Towards Parallel Classification of TBoxes

Mina Aslani and Volker Haarslev

Concordia University, Montreal, Canada

Abstract. One of the most frequently used inference services of description logic reasoners is the classification of TBoxes with a subsumption hierarchy of all named concepts as the result. In response to (i) emerging TBoxes from the semantic web community consisting of up to hundreds of thousand of named concepts and (ii) the increasing availability of multi-processor and multi- or many-core computers, we propose a parallel approach for TBox classification. First experiments on parallelizing well-known algorithms for TBox classification were conducted to study the trade-off between incompleteness and speed improvement.

1 Introduction

Due to the recent popularity of OWL ontologies in the semantic web one can observe a trend toward the development of very large or huge OWL-DL ontologies. For instance, well known examples from the bioinformatics or medical community are SNOMED, UMLS, GALEN, or FMA. Some (versions) of these ontologies consist of more than hundreds of thousands of named concepts and have become challenging even for the most advanced and optimized description logic (DL) reasoners. Although specialized DL reasoners for certain sublogics (e.g., CEL [2] for EL++) and OWL-DL reasoners such as FaCT++ [15], Pellet [14], or RacerPro[1] could demonstrate impressive speed enhancement due to newly designed optimization techniques, we expect the need for parallelizing description logic inference services in the near future in order to achieve a better scalability. Our research is also strongly motivated by recent trends in computer hardware where CPUs (will) feature multi-cores (2 to 8 cores) or many-cores (tens or even hundreds of cores), e.g., see [11]. These CPUs promise significant speed-ups for algorithms exploiting so-called thread-level parallelism. This type of parallelism is very promising for DL reasoning algorithms that can be executed in parallel but might share common data structures (e.g., and/or parallelism in proofs, classification of TBoxes, ABox realization or query answering).

First approaches on more scalable ABox reasoning algorithms were investigated with the Racer architecture [10] where novel instance retrieval algorithms were developed and analyzed, which exploit a variety of techniques such as index maintenance, dependency analysis, precompletion generation, etc. Other research focused on scalable ABox reasoning with optimization techniques to partition ABoxes into independent parts and/or creating condensed (summary)

[1] http://racer-systems.com

ABoxes [6, 9, 4]. These approaches rely on the observation that the structure of particular Aboxes is often redundant and these Aboxes contain assertions not needed for ABox consistency checking or query answering.

Parallel algorithms for description logic reasoning were first explored in the FLEX system [3] where various distributed message-passing schemes for rule execution were evaluated. The reported results seemed to be promising but the research suffered from severe limitations due to the hardware available for experiments at that time. The only other approach on parallelizing description logic reasoning [12] reports promising results using multi-core/processor hardware, where the parallel treatment of disjunctions and individual merging (due to number restrictions) is explored. There also exists work on parallel reasoning in first-order theorem proving but due to completely different proof techniques (resolution versus tableaux) and reasoning architectures this is not considered as relevant here. To the best of our knowledge this is the first reported approach on investigating parallel TBox classification.

2 Architecture

This section describes the architecture of the implemented system and its underlying algorithms for thread-based parallel classification. In contrast to starting to implement a parallel TBox classifier, we decided to first conduct a field study with the goal to evaluate the impact of parameters such as number of threads, number of concepts (also called partition size) to be inserted per thread, and strategy to partition a given set of concepts, on the completeness of the classifier if one assumes a type of parallelization that deliberately sacrifices completeness. Using such a strategy we wanted to get some experience about the quality of a sound but incomplete parallel classifier, or, in other words, we wanted to find out about the amount of work needed to find and add missed subsumptions.

In order to focus better on this study we developed a simulator using a multi-threaded architecture which utilizes an input file created by Racer. This file contains a list of concept names to be classified into a taxonomy and information about them. This information replaces the standard tableau reasoning procedure and is also used as lookup data for verifying the parallel classification results. The decision to avoid the implementation of tableau algorithms makes the simulator independent of a particular DL logic. Racer generates this file for a given OWL-DL ontology from a complete TBox classification. The information about a concept name available in the input file contains the concept's satisfiability status, a pseudo model (if satisfiable), its so-called told subsumers and disjoints, and parents and children (in the complete taxonomy). The information about the parents and children is used to compute the set of ancestors and descendants of a concept. This information is only used for (i) emulating a tableau subsumption test, i.e., by checking whether a possible subsumer (subsumee) is in the list of ancestors (descendants) of given concept, and (ii) in order to find missing subsumptions in the taxonomy computed by the parallel classifier.

Once the simulator has read in this file, the told subsumer information is passed to a preprocessing algorithm which creates a taxonomy skeleton based on the already known (told) subsumptions and generates a topological order list (e.g., using a depth first traversal) from this skeleton. In the topological order list, from left to right, parent concepts precede their children concepts. To our surprise, the results of our experiments indicate that our initial assumption about the adequacy of using a topological order for parallel processing is not true although this ordering is a good choice for the sequential processing.

To implement the concurrency in our system, at least two memory management approaches could be taken into account by using either (i) sets of local trees (so-called PARTREE approach) or (ii) one global tree. In the PARTREE algorithm [13] a local tree is assigned to each thread, and after all the threads have finished the construction of their local hierarchy, the local trees are merged into one global tree. TBox classification through a local tree algorithm would not need any communication or synchronization between the threads. PARTREE is well suited for distributed systems which do not have shared memory. On the other hand, the global tree approach implements a shared space which is accessible to different threads running in parallel. The global tree approach was chosen because it better fits to our envisioned multi-core environment. To ensure data integrity a lock mechanism for single nodes is used. This allows a proper lock granularity and helps to increase the number of simultaneous write accesses to the subsumption hierarchy under construction.

In order to avoid unnecessary tree traversals and tableau subsumption tests when computing the subsumption hierarchy, the parallel classifier adapted the enhanced traversal method [1], which is an algorithm that was designed for sequential execution. This technique consists of two phases for each concept to be inserted, where the top-search determines its parents or predecessors and the bottom-search phase its children or successors. Algorithm 1 and 2 outline the traversal procedures for the top-search phase.

Algorithm 1 top_search(*new*,*current*)

```
mark(current,'visited')
pos-succ ← ∅
for all y predecessor of current do
  if enhanced_top_subs(y,new) then
    pos-succ ← pos-succ ∪ {y}
if pos-succ = ∅ then
  return current
else
  result ← ∅
  for all y ∈ pos-succ do
    if y not marked 'visited' then
      result ← result ∪ top_search(new,y)
  return result
```

The procedure top_search outlined in Algorithm 1 traverses the taxonomy top-down from a current concept and tries to push the new concept down the taxonomy as far as possible. It uses an auxiliary procedure enhanced_top_subs (outlined in Algorithm 2).

Algorithm 2 enhanced_top_subs(y,c)

```
if y marked 'positive' then
  return true
else if y marked 'negative' then
  return false
else
  for all z successor of y do
    if enhanced_top_subs(z,c) and y is subsumed by c then
      mark(y,'positive')
      return true
    else
      mark(y,'negative')
      return false
```

In a symmetric manner the procedure bottom_search traverses the taxonomy bottom-up from a current concept and tries to push the new concept up the taxonomy as far as possible. It uses an auxiliary procedure enhanced_bottom_subs. Both procedures are omitted the sake of brevity.

The procedure parallel_tbox_classification is sketched in Algorithm 3. It is called with a list of named concepts and sorts them in a topological order w.r.t. to the initial taxonomy created from the already known predecessors and successors of each concept (using the told subsumer information). Alternatively, the procedure parallel_tbox_classification can be also executed with a random order of the given concept list. The classifier assign partitions with a fixed or dynamically increased size from the concept list to idle threads and activates idle threads with their assigned partition using the procedure insert_ partition sketched in Algorithm 4. All threads work in parallel. The exists either a fixed number of threads or the number of threads grows dynamically.

The procedure insert_ partition inserts all concepts of a given partition into the global taxonomy. For updating a concept or its predecessor or successors, it locks the corresponding nodes. It first performs for each concept the top-search phase (starting from the top concept $\top$) and afterwards the bottom-search phase (starting from the bottom concept $\bot$).

3 Field Study

In order to evaluate the adequacy of our proposed algorithms in practice, and also to assess the performance of our parallel TBox classification algorithm, we configured our system so that it runs various experiments over ontologies with

Algorithm 3 parallel_tbox_classification(*concept_list,shuffle_flag*)

```
topological_order_list ← topological_order(concept_list)
if shuffle_flag then
   topological_order_list ← random_shuffle(topological_order_list)
repeat
   assign each idle thread t_i a partition p_i from topological_order_list
   run idle thread t_i with insert_partition(p_i)
until all concepts in topological_order_list are inserted
compute missing subsumptions and ratio
print statistics
```

Algorithm 4 insert_ partition(*partition*)

```
for all new ∈ partition do
   parents ← top_search(new,⊤)
   lock(new)
   set predecessors of new to parents
   for all pred predecessor of new do
      lock(pred)
      add new to successors of pred
      unlock(pred)
   unlock(new)
   children ← bottom_search(new,⊥)
   lock(new)
   set successors of new to children
   for all succ successor of new do
      lock(succ)
      add new to predecessors of succ
      unlock(succ)
   unlock(new)
```

Table 1. Used test ontologies.

Ontology Name	DL Expressivity	No. of named concepts
Galen	$\mathcal{SHF}$	2,730
Galen1	$\mathcal{ALC}$	2,730
Umls-2	$\mathcal{ALCHIN}$	9,479

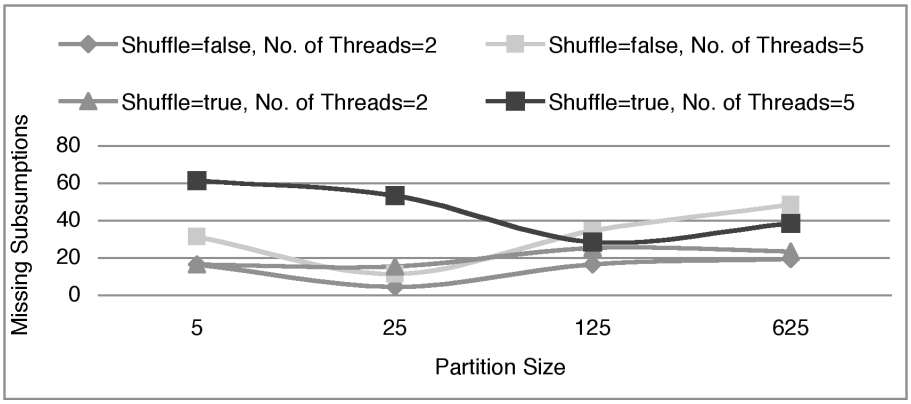

Fig. 1. Scenario 1: Missing subsumptions in Galen (2 and 5 threads).

various sizes, complexity and expressivity. The simulation work we conducted mainly consists of two parts. In the first part we use the topological order list as the input for our parallel TBox classification. In the second part, we randomly shuffle the topological order, based on a custom random shuffle algorithm, and thereafter we use the shuffled list as the input for our simulation. Once the shuffle order has been computed, it is saved and later reused for our tests. The simulations were run on three different scenarios using the ontologies shown in Table 1. We only have preliminary result for Galen, Galen1, and Umls-2. As already mentioned, two parameters which influence the parallel TBox classification, namely partition size, and the number of threads, vary in the following three scenarios. The results are measured on the basis of the number of missed subsumptions in the taxonomy produced by parallel classifier vs. the complete taxonomy.

3.1 Partition size and number of threads are constant

In this scenario, the topological order list is divided into partitions of a constant size for each test. After each test the partition size is increased by a factor of 5. As shown in Figure 1, when we configured our system to run with two threads, the number of missing subsumptions decreases and then slightly increases as the partition size increases. Running the simulation with 5 threads, the number of missing subsumptions decreases when the topological order is not shuffled;

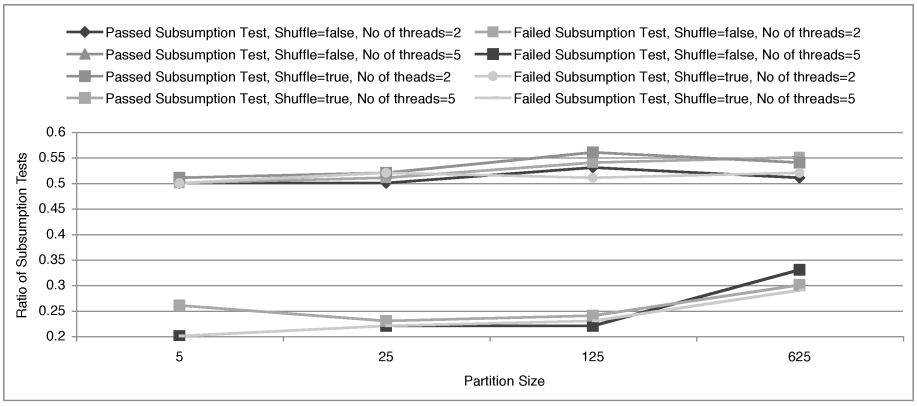

Fig. 2. Scenario 1: Ratio of passed and missed subsumption tests in Galen.

however, if it is shuffled the missing subsumptions decreases and then dramatically increases as the partition size increases. In the worst case, the missing subsumptions make 2% of all detected subsumptions.

Figure 2 shows the ratio for passed subsumption tests[2] and failed subsumption tests.[3] In this figure, we notice that the increase of the number of threads has a linear effect on ratio. In other words, when we add more threads to our system, the ratio linearly decreases and that is what we expected. Conducting the same tests for Umls-2 (not shown here) one gets at most 1% of missed subsumptions and the ratio has the same trend as Galen.

3.2 Partition size is dynamic and grows exponentially and the number of threads is constant

In this scenario, the partition size grows exponentially (5^n), however the number of threads remains constant in each test run. As displayed in Figure 3, in all the cases the missing subsumptions decrease except for the case when we have shuffled the topological order list and the number of threads is equal to 5. The percentage of missing subsumptions in the worst case is 0.25%. If the partitions grow in size, the algorithms perform closer to the sequential case; having the ratios (passed, failed subsumption tests) close to 1 (as in the sequential case) supports our conjecture (see Figure 4). For Umls-2 (not shown here) the number of missing subsumptions is 0.02% and the ratio follows a trend close to Galen.

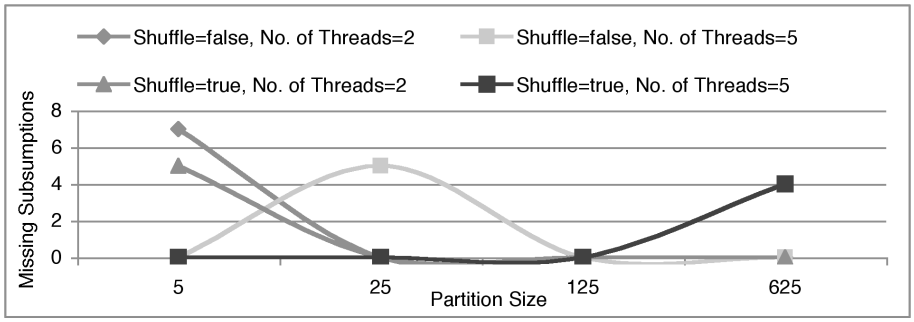

Fig. 3. Scenario 2: Missing subsumptions in Galen (2 and 5 threads).

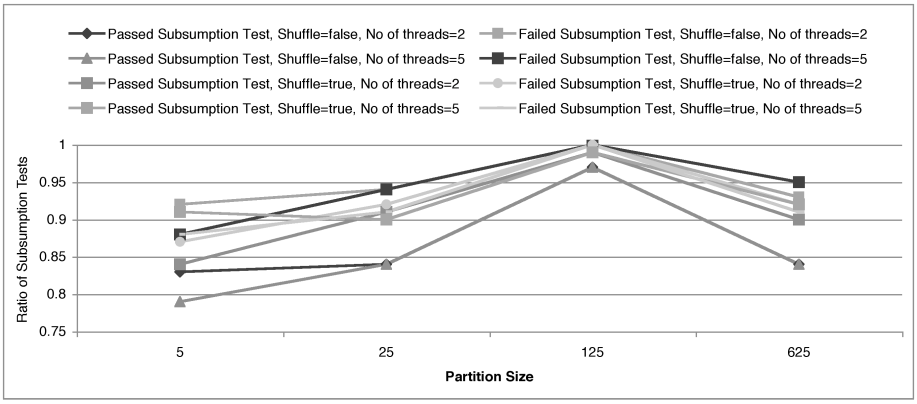

Fig. 4. Scenario 2: Ratio of passed and missed subsumption tests in Galen.

3.3 Number of threads is dynamic and grows exponentially

In this scenario, the number of threads grows exponentially (2^n) but the partition size remains constant. Figure 5 shows the test results for the cases when a topological and a random order is used. In this figure, we observe that the missing subsumptions varies up and down based on the initial value for the number of threads. Figure 6 displays the ratio for passed and failed subsumption tests. This figure shows that we have a smooth decrease of the ratio.

[2] Total number of passed (successful) subsumption tests in the parallel case divided by the total number of passed subsumption tests in the sequential case.

[3] Total number of failed (unsuccessful) subsumption tests in the parallel case divided by the total number of failed subsumption tests in the sequential case.

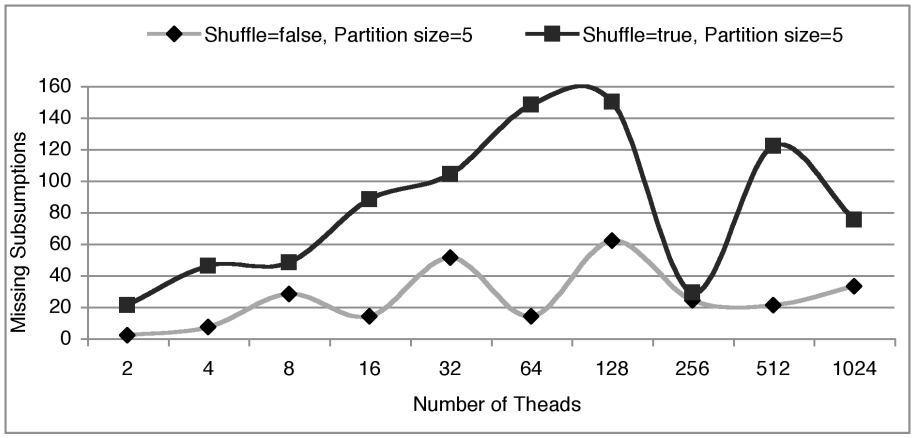

Fig. 5. Scenario 3: Missing subsumptions in Galen (increasing number of threads).

4 Discussion and Conclusion

While there exists some work on parallel algorithms for description logic reasoning [3, 12] and on parallel reasoning for first-order theorem proving (with completely different proof techniques based on resolution), parallel TBox classification has not been addressed yet. There has also been substantial work on reasoning through modularity and partitioning knowledge bases (e.g., [5, 8, 7]). In [5], the proposed greedy algorithm performs automated partitioning, and the authors have investigated how to reason effectively with partitioned sets of logical axioms that have overlap in content and may even require different reasoning engines. Their partition-based reasoning algorithms have been proposed for reasoning with logical theories in propositional and first-order predicate logic that are decomposed into related partitions of axioms.

As pointed out, there has also been extensive work on modularity. In [8], a logic-based framework for modularity of ontologies is proposed. This formalization is very interesting for the ontologies that can be modularized. For these cases, every module can be assigned to a particular thread and can be classified in parallel. The approach reported in [7] also proposed a technique for incremental ontology reasoning that is, reasoning that reuses the results obtained from previous computations. This technique is based on the notion of a module and can be applied to arbitrary queries against ontologies expressed in OWL-DL. The approach focused on a particular kind of modules that exhibit a set of compelling properties and apply their method to incremental classification of OWL-DL ontologies. The techniques do not depend on a particular reasoner or reasoning method and can be easily implemented in any existing prover.

In our paper, we have described an architecture for parallelizing well-known algorithms for TBox classification. Our work is targeted for ontologies where independent partitions cannot be constructed; therefore we did not use the pre-

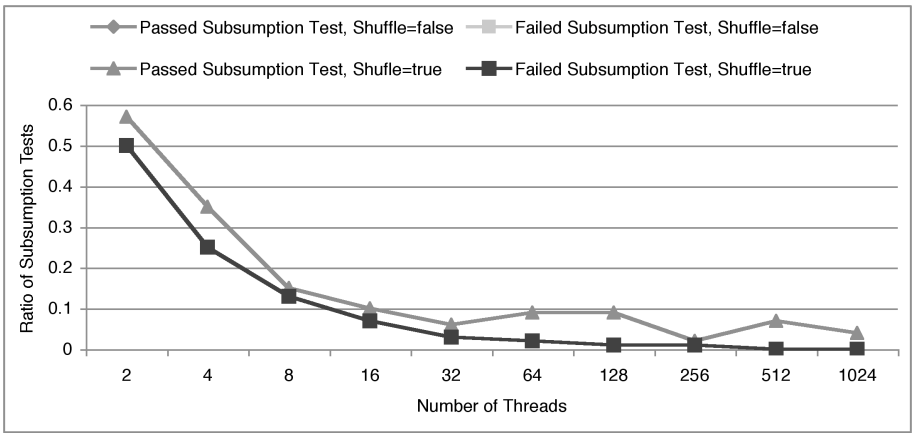

Fig. 6. Scenario 3: Ratio of passed and missed subsumption tests in Galen.

viously mentioned approaches in our system. The experimental evaluation of our proposed technique shows that the result are very promising because the number of missed subsumptions is surprisingly small. As we discussed in previous sections, due to missing subsumptions in classification, the algorithm is sound but incomplete. In our next steps we plan to ensure completeness by adding a recovery or repair phase. We will also plan to implement the technique in a multi-core / multi-processor environment.

References

1. F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4(2):109–132, 1994.
2. F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNAI*, pages 287–291. Springer-Verlag, 2006.
3. F. Bergmann and J. Quantz. Parallelizing description logics. In *Proc. of 19th Ann. German Conf. on Artificial Intelligence*, LNCS, pages 137–148. Springer-Verlag, 1995.
4. J. Dolby, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. Scalable semantic retrieval through summarization and refinement. In *21st Conf. on Artificial Intelligence (AAAI)*, pages 299–304. AAAI Press, 2007.
5. Amir Eyal and Sheila Mcllraith. Partition-based logical reasoning for first-order and propositional theories. *Artifical Intelligence*, 162(1-2):49–88, 2005.
6. A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary Abox: Cutting ontologies down to size. In *Proc. of Int. Semantic Web Conf. (ISWC)*, volume 4273 of *LNCS*, pages 343–356. Springer-Verlag, 2006.
7. Bernardo Cuenca Grau, Christian Halaschek-Wiener, , and Yevgeny Kazakov. History matters: Incremental ontology reasoning using modules. In *Proceedings of the*

6th International Semantic Web Conference (ISWC 2007), Busan, South Korea, November 11-15. Springer, 2007.
8. Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. A logical framework for modularity of ontologies. In *In Proc. of the 20th International Joint Conference on Artificial Intelligence(IJCAI 2007), Busan, South Korea, November 11-15*, pages 298–303, 2007.
9. Y. Guo and J. Heflin. A scalable approach for partitioning OWL knowledge bases. In *Proc. 2nd Int. Workshop on Scalable Semantic Web Knowledge Base Systems, Athens, USA*, pages 47–60, 2006.
10. V. Haarslev and R. Möller. On the scalability of description logic instance retrieval. *Journal of Automated Reasoning*, 2007. 54 pages. Submitted for review.
11. Intel Inc. Innovating for today and the future. http://www.intel.com/multi-core/ [Accessed February, 2008].
12. T. Liebig and F. Müller. Parallelizing tableaux-based description logic reasoning. In *Proc. of 3rd Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS '07), Vilamoura, Portugal, Nov 27*, LNCS. Springer-Verlag, 2007.
13. H. Shan and J. P. Singh. Parallel tree building on a range of shared address space multiprocessors: Algorithms and application performance. In *Proc. of the First Merged Symp. IPPS/SPDP 1998)*, pages 475–484, 1998.
14. Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
15. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning*, LNAI, pages 292–297. Springer-Verlag, 2006.