Query Answering and Scaling Extensions of Konclude

Andreas Steigmiller[1], Birte Glimm[2] and Thorsten Liebig[3]

[2]*Ulm University, Germany*
[3]*derivo GmbH, Germany*

Abstract
Konclude is a well-performing reasoner for ontologies formulated via the Web Ontology Language. In this paper, we give an overview of new or extended optimizations that lead to additional improvements, such as an individual derivations cache for handling large amounts of assertions more efficiently. We further describe new (functional) extensions (e.g., RDF/SPARQL support) and capabilities (such as conjunctive query answering) that are ready for first practical use-cases. Last but not least, we show some evaluations and comparisons for the new version of Konclude.

1. Introduction

Konclude [1] is designed as a high-performance reasoner for the Web Ontology Language (OWL), supporting the Description Logic (DL) $\mathcal{SROIQ}(\mathcal{D})$ [2] with nominal schemas, i.e., OWL 2 with most datatypes and some kind of support for rule-based knowledge. It is implemented in C++ by using the Qt framework and it is freely available under the Lesser GPL v3. Although the design focus of the system was to perform well for the expressive OWL 2 DL profile, it also has excellent and competitive performance on various reasoning tasks of the less expressive profiles. In fact, many implemented algorithms follow a pay-as-you-go approach, where one tries to keep the overhead for more expressive language features at a minimum. On the one hand, this is achieved by using specialized procedures such as completion/consequence-based saturation procedures in the reasoning system that are specifically designed for less expressive fragments and, hence, perform eminently well. On the other hand, the fully-fledged reasoning algorithms (e.g., tableau) exploit intermediate results/consequences from the specialized procedures and operate only on parts of the knowledge base that are not already sufficiently handled. Moreover, most optimization techniques are developed and implemented in a way such that they work with all language features, i.e., they are not simply deactivated if, for example, nominals occur, but are restricted to parts of the knowledge base that are not affected by nominals.

SemREC'21: Semantic Reasoning Evaluation Challenge, ISWC'21, Oct 24 – 28, Albany, NY
birte.glimm@uni-ulm.de (B. Glimm); liebig@derivo.de (T. Liebig)
https://www.uni-ulm.de/in/ki/inst/alumni/dr-andreas-steigmiller/ (A. Steigmiller); https://www.uni-ulm.de/in/ki/inst/team/prof-dr-birte-glimm/ (B. Glimm); https://www.derivo.de/ (T. Liebig)

Although Konclude showed remarkable performance in international competitions by winning several reasoning disciplines (cf. previous OWL Reasoner Evaluation Competitions [3, 4]), it struggled a bit with very large ontologies. On the one hand, this was caused by an inefficient management of assertions, which were handled as ordinary axioms (such that for each class/property assertion an internal concept expression had to be created). On the other hand, tableau-based methods try to create one model abstraction covering all assertions, which can require a huge amount of memory if there are many. There were also several shortcomings of Konclude, which had to be addressed. In fact, Konclude had no efficient way to handle more complex queries. Users who were interested in the instances of a complex class expression, had to add a new class name in the ontology that was defined as equal to that class expression, realize this extended ontology, and retrieve the instances of the newly added class name.

In the following, we describe how these issues were addressed by new or updated optimizations and by functional extensions (Section 3). Before going into more details, we sketch the overall architecture of Konclude (Section 2), which, however, did not change significantly. We also compare with the previous version (Section 4).

2. System Architecture

Konclude is mainly designed as a server application that provides reasoning services, i.e., clients usually communicate with Konclude over network via different interfaces. Besides OWLlink, the new version of Konclude (v0.7.0) also supports parts of SPARQL for querying and updating the knowledge bases. The command line interface of Konclude also supports both protocols (by reading the requests/queries from a file), but also provides direct commands to get the results of several standard reasoning tasks.

Konclude has various manager components that communicate with each other via events (cf. Figure 1). After parsing commands from the input, every command is delegated to a corresponding manager (e.g., Classification Manager, Answering Manager), which ensures the completion of it and may delegate sub-commands to other components. For example, if the consistency checking of an ontology is requested via command line, then commands are created for (i) parsing and loading the ontology into a temporal knowledge base, (ii) preprocessing the knowledge base, and (iii) check the consistency of it (via the Precomputing Manager). The commands can have dependencies to other commands such that they are processed in the correct order. For instance, the ontology must be loaded/updated before consistency is to be checked. Components for higher-level reasoning tasks (such as classification, realization, query answering) typically use the results of more basic services, either by requiring them beforehand or requesting them dynamically. For example, query answering is only started if classification is done, but it may dynamically request the realization of certain classes and/or properties.

Computationally intensive work (such as model construction) is outsourced to the Kernel, which manages a fixed number of worker threads such that the manager components remain responsive and can focus on creating several work packages to enable parallelization. Caches (for intermediate results) are also realised in form of such components and

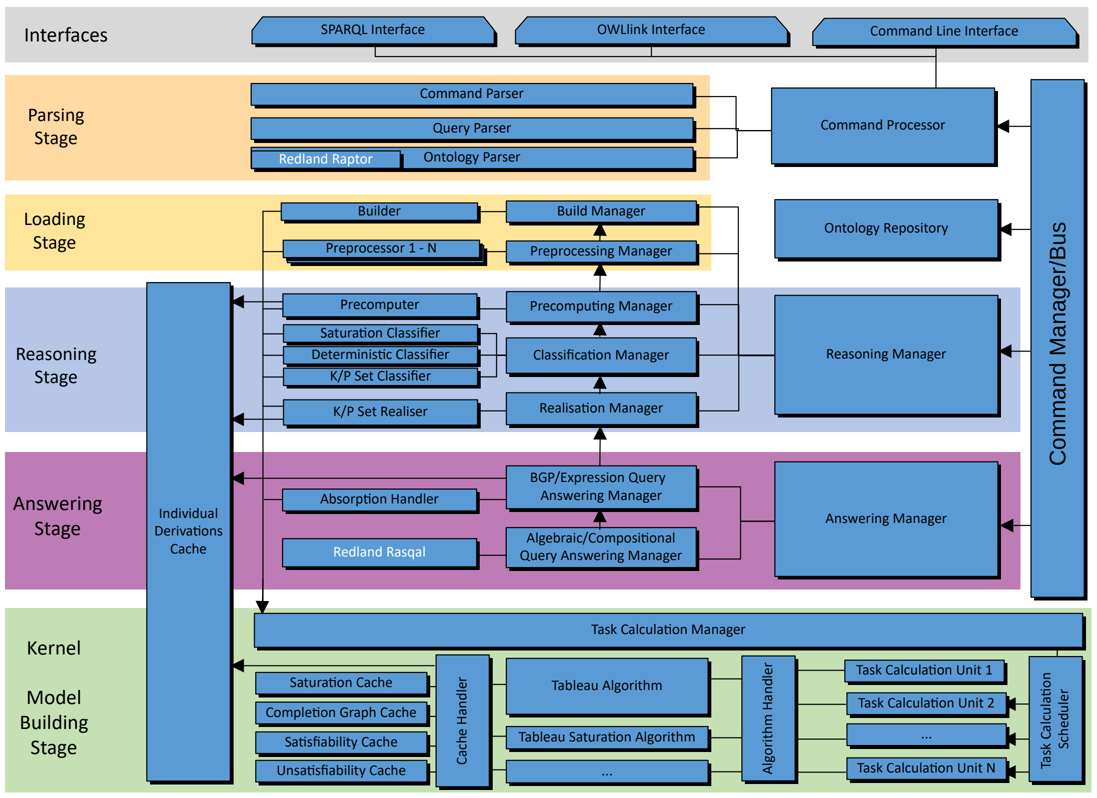

Figure 1: System architecture of the reasoning system Konclude

usually have a separate manager to which events are sent for integrating updates. The data structures of the caches are organized such that they can be read in parallel without blocking. The new individual derivations cache is also accessed by different manager components in order to be able to create and schedule corresponding reasoning tasks. For example, the realization process retrieves for which individuals classes are only possibly derived and creates model construction tasks that check whether one of these individual is actually an instance of such a class. Also see Konclude's system description for a more detailed overview of the architecture [1].

3. Improvements and Enhancements

Several aspects have been updated in Konclude to improve the performance and to extend the functionality. This includes smaller improvements such as a more efficient representation of facts, but also some bigger changes as well as new optimizations, as sketched in the following.

3.1. Optimization Adaptations

The realization reasoning process has been improved significantly in several ways. First, the data structures for managing possible and known class instantiations are optimized such that they require less memory and can be created concurrently and more quickly. Moreover, the realization process does not even require separate data structures for simpler ontologies (which are mainly deterministic) and simply reads the instantiations from cached model abstractions. Last but not least, if several individuals have to be checked whether they are instances of a certain class, then Konclude is able to merge more and more of these individuals into the same model abstraction to quickly determine non-instantiations. The latter is similarly to bulk processing and binary retrieval optimizations [5], but without introducing a significant overhead.

Tableau-based systems, such as Konclude, apply the tableau expansion rules to all (possibly anonymous) individuals, which typically results in tree-based structures that reflect the restrictions of the axioms. In order to avoid that an enormous number of these tree-based model abstractions must be built for ontologies with many assertions and individuals, we developed a new optimization for consistency checking, which allows for processing small parts of the individuals stepwise by using a so-called *individual derivations cache* to store, retrieve, and align the consequences of the separately created partial models [6]. To be more precise, for constructing the model abstraction for a small part of the assertions, we retrieve already derived consequences for occurring individuals from the cache and reuse them. If there is some incompatibility with the cache (e.g., due to non-deterministically derived/possible consequences), then we expand the model construction to neighbors until compatibility is achieved. We limit the expansion by marking the remaining individuals in the cache such that they are processed later separately.

3.2. Parallelization Enhancements

Konclude has a range of different parallelization techniques that improve certain reasoning aspects on multi-core systems. To further improve the parallelization, we extended the previously sketched caching technique appropriately. In fact, different assertions can be processed in parallel by different worker threads, i.e., each worker thread constructs a partial model abstraction for a small part of the assertions and extracts relevant information that is sent to the cache, which then integrates the new information asynchronously. If some threads derive some conflicting/incompatible consequences such that an alignment with other parts is not directly possible, then the corresponding parts are reprocessed later (potentially repeatedly). For most ontologies, this is, however, hardly required, i.e., this new parallelization technique significantly speeds up consistency checking for ontologies with many assertions as long as these ontologies are not too constrained. To be more precise, if the reasoner has to make many non-deterministic decisions and only few branches can be expanded to abstractions of models, then dependency directed backtracking has to be performed over the cache, which is usually not very efficient.

We further improved the parallel processing of other tasks. In fact, large triple files can be parsed and indexed in parallel. Moreover, several query answering steps are parallelized (e.g., joining of different intermediate results of different sub-queries) and even the serialization of answer results can now be performed in parallel.

3.3. Functional Extensions

There are several new major features in Konclude: Most notably, Konclude has now support for conjunctive query answering. This is realized in form of an absorption-based query answering approach [7], where the queries are rewritten into simple DL-axioms by using so-called binder concepts. These binder concepts "remember" the element for which they occur by encoding a binding for a specified variable. By propagating these bindings according to the absorption, it is possible to detect when queries are possibly satisfied in the model abstraction. This works as long as the reasoning procedure only has to consider a limited number of so-called new nominals, which is the case for most real-world ontologies. To make the absorption-based query answering approach more efficient, we also integrate results from other reasoning tasks such as realization. Last but not least, we integrated many minor query answering optimizations, such as query evaluation planning, query materialization, etc. By using the optional integrable Redland RDF Library, Konclude can even process more complex SPARQL queries. In fact, Konclude itself only processes the SPARQL basic graph patterns (BGPs) and then returns the result to the Redland query engine (Rasqal), which then evaluates the SPARQL algebraic operator (such as UNION, OPTIONAL, etc.). The Redland RDF Libraries also give Konclude the ability to parse and interpret RDF triple files with encoded OWL axioms.

4. Experiments and Evaluation

Several new aspects of Konclude have already been evaluated intensively. In fact, evaluations have shown that the absorption-based query answering technique works very well for many real-world ontologies even if they use expressive language features [7]. Moreover, there are basically no real-world ontologies for which the new nominals are threatening termination of the approach.

Evaluations for the new stepwise consistency checking technique showed that they enable the handling of very large and expressive ontologies [6]. The evaluations further showed that the parallelization techniques utilizing this stepwise handling lead to a significant speed up on multi-core systems.

In the following, we give a brief overview of how the update impacted the performance of Konclude, i.e., we compare the new version of Konclude (v0.7.0-1138) with the one from the ORE 2015 competition (v0.6.1-527) on the ontologies of the classification task/files of ORE 2015 [8].[1] We run the evaluation within a Ubuntu Docker container on an Intel i7-6900K CPU @ 3.20 GHz with Windows 10 as host operating system. The reasoners were restricted to 10 GB RAM. With a timeout of 150 seconds, the old Konclude version

[1]See https://gitlab.com/kncludeeval/semrec21 for the script/docker image to reproduce the evaluation. Binaries and source code of Konclude are available at https://github.com/konclude/Konclude.

Table 1
Classification results for OWL2Bench ontologies in seconds (T/O stands for timeout)

Konclude	QL		EL		RL		DL	
version	1	10	1	10	1	10	1	10
v0.6.1-527	2.5	32.1	2.9	40.3	2.5	37.6	T/O	T/O
v0.7.0-1138	1.9	18.9	2.0	16.1	2.1	20.7	T/O	T/O

(v0.6.1-527) classified all 200 ontologies within 1836 seconds, whereas the new Konclude version (v0.7.0-1138) required 1389 seconds. The new version had 2 timeouts, whereas the previous one reached the time limit for 4 ontologies.

We further evaluated some of the smaller versions of the OWL2Bench ontologies [9]. Table 1 shows the classification times for the different profile variants for the sizes 1 and 10. It is observable that the new version is faster for these ontologies, but both versions fail for the OWL 2 DL variant. Although the new version of Konclude is able to test the consistency of these OWL 2 DL variants by using the new stepwise handling with the cache for synchronization/alignment, it still struggles with higher level reasoning tasks. This could be due to some subsumption tests that affect large parts of the model.

5. Conclusions

We presented an overview over the changes as well as new features of Konclude and showed the results of a comparison with a previous version from the ORE 2015 competition. Although the new version performs generally better, there still seems to be room for improvements.

References

[1] A. Steigmiller, T. Liebig, B. Glimm, Konclude: system description, J. of Web Semantics 27 (2014).

[2] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible $\mathcal{SROIQ}$, in: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06), AAAI Press, 2006, pp. 57–67.

[3] S. Bail, B. Glimm, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia, A. Steigmiller, Summary ORE 2014 competition, in: Proc. 2nd Int. Workshop on OWL Reasoner Evaluation (ORE'13), volume 1207, CEUR, 2014.

[4] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, A. Steigmiller, The OWL reasoner evaluation (ORE) 2015 competition report, in: Proc. 11th Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS'15), 2015, pp. 2–15.

[5] V. Haarslev, R. Möller, On the scalability of description logic instance retrieval, J. of Automated Reasoning 41 (2008) 99–142.

[6] A. Steigmiller, B. Glimm, Parallelised abox reasoning and query answering with expressive description logics (2021).

[7] A. Steigmiller, B. Glimm, Absorption-based query answering for expressive description logics, in: Proc. 18th Int. Semantic Web Conf. (ISWC'19), LNCS, Springer, 2019, pp. 593–611.

[8] B. Parsia, N. Matentzoglu, R. S. Gonçalves, B. Glimm, A. Steigmiller, The OWL reasoner evaluation (ORE) 2015 competition report, J. of Automated Reasoning 59 (2017) 455–482.

[9] G. Singh, R. Mutharaju, P. Kapanipathi, Owl2bench dataset, 2021. URL: https://doi.org/10.5281/zenodo.4764368. doi:10.5281/zenodo.4764368.