Approximate OWL-Reasoning with Screech*

Tuvshintur Tserendorj[1], Sebastian Rudolph[2], Markus Krötzsch[2], and Pascal Hitzler[2]

[1] FZI Karlsruhe, Germany
[2] AIFB, University of Karlsruhe, Germany

Abstract. Applications of expressive ontology reasoning for the Semantic Web require scalable algorithms for deducing implicit knowledge from explicitly given knowledge bases. Besides the development of more efficient such algorithms, awareness is rising that approximate reasoning solutions will be helpful and needed for certain application domains. In this paper, we present a comprehensive overview of the Screech approach to approximate reasoning with OWL ontologies, which is based on the KAON2 algorithms, facilitating a compilation of OWL DL TBoxes into Datalog, which is tractable in terms of data complexity. We present three different instantiations of the Screech approach, and report on experiments which show that a significant gain in efficiency can be achieved.

1 Introduction

Scalability of reasoning remains one of the major obstacles in leveraging the full power of the Web Ontology Language OWL [1] for practical applications. Indeed, large-scale applications normally use only a fragment of OWL which is very shallow in logical terms, and thus cannot employ the more sophisticated reasoning mechanisms for accessing knowledge which is implicit in knowledge bases. While the use of such shallow techniques already has added value, it would be preferable if the more complex logical constructors in the language could also be used. Consequently, scalability of OWL reasoning needs to be investigated on a broad front in order to advance the state of the art by several orders of magnitude.

Among the many possible approaches to address scalability, one of them concerns the use of logic programming for this purpose. This can be traced back to the work on Description Logic Programs (DLP) [2, 3], which are a naive Horn fragment of OWL DL. Along the same lines lies the OWL DL-fragment Horn-$\mathcal{SHIQ}$ [4, 5], which is based on the sophisticated transformation algorithms implemented in the KAON2-system[3] [5, 6]. Horn-$\mathcal{SHIQ}$ is strictly more expressive than DLP and allows, for example, the free use of existential role restrictions.

* Research reported in this paper was partially supported by the EU in the IST project NeOn (IST-2006-027595, `http://www.neon-project.org/`)) and by the Deutsche Forschungsgemeinschaft (DFG) under the ReaSem project.

[3] http://kaon2.semanticweb.org

At the same time, a different effort to leveraging Horn logic for OWL reasoning rests on the idea of approximate reasoning, which presupposes an application scenario where speed is so important that it becomes reasonable to allow some incorrect inferences in order to speed up the reasoning. The corresponding implementation is called Screech [7], and it is based on the idea of approximating an OWL DL knowledge base by Horn clauses. Initial experiments reported in [7] – and briefly in [8] – have shown that Screech indeed improves runtime in some cases, but further evaluations had been missing so far.

In this paper, we will introduce two new variants of the Screech approach (in Sections 2 and 3), resulting in three related algorithms, which can be used in combination for approximate OWL reasoning. We will then report on experiments (in Section 4) which we performed for all approaches. They show that all three variants of Screech indeed result in significant speed-up under only a very small number of introduced mistakes.

2 The Screech Approach

2.1 The KAON2-Transformation

Reasoning with KAON2 is based on special-purpose algorithms which have been designed for dealing with large ABoxes. They are detailed in [5] and we present a birds' eyes perspective here, which suffices for our purposes. The underlying rationale of the algorithms is that algorithms for deductive databases have proven to be efficient in dealing with large numbers of facts. The KAON2 approach utilises this by transforming OWL DL ontologies to disjunctive datalog, and by the subsequent application of the mentioned and established algorithms for dealing with disjunctive datalog.

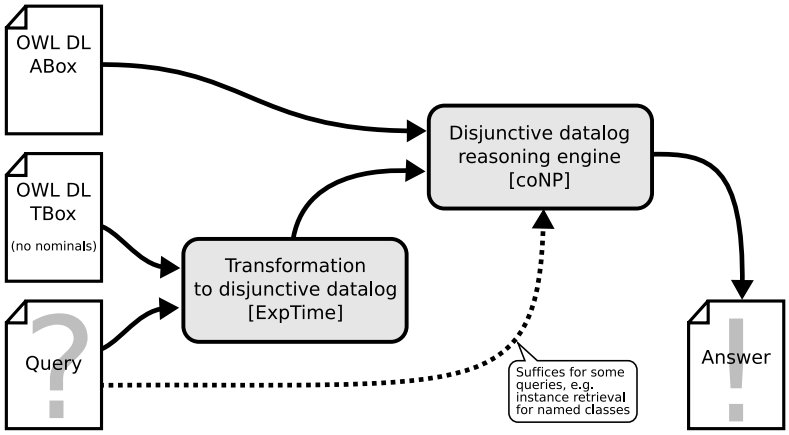

Fig. 1. KAON2 approach to reasoning

A birds' eyes perspective on the KAON2 approach is depicted in Figure 1. KAON2 can handle $\mathcal{SHIQ}(\mathbf{D})$ description logic ontologies, which corresponds roughly to OWL DL without nominals. The TBox, together with a query are processed by the sophisticated KAON2-transformation algorithm which returns a disjunctive datalog program. This, together with an ABox, is then fed into a disjunctive datalog reasoner which eventually returns an answer to the query.

In some cases, e.g. when querying for instances of named classes, the query does not need to be fed into the transformation algorithm but instead needs to be taken into account only by the datalog reaoner. This allows to compute the disjunctive datalog program offline, such that only the disjunctive datalog engine needs to be invoked for answering the query. All experiments we report on have been performed this way, i.e. they assume an offline transformation of the TBox prior to the experiments.

The program returned by the transformation algorithm is in general not logically equivalent to the input TBox. The exact relationship is given below in Theorem 1 due to [5]. Note that statement (b) suffices for our purposes. It also shows that the KAON2 datalog reasoning engine can in principle be replaced by other (sound and complete) reasoning engines without changing the results of the inference process.

Theorem 1. *Let K be a $\mathcal{SHIQ}(\mathbf{D})$ TBox and $D(K)$ be the datalog output of the KAON2 transformation algorithm on input K. Then the following claims hold.*

(a) K is unsatisfiable if and only if $D(K)$ is unsatisfiable.
(b) $K \models \alpha$ if and only if $D(K) \models \alpha$, where α is of the form $A(a)$ or $R(a, b)$, for A a named concept and R a role.
(c) $K \models C(a)$ for a nonatomic concept C if and only if, for Q a new atomic concept, $D(K \cup \{C \sqsubseteq Q\}) \models Q(a)$.

Convenient access to the KAON2 transformation algorithm is given by means of the KAON2 OWL Tool[4] dlpconvert,[5] which can also produce F-Logic [9] serialisations which can be used with F-Logic engines like OntoBroker.

2.2 Approximate OWL-Reasoning with Screech

Due to the inherent high complexity of reasoning with ontologies, it is to be expected that some application settings will defy even the smartest approaches for achieving sound and complete scalable algorithms. The method of choice for dealing with such situations is to use approximate reasoning, which trades correctness for time, but in a controlled and well-understood way.

The Screech approach is based on the fact that data complexity is polynomial for non-disjunctive datalog, while for OWL DL it is coNP complete even in the absence of nominals [4]. Screech utilises the KAON2 algorithms, but

[4] `http://owltools.ontoware.org/`
[5] `http://logic.aifb.uni-karlsruhe.de/wiki/Dlpconvert`

rather than doing (expensive) exact reasoning over the resulting disjunctive datalog knowledge base, it does approximate reasoning by treating disjunctive rules as if they were non-disjunctive ones, i.e. the disjunctive rules are approximated by Horn rules.

We will first describe the basic variant of SCREECH, which was introduced in [7], and which we call SCREECH-ALL. SCREECH-ALL is complete, but may be unsound in cases. Its data complexity is polynomial. Two other variants of SCREECH, SCHREECH-NONE and SCREECH-ONE, will be described in Section 3.

SCREECH-ALL uses a modified notion of *split program* [10] in order to deal with the disjunctive datalog. Given a rule

$$H_1 \vee \cdots \vee H_m \leftarrow A_1, \ldots, A_k,$$

as an output of the KAON2 transformation algorithm, the *derived split rules* are defined as:

$$H_1 \leftarrow A_1, \ldots, A_k \qquad \vdots \qquad H_m \leftarrow A_1, \ldots, A_k.$$

For a given disjunctive program P its *split program* P' is defined as the collection of all split rules derived from rules in P. It can be easily shown that for instance retrieval tasks, the result obtained by using the split program instead of the original one is complete but may be unsound. As the following proposition shows, this is even the case if all integrity constraints, i.e. rules of the form

$$\leftarrow B_1, \ldots, B_n$$

are removed from the split program.

Proposition 1. *Consider a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base KB that is logically consistent, let $\mathsf{DD}(KB)$ denote a disjunctive datalog program obtained by applying KAON2 to KB, and let P be the logic program obtained from $\mathsf{DD}(KB)$ by* SCREECH-ALL. *Then P has a least model which satisfies any atomic formula that is true in some minimal model of $\mathsf{DD}(KB)$.*

Especially, P entails all atomic formulae that are true in all (minimal) models of $\mathsf{DD}(KB)$, i.e. SCREECH-ALL *is complete for instance retrieval on consistent $\mathcal{SHIQ}(\mathbf{D})$ knowledge bases.*

Proof. First, note that we can restrict to propositional programs obtained as the (finite) ground instantiations of the relevant datalog programs. Hence it suffices to consider propositional models.

The fact that P has a least model is a standard conclusion from the fact that P is a definite logic program. To show the rest of the claim, consider any minimal model $\mathcal{M}$ of the ground instantiation of $\mathsf{DD}(KB)$ (note that KB has some model by consistency, and that some of those must be minial since only finitely many ground interpretations exist). Define a ground program $Q_{\mathcal{M}}$ as follows:

$$Q_{\mathcal{M}} = \{H_i \leftarrow B_1 \wedge \ldots \wedge B_m \mid \mathcal{M} \models B_1 \wedge \ldots \wedge B_m \text{ and } \mathcal{M} \models H_i, 1 \leq i \leq n)\}.$$

We claim that $Q_{\mathcal{M}}$ is a definite program with least model $\mathcal{M}$. Clearly $Q_{\mathcal{M}}$ is definite (thus has some least model), and has $\mathcal{M}$ as a model. But obviously any model of $Q_{\mathcal{M}}$ that is strictly smaller than $\mathcal{M}$ would also satisfy all rules of $\mathsf{DD}(KB)$, thus contradicting the assumed minimality of $\mathcal{M}$.

Now clearly $Q_{\mathcal{M}}$ is a subset of the screeched program P, and hence any model of P must be greater or equal to the least model $\mathcal{M}$ of $Q_{\mathcal{M}}$. Since $\mathcal{M}$ was arbitrary, this shows claim. □

It is possible to also deal with nominals, i.e. to deal with OWL DL (aka $\mathcal{SHOIN}(\mathbf{D})$) in an approximate manner. This was mentioned in [7], but for our purposes it will suffice to consider $\mathcal{SHIQ}$ knowledge bases only, which covers a significant portion of OWL DL.

Putting the pieces together, SCREECH-ALL utilises the following subsequent steps for approximate ABox reasoning for $\mathcal{SHIQ}$.

1. Apply transformations as in the KAON2 system in order to obtain a negation-free disjunctive datalog program.
2. Obtain the split program as described above.
3. Do reasoning with the split program, e.g. using the KAON2 datalog reasoning engine.

Given a TBox K, the split program obtained from K by steps 2 and 3 is called the *screeched version* of K. The first two steps can be considered to be preprocessing steps for setting up the intensional part of the database. ABox reasoning is then done in step 4. The resulting approach has the following theoretical features:

- It is complete with respect to OWL DL semantics.
- Data complexity is polynomial.

A prototype implementation of our approach is available as the SCREECH-ALL OWL approximate reasoner.[6] It is part of the KAON2 OWL Tools. KAON2[7] is the KArlsruhe ONtology framework, which includes a fast OWL reasoner based on the KAON2 transformation algorithms [5], and also includes many other features helpful for working with ontologies. We can convert a $\mathcal{SHIQ}$ ontology into a disjunctive datalog program, e.g. by using the KAON2 OWL Tool `dlpconvert` with the `-x` switch. SCREECH-ALL then accesses the results of the translation through the KAON2 API, creates the corresponding split programs and serializes them as Horn logic programs in Edinburgh Prolog syntax or in F-Logic [11, 12] syntax. We need to mention, however, that in general support for concrete domains and other features like integrity constraints is not necessarily implemented in off-the-shelf logic programming systems. In these cases, concrete domains etc. cannot be used. The KAON2 OWL Tool `ded`,[3] for example, performs a language weakening step by removing all concrete domains, and may come in handy in such situations.

[6] http://logic.aifb.uni-karlsruhe.de/screech
[7] http://kaon2.semanticweb.org

$$\begin{aligned}
\text{serbian} \sqcup \text{croatian} &\sqsubseteq \text{european} \\
\text{eucitizen} &\sqsubseteq \text{european} \\
\text{german} \sqcup \text{french} \sqcup \text{beneluxian} &\sqsubseteq \text{eucitizen} \\
\text{beneluxian} &\equiv \text{luxembourgian} \sqcup \text{dutch} \sqcup \text{belgian}
\end{aligned}$$

serbian(ljiljana)	serbian(nenad)	german(philipp)	french(julien)
chinese(yue)	german(peter)	german(stephan)	mongolian(tuvshintur)
indian(anupriya)	belgian(saartje)	german(raphael)	chinese(guilin)

Fig. 2. Example ontology

2.3 A Simple Example

We demonstrate the approach by means of a simple OWL DL ontology. It contains only a class hierarchy and an ABox, and no roles, but this will suffice to display the main issues.

The ontology is shown in Figure 2, and its intended meaning is self-explanatory. Note that the fourth line,

$$\text{beneluxian} \equiv \text{luxembourgian} \sqcup \text{dutch} \sqcup \text{belgian},$$

translates into the four clauses

$$\begin{aligned}
\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) &\leftarrow \text{beneluxian}(x), \qquad (1)\\
\text{beneluxian}(x) &\leftarrow \text{luxembourgian}(x),\\
\text{beneluxian}(x) &\leftarrow \text{dutch},\\
\text{and} \qquad \text{beneluxian}(x) &\leftarrow \text{belgian}(x).
\end{aligned}$$

Thus, our approach changes the ontology by treating the disjunctions in line (1) as conjunctions. Effectively, this means that the rule

$$\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) \leftarrow \text{beneluxian}(x)$$

is replaced by the three rules

$$\begin{aligned}
\text{luxembourgian}(x) &\leftarrow \text{beneluxian}(x),\\
\text{dutch}(x) &\leftarrow \text{beneluxian},\\
\text{and} \qquad \text{belgian}(x) &\leftarrow \text{beneluxian}(x).
\end{aligned}$$

This change affects the soundness of the reasoning procedure. However, most of the ABox consequences which can be derived by the approximation are still correct. Indeed, there are only two derivable facts which do not follow from the knowledge base by classical reasoning, namely

$$\text{dutch(saartje)} \qquad \text{and} \qquad \text{luxemburgian(saartje)}.$$

All other derivable facts are correct.

3 Variants of Screech

We will now introduce two other variants of SCREECH, besides SCREECH-ALL introduced above. These other variants are called SCREECH-NONE and SCREECH-ONE.

SCREECH-NONE is defined by simply removing all disjunctive rules (and all integrity constraints) after the transformation by the KAON2-algorithm. For the example from Section 2.3, this means that the rule

$$\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) \leftarrow \text{beneluxian}(x)$$

is simply deleted. The resulting reasoning procedure is sound, but incomplete, assuming that we start with a $\mathcal{SHIQ}(\mathbf{D})$ knowledge base.

SCREECH-ONE is defined by replacing each disjuntive rules by *exactly one* of the split rules. This selection can be done randomly, but will be most useful if the system has some knowledge – probably of statistical nature – on the size of the extensions of the named classes.[8] For our example from Section 2.3, when considering the rule

$$\text{luxembourgian}(x) \vee \text{dutch}(x) \vee \text{belgian}(x) \leftarrow \text{beneluxian}(x),$$

we can use the additional knowledge that there are more dutch people than belgians or luxenbourgians, thus this rule is replaced by the single rule

$$\text{dutch}(x) \leftarrow \text{beneluxian}(x).$$

We also remove all integrity constraints after the translation. The resulting reasoning proceedure is neither sound nor complete. We thus obtain the following result.

Proposition 2. *Instance retrieval with* SCHREECH-NONE *is sound but generally incomplete. Instance retrieval with* SCHREECH-ONE *in general is neither sound nor complete.*

Proof. Soundness of SCHREECH-NONE is immediate from the fact that calculations are performed on a subset of the computed clauses, together with monotonicity of the employed datalog variant. For all other claims it is easy to find counterexamples. □

The properties of SCREECH are summarised in Table 1.

From a theoretical point of view, it would be satisfying to characterize the described approximations in terms extremal bounds in certain logical fragments. However, we remark that the unsound screech variants do not yield greatest Horn lower bounds in the sense of [13] w.r.t. the disjunctive datalog program, not even

[8] This was suggested by Michael Sintek.

Table 1. SCREECH variants and their basic properties

variant	description	sound	complete
SCREECH-ALL	use *all* of the split rules	no	yes
SCREECH-NONE	use *none* of the split rules	yes	no
SCREECH-ONE	use *one* of the split rules	no	no

if we modify the definition to allow only definite Horn rules. As a counterexample for SCREECH-ALL, consider the program $\{\leftarrow C(a), C(a) \vee C(b) \leftarrow\}$. Its screeched version is $\{C(a) \leftarrow, C(b) \leftarrow\}$, but its greatest lower bound in the sense of [13] would be $\{C(b) \leftarrow\}$. Analogously, we note that SCREECH-ONE yields no greatest lower bound, even if integrity constraints are included (which obviously makes the procedure complete while still being unsound). To see this, consider the program $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), A(x) \vee B(x) \leftarrow C(x)\}$. Its (extended) SCREECH-ONE versions are $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), A(x) \leftarrow C(x)\}$ and $\{C(a) \leftarrow, C(b) \leftarrow, \leftarrow A(a), \leftarrow B(b), B(x) \leftarrow C(x)\}$, but its greatest lower bound would be $\{C(a) \leftarrow, C(b) \leftarrow, B(a) \leftarrow, A(b) \leftarrow\}$.

3.1 Expected results

Prior to performing our experiments – which we will report in Section 4 – we formulated the expected outcome from the different variants of SCREECH.

- SCREECH-ONE – assuming the mentioned knowledge about the size of the extensions of atomic classes – compared to SCREECH-ALL should show overall *less* errors for some suitable knowledge bases. We also expected SCREECH-ONE to be quicker than SCREECH-ALL.
- SCREECH-NONE should be quicker than SCREECH-ALL and SCREECH-ONE. We expected that the number of errors should be comparable with SCREECH-ALL, but more severe than SCREECH-ONE.

We furthermore expected, that the parallel execution of SCREECH-ALL and SCREECH-NONE should help to determine *exact* answers in some cases quicker than using the KAON2 datalog reasoner. This expectation is based on the following fact: If the extensions of some class C as computed by SCREECH-ALL and SCREECH-NONE are *of the same size*, then the computed extensions are actually correct (sound and complete) with respect to the original knowledge base.

4 Experimental Evaluation

An approximate reasoning procedure needs to be evaluated on real data from practical applications. Handcrafted examples are of only limited use as the applicability of approximate methods depends on the structure inherent in the experimental data.

So we evaluated some popular publicly available ontologies. In some cases we had to cautiously modify them in order to enable KAON2 to perform reasoning tasks on them, but the general approach was to first use KAON2 for transforming the TBoxes to disjunctive datalog. Also offline, a screeched version of the TBox was produced. We then invoked the KAON2 disjunctive datalog engine on both the resulting disjunctive datalog program and on the screeched version, to obtain a comparison of performance.[9]

For all our experiments, we used a T60p IBM Thinkpad with 1.9GB of RAM, with the Java 2 Runtime Environment, Standard Edition (build 1.5.0_09-b03).

Results in a nutshell

We performed comprehensive experiments with GALEN, WINE, DOLCE, and SEMINTEC. Before we report in more detail, we list a summary of the results.

- SCREECH-ALL shows an average speedup in the range between 8 and 67%, depending on the ontology under consideration, while 39 to 100% of the computed answers are correct. Most interestingly, a higher speedup usually seemed to correlate with less errors.
- SCREECH-ONE compared to SCREECH-ALL shows overall *less* errors. In most cases, all correct class members are retrieved. Its runtime is comparable to SCREECH-ALL.
- SCREECH-NONE compared to SCREECH-ALL shows similar run-time. In most cases, the extensions are computed *correctly* – with the exception of WINE, for which we get 2% missing answers.
- Running SCREECH-ALL and SCREECH-NONE in parallel and comparing the results, allows the following: If the computed extensions are of the same size, then we know that all (and only correct) class members have been found. This is the case for more than 76% of all classes we computed.

GALEN

We first report on our experiments with the OWL DL version of the GALEN Upper Ontology.[10] As it is a TBox ontology only, we populated GALEN's 175 classes randomly with 500 individuals.[11] GALEN does not contain nominals or concrete domains. GALEN has 673 axioms (the population added another 500). After the TBox translation to disjunctive datalog we obtained ca. 1800 disjunctive datalog rules,[12] ca. 60 of which contained disjunctions.[13] The SCREECH-ALL

[9] The raw data of the experiments can be found online under `http://logic.aifb.uni-karlsruhe.de/wiki/Screech`.

[10] http://www.cs.man.ac.uk/∼rector/ontologies/simple-top-bio/

[11] Using the `pop` KAON2 OWL tool.

[12] The exact numbers differ slightly on different runs, as the KAON2 translation algorithm is non-deterministic. Here it was between 1737 and 1909.

[13] The number of disjunctive rules ranged between 51 and 81 on different runs.

Table 2. Summary of the three SCREECH versions on GALEN. *miss* indicates the elements of the extensions which were *not* found by the approximation, *corr* indicates the elements which were correctly found, and *more* indicates elements which were incorrectly computed to be part of the extension. *time* gives the runtime (in ms) for the respective SCREECH version, while *KAON2* gives the runtime (in ms) using the disjunctive rules. *f-meas* is the f-measure known from information retrieval, computed as $(2 \cdot \text{precision} \cdot \text{recall})/(\text{precision} + \text{recall})$ with precision = corr/(corr + more) and recall = corr/number of actual instances, *corr.class* gives the fraction of classes for which the extension was computed correctly, and *time/KAON2* is the ratio between time and KAON2.

Variant	miss	corr	more	time	KAON2	f-meas	corr.class	time/KAON2
SCREECH-ALL	0	5187	465	513546	1562898	0.957	0.78	0.33
SCREECH-ONE	5	5182	134	569658	1562898	0.987	0.98	0.36
SCREECH-NONE	10	5177	0	366711	1562898	0.999	0.78	0.23

split resulted in 113 new rules, replacing the disjunctive ones. 149 integrity constraints were also removed.

We then queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits.

A summary of the results can be seen in Table 2. For 137 of the 175 classes (i.e. 78%), the computed extensions under SCREECH-ALL and SCREECH-NONE had the same number of elements, which allows to conclude – without using the disjunctive rules – that for those classes the extensions were computed correctly. For some classes, so for the class `Physical-occurrent-entity`, computing the extension under SCREECH-ALL saved 99% of the runtime.

While the different versions of SCREECH have about the same runtime, the differences in the number of introduced errors is remarkable. Indeed, SCREECH-NONE makes almost no mistakes. The parallel execution of SCREECH-NONE and SCREECH-ALL, as mentioned, allows to compute the correct extensions of 78% of the classes – and to know that the computations are correct – in less than a quarter of the time needed by using the unmodified knowledge base.

DOLCE

DOLCE[14] (a Descriptive Ontology for Linguistic and Cognitive Engineering) is a foundational ontology, developed by the Laboratory for Applied Ontology in the Institute of Cognitive Sciences and Technology of the Italian National Research Council. In full, it exceeds the reasoning capabilities of current reasoners, hence we used a fraction for our experiments consisting of 1552 axioms. Since DOLCE is a pure TBox-Ontology, we randomly populated it with 494 individuals to be able to carry out instance retrieval.

[14] `http://www.loa-cnr.it/DOLCE.html`

Table 3. Summary of the three SCREECH versions on DOLCE. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
SCREECH-ALL	0	3697	2256	472941	516064	0.766	0.76	0.92
SCREECH-ONE	0	3697	512	425748	516064	0.935	1.0	0.82
SCREECH-NONE	0	3697	0	397260	516064	0.0	1.0	0.77

The conversion into disjunctive datalog yielded ca. 1780 rules[15] of which ca. 72 are disjunctive.[16] The SCREECH-ALL split resulted in 176 new rules, replacing the disjunctive ones. We also removed ca. 200 integrity constraints.[17]

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits.

A summary of the results can be seen in Table 3. For 93 of the 123 classes (i.e. 76%), the computed extensions under SCREECH-ALL and SCREECH-NONE had the same number of elements, which allows to conclude – without using the disjunctive rules – that for those classes the extensions were computed correctly.

Remarkable under DOLCE is that SCREECH-NONE makes no mistakes, while the runtime improvement is rather mild. This indicates that the disjunctive knowledge in DOLCE does not contribute any results.

WINE

The next ontology we tested was the WINE ontology.[18] It is a well-known ontology containing a classification of wines. Moreover, it is one of the rare ontologies with both an ABox and a nontrivial TBox. It also contains nominals, which we removed in an approximate way following [7].[19] The resulting ontology contains 20762 axioms, including functionality, disjunctions, and existential quantifiers. The corresponding ABox contains 6601 axioms.

The translation procedure into disjunctive datalog produces altogether ca. 550 rules,[20] among them 24 disjunctive ones. The SCREECH-ALL split resulted in 48 new rules, replacing the disjunctive ones. We also removed 3 integrity constraints after the translation.

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits.

[15] 1775–1788
[16] 72–73
[17] 188–190
[18] `http://www.schemaweb.info/schema/SchemaDetails.aspx?id=62`
[19] We used the TBox *after* that processing as baseline, since we are interested in the comparion of the different versions of SCREECH.
[20] 526–572

Table 4. Summary of the three SCREECH versions on WINE. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
SCREECH-ALL	0	30627	1353	588562	707476	0.978	0.93	0.83
SCREECH-ONE	0	30627	615	494456	707476	0.990	0.94	0.70
SCREECH-NONE	697	29930	0	504914	707476	0.988	0.90	0.71

A summary of the results can be seen in Table 4. For 116 of the 140 classes (83%), the computed extensions under SCREECH-ALL and SCREECH-NONE had the same number of elements, which allows to conclude – without using the disjunctive rules – that for those classes the extensions were computed correctly.

WINE is the only ontology we tested for which SCREECH-NONE resulted in mildly significant number of mistakes. However, precision is still at 0.977, i.e. very good. Considering the fact that WINE was created to show the expressiveness of OWL DL, it is remarkable that all three SCREECH versions show a very low amount of errors, while runtime increases by 17–30%. For some classes – e.g. for `Chianti`, over 91% of the runtime were saved using SCREECH-ALL.

SEMINTEC

In a last investigation, we consider an ontology, the translation of which turned out to not contain proper disjunctive rules. Nevertheless, removing integrity constraints is supposed to result in improving runtime behaviour (while in this case even preserving soundness).

So, the last ontology we considered was created in the SEMINTEC project[21] at the university of Poznan and is concerned with financial services. Its TBox contains 130702 axioms of comparably simple structure, apart from some functionality constraints which require equality reasoning. The ABox contains 35882 axioms.

The TBox translation generated 217 rules, all of them being Horn, among which were 113 integrity constraints.

As before, we queried all named classes for their extensions using the KAON2 datalog engine, both for processing the disjunctive datalog program and for the various splits.

Table 5. Summary of SCREECH on SEMINTEC – note that all three versions of SCREECH coincide, since no disjuntive rules are produced by the translation. For the legend, see Table 2.

Variant	miss	corr	more	time	KAON2	f-meas.	corr.class	time/KAON2
SCREECH	0	51184	0	41796	86045	1.0	1.0	0.49

[21] `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

Table 6. Overview of Screech evaluations. Mark that for due to the completeness of Screech-all, the recall values are always 100% as well as the precision values for Screech-none due to its soundness. Moreover, the three Screech variants coincide in the case of the SEMINTEC ontology.

	Screech-all		Screech-one			Screech-none	
ontology	time saved	precision	time saved	precision	recall	time saved	recall
GALEN	38.0%	91.8%	63.6%	97.5%	99.9%	76.5%	99.8%
DOLCE	29.1%	62.1%	17.5%	87.8%	100%	23.0%	100%
WINE	34.5%	95.8%	30.1%	98.0%	100%	28.6%	97.7%
SEMINTEC	67.3%	100%	67.3%	100%	100%	67.3%	100%

A summary of the results can be seen in Table 5. As in the case of absence of disjunctive rules all three variants of Screech coincide, for all of the 59 classes, the extensions were computed correctly.

For SEMINTEC, we achieve a performance improvement of 54% while the computation remains correct. For some classes - in particular for some with very small extensions, computing the extension under Screech-all saved about 95% of the runtime. For some classes with larger extension – like `Leasing`, 92% of runtime was saved.

5 Conclusions

Motivated by the obvious need for techniques enhancing the scalability of reasoning related tasks, we have investigated three variants of the Screech approach to approximate reasoning in OWL ontologies.

On the theoretical side, we gave the completeness result for Screech-all and the soundness result for Screech-none, yet a desirable characterisation of the approximations in terms of extremal bounds following the theory of Horn-approximations was shown not to hold by providing counterexamples.

However, on the practical side the obtained results were promising: the performance improvement is stable over all ontologies which we included in our experiments. The performance gain varied between 17.5 and 76.5%, while the amount of correctly retrieved classes was above 87.8% for all but one of the ontologies – see Table 6. It is encouraging to see that the approach appears to be feasible even for the sophisticated WINE ontology, and also for the SEMINTEC ontology, although in the latter case we only remove integrity constraints.

Concerning the comparatively bad results on DOLCE, we note that the results are quite counterintuitive. One would naively expect that performance improvements go hand-in-hand with loss of precision. However, for DOLCE we measured both the least runtime improvement and the worst performance in terms of correctness. Concerning correctness, we suspect that the comparatively large number of incorrect answers is caused by the fact that DOLCE uses a large number of complete partitions of the form $A \equiv A_1 \sqcup \cdots \sqcup A_n$, where all the A_i are also specified to be mutually disjoint. It is intuitively clear that this

kind of axioms introduces disjunctive (non-Horn-style and therefore harder to approximate) information on the one hand and integrity constraints (those being neglected in our approximation) on the other. However, this does not in itself explain why we did not observe a higher speedup. This indicates that the properties of ontologies which lead to performance improvement through screeching must be less straightforward than initially expected. For a clarification, more evaluations taking into account a wider range of ontologies with differing characteristics w.r.t. expressivity, used language features, or statistical measures like degree of population will lead to substantial hypotheses.

In general, we see a great potential in the strategy to combine various (possibly approximate) algorithms having known properties as soundness and/or completeness maybe with respect to differing types of queries. For instance, the proposed "sandwich technique" can be used to solve instance retrieval tasks in some cases even without calling the more costly sound and complete reasoners. If the sets of individuals I_S and I_C retrieved by two algorithms—one of those being sound and the other one complete—coincide, the result is known to be exact. But even if not, the result is at least partly determined (as all elements of I_S are definitely instances and all individuals *not* in I_C are not) and it might be beneficial to query a sound and complete reasoner for class membership only for individuals of the set $I_C \setminus I_S$ of individuals for which class membership is still undecided. Clearly, the strategy to combine several approximate algorithms will be especially reasonable if parallel computation architectures are available.

References

1. McGuinness, D.L., van Harmelen, F.: OWL web ontology language overview (2004) `http://www.w3.org/TR/owl-features/`.
2. Grosof, B., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logics. In: Proc. of WWW 2003, Budapest, Hungary, May 2003, ACM (2003) 48–57
3. Volz, R.: Web Ontology Reasoning with Logic Databases. PhD thesis, Institute AIFB, University of Karlsruhe (2004)
4. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In Kaelbling, L.P., Saffiotti, A., eds.: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland. (2005) 466–471
5. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Universität Karlsruhe (2006)
6. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic ABoxes. In: Proc. of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR 2006), Phnom Penh, Cambodia (2006)
7. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In Gil, Y., et al., eds.: Proceedings of the 4th International Semantic Web Conference, Galway, Ireland, November 2005. Volume 3729 of Lecture Notes in Computer Science., Springer, Berlin (2005) 383–397

8. Rudolph, S., Krötzsch, M., Hitzler, P., Sintek, M., Vrandecic, D.: Efficient OWL reasoning with logic programs – evaluations. In Marchiori, M., Pan, J.Z., de Sainte Marie, C., eds.: Proceedings of The First International Conference on Web Reasoning and Rule Systems 2007 (RR2007). Volume 4524 of Springer Lecture Notes in Computer Science., Springer (2007) 370–373
9. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995) 741–843
10. Sakama, C., Inoue, K.: An alternative approach to the semantics of disjunctive logic programs and deductive databases. Journal of Automated Reasoning **13** (1994) 145–172
11. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the ACM **42** (1995) 741–843
12. Angele, J., Lausen, G.: Ontologies in F-logic. In Staab, S., Studer, R., eds.: Handbook on Ontologies. Springer (2004) 29–50
13. Selman, B., Kautz, H.A.: Knowledge compilation using horn approximations. In: Proc. 9th National Conference on Artificial Intelligence (AAAI-91). (1991) 904–909