Experimenting with ELK Reasoner on Android

Yevgeny Kazakov and Pavel Klinov

The University of Ulm, Germany
{yevgeny.kazakov, pavel.klinov}@uni-ulm.de

Abstract. This paper presents results of a preliminary evaluation of the OWL EL reasoner ELK running on a Google Nexus 4 cell phone under Android 4.2 OS. The results show that economic and well-engineered ontology reasoners can demonstrate acceptable performance when classifying ontologies with thousands of axioms and take advantage of multi-core CPUs of modern mobile devices. The paper emphasizes the engineering aspects of ELK's design and implementation which make this performance possible.

1 Introduction and Motivation

Mobile computing has been on the rise for the last decade and the Semantic Web applications are no exception. Increasingly many mobile applications can benefit from semantic technologies, especially when it comes to context-aware information processing [1]. Specifically, it is desirable to be able to combine data obtained by various mobile IO devices (sensors), such as GPS devices, Wifi or cellular networks, etc., with background information supplied by ontologies. For example, an intelligent application can use an ontology representing various kinds of businesses, e.g., restaurants, grocery stores, etc., with facts determining the user's location to suggest places to go. Or a medical application can use a medical ontology in conjunction with private user's medical data to provide counselling or other services. Such applications require reasoning to make use of implicit knowledge and sometimes *may* require reasoning to happen on the device itself (rather than on a remote server or in the cloud) for reasons such as privacy [2].

Recently there has been interest in ontology reasoners designed specifically for mobile platforms. Some researchers claim that mobile devices, being resource-constrained, require reasoner developers design their reasoning engines *specifically* for mobile computing environments [3]. Few such reasoner implementation and evaluation reports are available, for example, Delta reasoner [3] and Pocket KR Hyper [2]. At the same time we are not aware of any experience of porting existing reasoners to mobile platforms. This is a little surprising since modern devices boast substantial computational power. Having the same reasoning core working for both desktop/server and mobile devices with minimal changes would be attractive from the maintainability point of view.

This paper is a step in that direction. It presents an evaluation of ELK, a concurrent reasoner for OWL EL profile [4] implemented in Java, on Google's Nexus 4 phone under Android 4.2 operating system. The results demonstrate that ELK is able to provide acceptable classification performance on mid-to-large sized ontologies (up to tens of thousands of axioms) and is even able to classify SNOMED CT, one of the largest medical ontologies, which remains a challenge for many OWL reasoners even on desktops.

$$\mathbf{R_0}\ \frac{}{C \sqsubseteq C}: C \text{ occurs in } \mathcal{O} \qquad \mathbf{R}^+_\sqcap\ \frac{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}{C \sqsubseteq D_1 \sqcap D_2}: D_1 \sqcap D_2 \text{ occurs in } \mathcal{O}$$

$$\mathbf{R}_\top\ \frac{}{C \sqsubseteq \top}: C \text{ and } \top \text{ occur in } \mathcal{O} \qquad \mathbf{R}_\exists\ \frac{E \sqsubseteq \exists R.C \quad C \sqsubseteq D}{E \sqsubseteq \exists S.D}: \begin{array}{l}\exists S.D \text{ occurs in } \mathcal{O}\\ R \sqsubseteq^*_\mathcal{O} S\end{array}$$

$$\mathbf{R}_\sqsubseteq\ \frac{C \sqsubseteq D}{C \sqsubseteq E}: D \sqsubseteq E \in \mathcal{O} \qquad \mathbf{R}_\circ\ \frac{E \sqsubseteq \exists R_1.C \quad C \sqsubseteq \exists R_2.D}{E \sqsubseteq \exists S.D}: \begin{array}{l}S_1 \circ S_2 \sqsubseteq S \in \mathcal{O}\\ R_1 \sqsubseteq^*_\mathcal{O} S_1\\ R_2 \sqsubseteq^*_\mathcal{O} S_2\end{array}$$

$$\mathbf{R}^-_\sqcap\ \frac{C \sqsubseteq D_1 \sqcap D_2}{C \sqsubseteq D_1 \quad C \sqsubseteq D_2}$$

Fig. 1. The inference rules for reasoning in $\mathcal{EL}^+$

Importantly, the changes between the standard and mobile versions of ELK are negligible. This work is preliminary, in particular, it does not aim at comparing performance of ELK on a mobile device to that of other existing OWL reasoners (or across different mobile devices).

2 Preliminaries

In this paper, we will focus on the DL $\mathcal{EL}^+$ [5], which can be seen as $\mathcal{EL}^{++}$ [6] without nominals, datatypes, and the bottom concept $\bot$. $\mathcal{EL}^+$ concepts are defined using the grammar $\mathbf{C} ::= A \mid \top \mid C_1 \sqcap C_2 \mid \exists R.C$, where A is an *atomic concept*, R an *atomic role*, and C, C_1, $C_2 \in \mathbf{C}$. $\mathcal{EL}^+$ *axiom* is either a *concept inclusion* $C_1 \sqsubseteq C_2$ for C_1, $C_2 \in \mathbf{C}$, a *role inclusion* $R \sqsubseteq S$, or a *role composition* $R_1 \circ R_2 \sqsubseteq S$, where R, R_1, R_2, S are role names. $\mathcal{EL}^+$ *ontology* $\mathcal{O}$ is a finite set of $\mathcal{EL}^+$ axioms. Given an ontology $\mathcal{O}$, we write $\sqsubseteq^*_\mathcal{O}$ for the smallest reflexive transitive binary relation over roles such that $R \sqsubseteq^*_\mathcal{O} S$ holds for all $R \sqsubseteq S \in \mathcal{O}$.

Entailment of axioms by an ontology is defined in a usual way; a formal definition can be found, e.g., in [5]. A concept C is *subsumed* by D w.r.t. $\mathcal{O}$ if $\mathcal{O} \models C \sqsubseteq D$. In this case, we call $C \sqsubseteq D$ an *entailed subsumption*. The *ontology classification task* requires to compute all entailed subsumptions between atomic concepts occurring in $\mathcal{O}$.

The $\mathcal{EL}^+$ reasoning procedure implemented in ELK works by applying inference rules to derive subsumptions between concepts. Figure 1 shows the rules from $\mathcal{EL}^{++}$ [6] restricted to $\mathcal{EL}^+$, but presents them in a way that does not require the normalization stage [4]. Some rules have side conditions given after the colon that restrict the expressions to which the rules are applicable. For example, rule $\mathbf{R}^+_\sqcap$ applies to each C, D_1, D_2, such that $D_1 \sqcap D_2$ occurs in $\mathcal{O}$ with premises $\{C \sqsubseteq D_1,\ C \sqsubseteq D_2\}$, and the conclusion $C \sqsubseteq D_1 \sqcap D_2$. Note that the axioms in the ontology $\mathcal{O}$ are only used in side conditions of the rules and never used as premises of the rules.

The rules in Figure 1 are complete for deriving subsumptions between the concepts occurring in the ontology. That is, if $\mathcal{O} \models C \sqsubseteq D$ for C and D occurring in $\mathcal{O}$, then $C \sqsubseteq D$ can be derived using the rules in Figure 1 [6]. Therefore, in order to classify the ontology, it is sufficient to compute the closure under the rules and take the derived subsumptions between atomic concepts.

Computing the closure under inference rules, such as in Figure 1, can be performed using a well-known *forward chaining* procedure presented in Algorithm 1 in an abstract

Algorithm 1: Abstract rule-based classification procedure

input : **C**: the set of named concepts from $\mathcal{O}$, R: a set of inference rules
output : Closure: a set of inferences closed under R

```
1 Closure, Todo ← ∅;
2 for A ∈ C do                                              /* initialize */
3     Todo ← Todo ∪ apply(R_0[A]) ∪ apply(R_⊤[A]);
4 while (exp ← Todo.poll() ≠ null) do                       /* compute closure */
5     if exp ∉ Closure then
6         Closure ← Closure ∪ exp;
7         for r ∈ R[exp, Closure] do
8             Todo ← Todo ∪ apply(r);
9 return Closure;
```

way. The algorithm works with *expressions* of the form $C \sqsubseteq D$ or $C \sqsubseteq \exists R.D$, where C and D are concepts and R is a role. It derives expressions by applying rules R in Figure 1. It collects those expressions to which all rules have been applied in a set Closure and the remaining ones in a queue Todo. The algorithm first initializes Todo with conclusions of the initialization rule $\mathbf{R_0}$, see lines 2–3. Then it repeatedly takes the next expression $exp \in$ Todo, inserts it into Closure if it does not occur there, and applies all applicable rules to it (lines 4–8). Informally, we use R[. . .] to denote selection of rules for specific premises and/or side conditions. The conclusions derived by the applied rules are then inserted in Todo.

ELK implements a concurrent version of Algorithm 1 which maintains a *context* for each concept that occurs on the left hand-side of an axiom in $\mathcal{O}$. Contexts maintain their own Todo queues and are processed in parallel threads of execution (referred to as *workers*). Details can be found in [4].

3 Evaluation on Google Nexus 4

This section present the results of a preliminary evaluation of ELK's classification performance on a Google Nexus 4 cell phone. The device runs under Android 4 OS and features a Qualcomm Snapdragon™ S4 Pro CPU (4 cores, 1.7 GHz) and 2 GB RAM, of which 500 MB was allocated to JVM. To put the results into a perspective, we also ran ELK on a PC with Intel Core i5-2520M 2.50GHz CPU with 8 GB RAM (JVM was allocated the same 500 MB).

Five $\mathcal{EL}$ ontologies often used for benchmarking $\mathcal{EL}$ reasoners have been selected for the experiments (the number of logical axioms given in parenthesis): Chemical Entities of Biological Interest (ChEBI, 67,182), the e-Mouse Atlas Project (EMAP, 13,730), and the Fly Anatomy (19,137) are some of large OBO Foundry[1] and Ontobee[2] ontologies that also include some non-atomic concepts. GO (28,896) is the older version of the

[1] http://www.obofoundry.org/
[2] http://www.ontobee.org/

Gene Ontology published in 2006. $\mathcal{EL}$-GALEN (36,547) is an $\mathcal{EL}^+$-restricted version of the GALEN ontology. All ontologies are freely available from the ELK website.[3]

Each ontology was classified with different, from 1 to 6, number of workers and the results are presented in Table 1. The most obvious experimental outcome is that classification on the cell phone is about two orders of magnitude slower than on a PC, i.e., the difference appears larger than in the mere computational power of the two systems (at least, if the latter is compared in terms of just CPU rate and the amount of RAM). Comparison of the "LI Ratio" columns reveals that the relative difference during the classification stage (CPU-bound processing) is larger than during the loading and indexing stage (mostly IO-bound). One possible explanation is that CPU caches, for which ELK's data structures are optimized (see the next section), are more effective on PC than on this cell phone. Difference in the RAM speed may have also played a role.

It can be noted that ELK's concurrent classification algorithm brings benefits on the cell phone just as well as on PC. The difference is especially visible between 1 worker and 2 (or more) workers. It is only visible for the classification stage because during indexing most time is spent on loading axioms from external memory and parsing.

Finally, we attempted to classify the official January 2013 release of SNOMED CT, one of the largest medical ontologies (296,529 axioms).[4] The intent was to push ELK (and the phone) to its limits. Tad surprisingly, ELK still managed to complete classification in 1h and 20m, out of which nearly 10m was spent on loading/indexing and the rest on reasoning. It has used nearly all (475 MB) memory available to JVM. For reference, it takes about 10s to classify SNOMED CT on a laptop with Intel Core i5-2520M 2.50GHz CPU and 4GB of RAM available to JVM.

4 Implementation Notes

This section provides some engineering details on implementation of ELK. The methods listed below are not specific to a particular computational platform. However, they are particularly relevant to mobile devices since they seek to reduce the memory footprint of the reasoner.

Entity filtering: In large ontologies it is often the case that some OWL entities (concept (sub)expressions or roles) appear in many axioms. ELK's internal entity filter guarantees that each entity is represented by precisely one Java object. This has two advantages: First, it reduces memory consumption and thus reduces the number of GC cycles. Second, it allows for fast equality checking by comparing references (basically, memory pointers). The latter is especially important for searching for an object in collections, e.g., sets or arrays.

Economic data structures: The ELK's classification algorithm operates with many collections of objects representing OWL entities, such as subsumers for a given concept, conjuncts in a given conjunctive concept expression, etc. The important thing is that most of those collections are small, i.e. usually up to hundred elements. ELK provides a custom array-based, cache-friendly hashtable implementation with linear prob-

[3] `https://code.google.com/p/elk-reasoner/wiki/TestOntologies`

[4] We did not include it in the main experiment since it would take too much time to vary the number of workers for it.

Table 1. Time (in ms) and memory usage (in MB) results for loading/indexing and classification on a Google Nexus 4 and a PC. The LI Ratio column shows the proportion of total time (in %) spent for loading and indexing the ontology.

		Google Nexus 4				PC		
Ontology	Workers	Load./Index.	Classif.	LI Ratio	Memory	Load./Index.	Classif.	LI Ratio
ChEBI	1	31,370	207,020	13	67	351	1,055	25
ChEBI	2	29,423	160,334	16	72	323	715	31
ChEBI	3	32,213	148,369	18	72	337	611	36
ChEBI	4	32,443	147,868	18	68	324	646	33
ChEBI	5	32,900	114,054	22	65	362	570	39
ChEBI	6	29,997	107,033	22	72	341	597	36
EMAP	1	20,667	6,970	75	23	366	93	80
EMAP	2	19,580	4,337	82	24	389	83	82
EMAP	3	20,311	3,750	84	25	413	72	85
EMAP	4	19,081	3,508	84	24	396	68	85
EMAP	5	19,921	3,467	85	23	383	73	84
EMAP	6	19,949	3,390	95	25	360	86	81
Fly Anatomy	1	7,882	31,478	20	22	195	276	39
Fly Anatomy	2	8,231	18,953	30	23	248	252	52
Fly Anatomy	3	9,143	16,951	35	24	256	223	51
Fly Anatomy	4	8,483	16,041	35	24	225	275	45
Fly Anatomy	5	7,743	15,439	33	26	278	253	52
Fly Anatomy	6	8,462	15,992	34	25	283	250	53
GO	1	30,745	33,441	48	38	518	214	71
GO	2	33,856	20,503	62	38	651	217	75
GO	3	31,395	15,752	67	38	639	236	73
GO	4	31,348	15,516	67	38	581	217	73
GO	5	31,419	18,721	63	39	713	222	76
GO	6	30,464	17,055	64	38	714	232	75
$\mathcal{EL}$-GALEN	1	21,319	211,839	9	76	403	1,582	20
$\mathcal{EL}$-GALEN	2	21,053	145,657	12	76	389	979	28
$\mathcal{EL}$-GALEN	3	21,230	129,322	14	76	394	922	30
$\mathcal{EL}$-GALEN	4	21,702	176,283	11	76	444	841	35
$\mathcal{EL}$-GALEN	5	21,996	157,872	12	80	385	867	31
$\mathcal{EL}$-GALEN	6	22,259	114,014	16	85	409	897	31

ing which is fine-tuned for small sets and supports very fast lookup and iteration (as, consequently, intersection) operations.

Optimized class taxonomy: ELK's implementation of taxonomy is specifically optimized for ontology class hierarchies, which are mostly shallow trees (or DAGs) with a possible large branching factor. For example, the bottom node which represents unsatisfiable concepts ($\bot$ and others) does not store references to its parent nodes (satisfiable concepts with no subsumees) and neither do its parents store a reference to $\bot$. Also, the taxonomy supports concurrent updates and can be built incrementally, i.e., new nodes can be added as soon as all subsumers for a given concept have been inferred.

Indexing: ELK does not explicitly store axioms after the ontology has been loaded. Instead, it creates instances of the rules in Figure 1 as it loads the axioms and stores them in the objects which represent entities occurring in the axiom. This is done to, first, avoid the cost of storing potentially a large number of complex axioms, second, enable a fast implementation of the $\mathsf{R}[\ldots]$ operator for finding applicable rules, and finally, group together different rules applicable to the same premises. For example, if $\mathcal{O}$ contains axioms $A \sqsubseteq B$, $A \sqsubseteq C$, and $A \sqsubseteq D$, they can be grouped into a threefold instance of $\mathbf{R}_{\sqsubseteq} : A \mapsto \{B, C, D\}$, which derives $X \sqsubseteq B$, $X \sqsubseteq C$, and $X \sqsubseteq D$ in one go when applying to $X \sqsubseteq A$ (for some concept X). In addition, such indexing ensures that if some construct, e.g., role composition axioms, never occurs in the ontology, then the corresponding rule, e.g., $\mathbf{R}_{\mathbf{o}}$, will never even be considered for selection. Finally, the rule instances can be quickly updated if some axioms are added or deleted without the need to reload the ontology from external memory.

5 Conclusion

ELK has not been designed for mobile platforms. However, it has been heavily engineered to minimize memory consumption and take advantage of multi-core CPUs. This paper provides some insight into what happens when such a reasoner is run on a mobile device. Our experimental results are preliminary but they suggest that well-engineered reasoners can provide acceptable performance on modern cell phones, and can even classify some of the largest available ontologies. It is worth mentioning that ELK does not depend on external libraries (other than for logging) and thus runs nearly out-of-the-box under the Android's JVM. Therefore, mobile users can immediately benefit from all improvements being made in the main ELK's development branch.

In the future it would make sense to perform a more detailed evaluation on multiple devices, including a comparison between existing reasoners (at least those which can work on multiple platforms out-of-the-box). Also, more fine-grained experiments and thorough profiling are required to understand the reasons why reasoning is that much slower on a cell phone. This may lead not only to faster mobile reasoning but also improve performance on desktops and servers.

References

1. Specht, G., Weithöner, T.: Context-aware processing of ontologies in mobile environments. In: Mobile Data Management Conference. (2006) 86–89

2. Kleemann, T.: Towards mobile reasoning. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
3. Motik, B., Horrocks, I., Kim, S.M.: Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In: World Wide Web Conference (Companion Volume). (2012) 63–72
4. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E., eds.: Proc. 10th Int. Semantic Web Conf. (ISWC'11). Volume 7032 of LNCS., Springer (2011) 305–320
5. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
6. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In Kaelbling, L., Saffiotti, A., eds.: Proc. 19th Int. Joint Conf. on Artificial Intelligence (IJCAI'05). (2005) 364–369