocation function introduced in Definition 3 guarantees that the distributed version of a complete resolution method is also complete.

[2]Strictly speaking, the clause is just marked as deleted to avoid repropagation.

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

Completeness and termination are preserved, because we neither impede nor add any inference by distribution. The same resolution step is never carried out twice, because resolvable clauses are always assigned to the same unique reasoner which takes care of avoiding local redundancy.

3 Distributed Resolution for $\mathcal{ALC}$

Due to limited space, we only sketch our method and demonstrate its application. Refer to [6] for detailed description and formal definition of distributed resolution for $\mathcal{ALC}$. 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 an allocation function that satisfies the allocation restriction. Our method is based on *ordered resolution*, a calculus that terminates on input obtained from $\mathcal{ALC}$ ontologies ($\mathcal{ALC}$*-clauses*) if its parameters *selection function* and *ordering of literals* are set appropriately [7, 4]. We found that – apart from deciding $\mathcal{ALC}$– this parameterization has the effect that in each $\mathcal{ALC}$-clause there is a unique literal that can be resolved upon (i.e. the literals A or $\neg B$ in Definition 1) in an ordered resolution inference. This condition is the bases for defining an allocation function. In particular, for determining where (and if) a derived clauses is sent, we first pick the unique resolvable literal of the clause, then the top predicate of this literal and finally the reasoner this term is allocated to by an arbitrary partitioning of the ontology signature. Figure 1 illustrates application of Distributed Ordered Resolution. Stated clauses are noted above the horizontal line, below ($4^{1,3}$) means clause (4) was derived from clauses (1) and (3). The first literal of a clause is the resolvable literal (let *Set* $>$ *Pair*), arrows depict propagation of clauses. Clause (4) derived in module B is sent to A because the top predicate *Set* of the resolvable literal $\neg Set(f(x_4))$ is allocated to A. From the received clause and clause (1) module A derives a clause (4) that is not propagated because the top predicate of the resolvable literal $\neg Set(x_4)$ is hosted by A. Clause (5) in module A is propagated again because *Pair* is hosted by B.

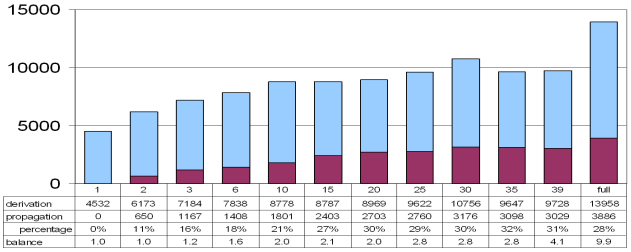
4 Experiments

In this section we now address some practical properties of the complete distributed resolution method that determine the usefulness of the method. In particular, we investigate the overhead of distributed reasoning in terms of the amount of communication between reasoners as well as the impact of only performing reduction locally. For this purpose we implemented the definitorial form normalization and modified the resolution reasoner SPASS to only perform subsumption deletions when subsumer and subsumee are assigned to the same reasoner by the allocation function. Then we started the reasoner with an appropriate configuration that corresponds to the resolution method for $\mathcal{ALC}$ described above to check consistency of the ontology.
For our experiments we used the AMINOACID[3] ontology, a relatively rich $\mathcal{ALCF}$ ontology. Because our method cannot handle functional properties we approximated the ontology by removing the two functional type statements. Subsequently, we transformed the ontology into its definitorial form and then into clauses. The partitioning of ontology terms was obtained by first running a satisfiability test using a complete partitioning that allocates every term to a different module. We encoded the resulting communication between the modules as a graph and partitioned it using METIS[4] From the resulting partitioning we obtained a mappings that assign each concept or property name to a reasoner. These mappings were used to define the modules in the subsequently reported experiments.

4.1 Percentage of Propagated Clauses

The crucial point for determining performance of distributed ordered resolution is the number of propagated clauses as this propagation introduces an additional communication overhead that reduces the usefulness of distribution and can eventually lead to a performance bottleneck if a large fraction of the clauses have to be propagated because 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 decisive criterion.
Our hypothesis is that in realistic settings the number of clauses that have to be propagated is rather small. We tested this hypothesis on the dataset mentioned above. For computing the percentage of propagated clauses, we recorded for every ordered resolution derivation the module that performed the derivation (i.e. the module number of the resolved literal) and the module to which the derived clause has been send (i.e. the module number of the resolvable literal of the derived clause).
The results in Figure 2 show that, as expected, the fraction of clauses that have to be propagated (the lower part of the bars) increases with the number of reasoners used. For all

[3] http://www.co-ode.org/ontologies/amino-acid.owl
[4] http://glaros.dtc.umn.edu/gkhome/views/metis/



	1	2	3	6	10	15	20	25	30	35	39	full
derivation	4532	6173	7184	7838	8778	8787	8969	9622	10756	9647	9728	13958
propagation	0	650	1167	1408	1801	2403	2703	2760	3176	3098	3029	3886
percentage	0%	11%	16%	18%	21%	27%	30%	29%	30%	32%	31%	28%
balance	1.0	1.0	1.2	1.6	2.0	2.1	2.0	2.8	2.8	2.8	4.1	9.9

Figure 2. Number of derived and propagated clauses and size balance for satisfiability test in different partitionings.

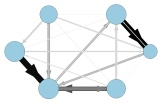


Figure 3. Module graph for the partitioning of six modules.

tests less than a third of the derived clauses are propagated to another module. The distribution of computation load among the modules is measured by the balance value. Balance is defined as the largest number of clauses derived in a single module divided by the average number, e.g. for six modules, the most active module derived 18% more clauses than the average. Figure 3 illustrates the communication between the modules, with thickness of arrows corresponding to the number of propagations and size proportional to the number of clauses derived in each module. For the six module setting we additionally tested all subsumption queries with positive result and a random subset of 30% of the negative subsumption queries. The number of derivations range from 6 to 6208 for positive and 7841 to 8669 for negative queries of which 0% to 35% respective 16% to 18% have to be propagated. For negative queries the number of derivations and propagations needed is higher and almost constant because the whole model has to be saturated.

4.2 Effect of Restricted Reduction

Reduction rules are essential for efficient resolution reasoning. While ordered resolution and factoring are sufficient to guarantee termination for $\mathcal{ALC}$ ontologies in theory, additional deletion of redundant clauses is necessary for practical termination. The most important reduction – subsumption deletion – is restricted by our method. While in common resolution, all clauses that are subsumed by another clause can be deleted, our system only deletes clauses whose subsumer is contained in the same module. For investigating the effect of this restriction we compared the number of derived clauses in unrestricted $\mathcal{ALC}$-resolution versus distributed $\mathcal{ALC}$-resolution. The results depicted in Figure 2 are encouraging, for distribution to three modules, there is an increase of 50% in the number of derived clauses compared to unrestricted resolution in a single module. Increasing the number of modules from 10 to 39 affects the derivations marginally, only the balance of module sizes degrades.
Thus, deleting clauses that are redundant with respect to the reasoner they are processed by seems to be sufficient in practice.

5 Summary

We have addressed the problem of distributing reasoning methods for ontologies across different reasoners on the web. Our focus was on distributing resolution reasoning and we showed that resolution can be distributed without losing completeness if resolvable clauses are always processed by the same reasoner. Based on this observation, we presented a sound and complete distributed resolution method for ontologies that builds upon existing work on resolution reasoning for $\mathcal{ALC}$ models. This work can be seen as a first step towards a highly scalable distributed reasoning architecture for the semantic web and is in line with recent efforts in building such a reasoning infrastructure[5].

References

[1] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.
[2] B. C. Grau, B. Parsia, and E. Sirin. Combining owl ontologies using e-connections. *Journal Of Web Semantics*, 4(1), 2005.
[3] B. J., D. Caragea, and V. Honavar. A tableau-based federated reasoningalgorithm for modular ontologies. In *In Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 404–410, 2006.
[4] B. Motik. *Reasoning in Description Logics using Resolution and Deductive Databases*. PhD thesis, Universität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
[5] J. Z. Pan, L. Serafini, and Y. Zhao. Semantic import: An approach for partial ontology reuse. In *Proc. of the ISWC 2006 Workshop on Modular Ontologies*, 2006.
[6] A. Schlicht and H. Stuckenschmidt. Distributed resolution for alc - first results. In *ESWC Workshop on Advancing Reasoning on the Web*, 2008.
[7] T. Tammet. *Resolution methods for Decision Problems and Finite Model Building*. PhD thesis, Chalmers University of Technology and University of Göteborg, 1992.
[8] C. Weidenbach. *Combining Superposition, Sorts and Splitting*, volume II, chapter 27. Elsevier, 2001.

[5] the Large Knowledge Collider: http://www.larkc.eu/