

Next Steps for ReAD: Modules for Classification Optimisation

Haoruo Zhao¹, Bijan Parsia¹ and Uli Sattler¹

¹The University of Manchester, Oxford Rd, Manchester, UK M13 9PL

Abstract


Ontology Classification is a central DL reasoning task and supported by several highly-optimised reasoners for OWL ontologies. Different notions of modularity, including the atomic decomposition (AD), have already been exploited by different *modular reasoners*. In our previous work, we have designed and implemented a new AD-informed and MORE-inspired algorithm that uses Hermit and ELK as delegate reasoners, but avoids any duplicate subsumption tests between these two reasoners. In this paper, we push the algorithm further with easyfication (checking subsumption tests in a module rather than in the whole ontology) and parallelization. We empirically evaluate our algorithm with a set of SNOMED CT extensions ontologies and a corpus of BioPortal ontologies. We also design, implement and empirically evaluate a new modular reasoner, called Crane, which works with “coarsened” AD.


Keywords

Classification, Delegate Reasoner, Modular Reasoning


1. Introduction


With the development of syntactic locality-based modules [1, 2] there was a surge of interest in using them to optimise classification time of description logic (DL) [3, 4] TBoxes [5, 6, 7, 8, 9, 10]. They, especially in their so-called \perp form, have a lot of properties which seem attractive for reasoner optimisation: They are subsumer complete (i.e., a \perp -module entails all subsumers for its signature), can be extracted in polynomial time, and are subsets of the original ontology. These features make “black-box” approaches very straightforward: first decompose the ontology into some covering set of \perp -modules, classify each \perp -module as if it were an independent ontology, then union the results. If you have dispatch criteria, one can build a coalition reasoner by adding a dispatching function to several independent reasoners. The MORE [5] family of reasoners took the latter approach, breaking their input into two modules¹: an \mathcal{EL}^{++} -only [11, 12] module which was delegated to an \mathcal{EL}^{++} -reasoner [13, 14] and the “remainder” module which was delegated to a *SROIQ* [15] (or similar) reasoner [16, 17, 18]. Chainsaw [6], contrariwise, only used one reasoner but broke the input into the smallest possible module.

 DL 2022: 35th International Workshop on Description Logics, August 7–10, 2022, Haifa, Israel

 haoruo.zhao@manchester.ac.uk (H. Zhao); bijan.parsia@manchester.ac.uk (B. Parsia);

uli.sattler@manchester.ac.uk (U. Sattler)

 <http://www.cs.man.ac.uk/~bparsia/> (B. Parsia); <http://www.cs.man.ac.uk/~sattler/> (U. Sattler)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

¹There are variants which have a more complex approach to the remainder, but the most prominent releases versions are close to this description.

Both approaches had successes but, even excluding decomposition time, they were not robust. While for certain ontologies they showed worthwhile improvement, sometimes they make things significantly worse. One hypothesis is that locality-based modules tend to overlap a lot thus potentially inducing repeated work. The *only* information communicated between classified modules is their final class hierarchy. To take an extreme example, if the \mathcal{EL}^{++} module and the remainder module have 90% overlap, the *SRIOIQ*-reasoner might take nearly as long on that as it would on the whole ontology and it would still have the cost of the \mathcal{EL}^{++} -reasoner on top. If the whole ontology took infeasibly long for the *SRIOIQ*-reasoner to classify, then the modularisation was pointless.

In [19], different strategies for “chunking” a set of Chainsaw like modules were explored, attempting to mitigate the duplicated work problem. However, none of the strategies proved very robust or provide reliable improvements over a whole ontology approach.

Alternatively, one can abandon the black-box approach and modify a reasoner to communicate partial results between classifications of different modules. In [10], we demonstrated a prototype of a MORE style reasoner, ReAD, that modified HerMiT’s traversal algorithm [20, 21] both to avoid repeating work that the \mathcal{EL}^{++} reasoner has already done and also to use information from a modular structure over the ontology (that is, the Atomic Decomposition [22]) to avoid fruitless subsumption tests.

In this paper, we report on several new variants of the basic light-touch glass-box approach. In particular, we apply a (glass-box) Chainsaw like algorithm to the remainder module with and without parallelisation both for the most fine grained modules (Granular ReAD) possible and for a slightly coarsened set of modules (Crane). Somewhat surprisingly, we show that Crane, excluding decomposition time, shows near dominance over all our variants and both HerMiT and a HerMiT-MORE style system, often dramatically so. We also show that Crane is comparable with Konclude even including decomposition time on versions of SNOMED CT that has been enriched with disjunctive axioms.

These results suggest that glass-box approaches can realise the promise of modular reasoning.

2. Preliminaries

We assume people are familiar with *Description Logic* [3] and basic inference services such as consistency checking and classification.

In this paper, we use \mathcal{O} for an ontology, \mathbf{N}_C for a set of concept names and \mathbf{N}_R for a set of role names. A *signature* is a set $\Sigma \subseteq \mathbf{N}_C \cup \mathbf{N}_R$ of *terms*. For an axiom, a concept expression or an ontology X , we use \tilde{X} to denote its signature, i.e. the set of concept and role names. We use $C(\mathcal{O})$ for the set of concept names in \mathcal{O} , and $C^+(\mathcal{O}) = C(\mathcal{O}) \cup \{\perp, \top\}$ for concept names in \mathcal{O} together with \top, \perp .

Modules based on *syntactic* or semantic *locality* [1] have been introduced as logically well-behaved, tractable approximations of *Conservative Extension (CE)* modules [23, 24, 25]. These modules are subsets of the ontology and we focus here on syntactic \perp -locality in this paper (we use module to represent that). These modules are computable polynomial time, and satisfy a range of other useful properties [2, 26], in particular

1. preserve all entailments of \mathcal{O} over Σ ,

2. preserve all entailments of \mathcal{O} over $\widetilde{\mathcal{M}}$,
3. are unique, i.e., there is no other \perp -locality based module of \mathcal{O} for Σ ,
4. are subsumer-preserving, i.e., for concept names $A \in \widetilde{\mathcal{M}}, B \in \widetilde{\mathcal{O}}, \mathcal{O} \models A \sqsubseteq B$ implies $\mathcal{M} \models A \sqsubseteq B$ for \perp -locality based module, and
5. are monotonic, i.e., if $\Sigma_1 \subseteq \Sigma_2$, then $\mathcal{M}(\Sigma_1, \mathcal{O}) \subseteq \mathcal{M}(\Sigma_2, \mathcal{O})$.

The *atomic decomposition (AD)* $\mathfrak{A}(\mathcal{O})$ [27, 22] partitions an ontology into logically inseparable sets of axioms, so-called *atoms* \mathfrak{a} , and relates these atoms via a *dependency relation* \succeq . The principal ideal of \mathfrak{a} , defined as $\downarrow \mathfrak{a} = \{\alpha \in \mathfrak{b} \mid \mathfrak{a} \succeq \mathfrak{b}\}$, is a *genuine module* [27, 22].

2.1. An AD Derived Todo List

In [10], we explored the foundations of using the AD for avoiding STs during classification. Using the AD, we identify a (hopefully small) set of subsumption tests (STs) $\text{Subs}(\mathcal{O})$ that are necessary and sufficient for classification of \mathcal{O} . All other tests are unnecessary because they are known not to hold since their signatures are never subsets of a common genuine \perp -module signature. Consider the following sets:

$$\begin{aligned}
\text{Ats}(A) &:= \{\mathfrak{a} \in \mathfrak{A}(\mathcal{O}) \mid A \in \widetilde{\mathfrak{a}}\} && \text{the atoms of } A \\
\text{MinAts}(A) &:= \{\mathfrak{a} \in \text{Ats}(A) \mid \nexists \mathfrak{b} \in \text{Ats}(A) \text{ with } \mathfrak{a} \succ \mathfrak{b}\} && \text{the lowest atoms of } A \\
\text{CanS}(\mathfrak{a}) &:= \{A \mid \mathfrak{a} \in \text{MinAts}(A) \text{ and } \#\text{MinAts}(A) = 1\} && \text{an atom's candidate set} \\
\text{BTop}(\mathcal{O}) &:= \{A \mid A \in \widetilde{\mathcal{O}} \cap \mathbf{N}_C \text{ and } \#\text{MinAts}(A) > 1\} && \text{concept names below } \top
\end{aligned}$$

The *candidate set* $\text{CanS}(\mathfrak{a})$ of an atom \mathfrak{a} are those concept names for which STs need to be run for \mathfrak{a} , and concepts in $\text{BTop}(\mathcal{O})$ have only trivial subsumers.

Definition 1. [10] *The set of STs $\text{Subs}(\mathfrak{a})$ of an atom \mathfrak{a} is defined as follows:*

$$\begin{aligned}
\text{Subs}(\mathfrak{a}) &:= \{(A, B) \mid A \in \text{CanS}(\mathfrak{a}), B \in \downarrow \mathfrak{a}, \text{ and } A \neq B\} \cup \\
&\quad \{(A, \perp) \mid A \in \text{CanS}(\mathfrak{a})\} \cup \\
&\quad \{(\top, B) \mid B \in \text{CanS}(\mathfrak{a})\}.
\end{aligned}$$

$$\text{Subs}(\mathcal{O}) := \bigcup_{\mathfrak{a} \in \mathfrak{A}(\mathcal{O})} \text{Subs}(\mathfrak{a}) \cup \{(\top, A) \mid A \in \text{BTop}(\mathcal{O})\} \cup \{(\top, \perp)\}.$$

As a consequence of the following theorem, a reasoner that tests only STs in $\text{Subs}(\mathcal{O})$ during classification will (a) test all required, non-trivial² STs and (b) never duplicate a test.³

Theorem 1. [10] *For $A, B \in \widetilde{\mathcal{O}} \cap \mathbf{N}_C \cup \{\top, \perp\}$ with $\perp \neq A \neq B \neq \top$. If $\mathcal{O} \models A \sqsubseteq B$ then $(A, B) \in \text{Subs}(\mathcal{O})$, and (A, B) is either in exactly one $\text{Subs}(\mathfrak{a})$ or of the form (\top, A) or (\top, \perp) .*

²Of course we avoid testing tautologies.

³It may, though, include a test (A, C) in addition to (A, B) and (B, C) .

2.2. Enhanced KP Algorithm, MORE and ReAD

HermiT uses the *Enhanced KP (EKPA) classification algorithm* [28, 21]. It builds a completion graph with two sets of concepts pairs: One is *known (K) subsumer pairs* which records positive subsumption relations. The other one is *remaining possible (P) subsumer pairs* which contains remaining STs. The whole algorithm aims to empty P and increase K. HermiT aggressive strives to fill K whenever it performs a satisfiability check, whether directly on a concept name or as part of ST. For example, if the structures HermiT builds to test whether $A \sqsubseteq B$ allow it to determine that $E \sqsubseteq F$ it will add $E \sqsubseteq F$ to K and remove it from P.

MORE [5] is a black-box modular reasoning approach and splits the ontology into two modules of different expressivity, and then uses a fast, *delegate* reasoner on the inexpressive module. In its empirical evaluation, MORE uses ELK [14] to classify the module in \mathcal{EL}^{++} [12, 11] and HermiT [18] for the remaining OWL 2 module.

ReAD [10] is a glass-box and MORE-inspired approach. Compared to MORE, ReAD uses ELK to classify a union of all \mathcal{EL}^{++} modules,⁴ called \mathcal{T}_{EL} , by exploiting AD and uses a modified HermiT to classify a union of remaining modules. called \mathcal{T}_{RAS} . The EKPA in HermiT is modified to ensure 1) we do not check $\text{Subs}(\alpha)$ if $\alpha \subseteq \mathcal{T}_{EL}$; 2) we remove pairs (A, B) from P if there is no $\alpha \subseteq \mathcal{T}_{RAS}$ with $(A, B) \in \text{Subs}(\alpha)$.

3. Chainsawing the Remainder Module

ReAD avoids repeating work done by the EL reasoner and exploits some information from the AD in HermiT. But HermiT still remains a significant bottleneck. Thus, the natural next step is to try to break down the remainder module into smaller, we hope in aggregate, easier modules.

3.1. Granular ReAD (ReAD_G)

Since we pre-compute an AD, we can use it to reason over a covering set of minimal modules of \mathcal{T}_{RAS} . Rather than checking $\mathcal{T}_{RAS} \models^? A \sqsubseteq B$ with $(A, B) \in \text{Subs}(\alpha)$ and $\alpha \subseteq \mathcal{T}_{RAS}$, we check $\downarrow \alpha \models^? A \sqsubseteq B$ with $(A, B) \in \text{Subs}(\alpha)$. For every genuine module $\downarrow \alpha$ with $\alpha \subseteq \mathcal{T}_{RAS}$, we initialize a modified HermiT which only checks $\text{Subs}(\alpha)$ and classifier $\downarrow \alpha$. One benefit for that is that we can parallelize this procedure, called ReAD_{GP}. Algorithm 1 shows how ReAD_G works. Compared to ReAD's algorithm in [10], the modification is in lines 12-14. The parallelization is also built for lines 12-14. Compared to ReAD's algorithm, we modify less in EKPA algorithm. We only ensure we check (A, B) if $A \in \text{CanS}(\alpha)$ but we do not remove any pairs (A, B) from P.

Since ontologies tend to have lots of atoms, there is a risk that the overhead for “reasoner swap” (building the reasoner, initializing, and deleting the reasoner instances) will be significant. There are also three potential limitations for this approach due to interactions with other optimizations for classification. First, HermiT uses a Hypertableau algorithm as its ST engine and may get some “free” inferred subsumption relations from each ST.

⁴The differences in deriving the EL and Remainder modules in the direct MORE or via a union of appropriate modules does not seem significant for classification. In our case, we have the AD so it is convenient to use the union technique.

Algorithm 1 ReAD_G

Require: an ontology \mathcal{O}

- 1: Initialize a hierarchy $\mathcal{H} := \{(\perp, \top)\}$
 - 2: use MORE approach to compute a (potentially large) \mathcal{EL}^{++} module \mathcal{M}_{EL} and a remaining module $\mathcal{M}_{\text{RAS}} = \mathcal{M}(\text{C}^+(\mathcal{O}) \setminus \widetilde{\mathcal{M}}_{\text{EL}}, \mathcal{O})$
 - 3: Compute the \perp - $\mathcal{A}\mathcal{M}_{\text{RAS}}$
 - 4: $\text{ELAtoms} := \{\mathbf{a} \in \perp\text{-}\mathcal{A}(\mathcal{O}) \mid \downarrow \mathbf{a} \text{ is in } \mathcal{EL}^{++}\}$ {Find all \mathcal{EL}^{++} modules}
 - 5: $\mathcal{T}_{\text{EL}} := \mathcal{M}_{\text{EL}} \cup_{\mathbf{a} \in \text{ELAtoms}} \mathbf{a}$ {Compute union of \mathcal{EL}^{++} modules}
 - 6: $\mathcal{H} := \text{Classify}(\mathcal{T}_{\text{EL}})$ {use ELK for this}
 - 7: **for** atom $\mathbf{a} \in \text{RemainingAtoms}$ {build a Hermit for $\downarrow \mathbf{a}$ } **do**
 - 8: $\mathcal{H} := \mathcal{H} \cup \text{Classify} \downarrow \mathbf{a}$ {use modified Hermit only checks $\text{Subs}(\mathbf{a})$ }
 - 9: **end for**
 - 10: **return** \mathcal{H} {If a call to Classify found inconsistency, we would have exited early}
-

Some optimizations for STs like pseudo-model merging are not shared between reasoner instantiation.

Finally, ET-based algorithms avoid STs by exploring the transitivity of the (inferred) subsumption relation but these “transitive shortcuts” are included in $\text{Subs}(\mathcal{O})$. Hence we may lose this benefit of ET for these inferred subsumption relation.

Going more glass-box (e.g., caching completion graphs or pseudomodels) might help in some cases, but it may be that smaller is not always better. More fundamentally, computing the AD can be very expensive, often dwarfing classification time. While work is ongoing to optimise AD computation, for any AD approach to be reasonable, we have to assume that the AD is computed offline and perhaps maintained through updates. While this is not particularly challenging, it is a change to the basic infrastructure.

3.2. “Coarsened” AD and Crane

Motivated by these issues, we have experimented with using a “coarser” version of the AD. In [22], we find ADs are often very fine-grained and computing AD is, in particular, very costly despite it being in PTIME. In our new system, Crane, we build “coarsened” AD as an approximation of the AD and use this for AD-based traversal and classification. Furthermore, we compute this coarsened AD on the fly interleaved with classification call. This could be changed to a pre-computation strategy.

The general idea is to get coarsened AD and thus larger atoms and modules and classify them. When we classify a coarsened module, we check the subsumers of the concept names in this coarsened module. We have a global set `ClassifiedC` of concept names to record the concept names which are already checked for their subsumers.

We first compute the signature Σ_{EL} from the MORE approach and use it to compute an \mathcal{EL}^{++} module \mathcal{M}_1 and a remaining module \mathcal{M}_2 . We use ELK to classify \mathcal{M}_1 . Now we already checked all subsumers of concept names in \mathcal{M}_1 . We add these concept names into `ClassifiedC` in line 4. In Algorithm 2, we make this genuine module coarsened so we get the module w.r.t the signature of several axioms. Here picking how many axioms to get the coarsened module is

crucial and a parameter to Crane. In our experiments, we have set this parameter to $n = 100$. Then we can also further decompose these modules. Here we choose to further decompose these modules once. So we use the threshold $\frac{n}{10} = 10$ to extract smaller modules and then classify them.

Algorithm 2 Crane

Require: an ontology \mathcal{O} , a parameter n

- 1: compute Σ_{EL} {use the code from MORE for this}
- 2: compute $\mathcal{M}_1 = \mathcal{M}(\Sigma_{\text{EL}}, \mathcal{O})$ and $\mathcal{M}_2 = \mathcal{M}(\tilde{\mathcal{O}} \setminus \Sigma_{\text{EL}}, \mathcal{O})$
- 3: $\mathcal{H} := \mathcal{H} \cup \text{Classify}(\mathcal{M}_1)$ {use ELK for this}
- 4: $\text{ClassifiedC} := \mathcal{M}_1 \cap \text{C}^+(\mathcal{O})$
- 5: $\text{UnclassifiedAxioms} := \mathcal{O} \setminus \mathcal{M}_1$
- 6: **while** $\text{UnclassifiedAxioms} \neq \emptyset$ **do**
- 7: get n axioms \mathcal{T} from $\text{UnclassifiedAxioms}$
- 8: compute $\mathcal{M}_{\text{COAR}} = \mathcal{M}(\tilde{\mathcal{T}}, \mathcal{M}_2)$
- 9: $\text{UnclassifiedModuleAxioms} := \mathcal{M}_{\text{COAR}} \cap \text{UnclassifiedAxioms}$
- 10: **while** $\text{UnclassifiedModuleAxioms} \neq \emptyset$ **do**
- 11: get $\frac{n}{10}$ axioms \mathcal{T}_1 from $\text{UnclassifiedModuleAxioms}$
- 12: $\mathcal{M}_{\text{COARFurther}} := \mathcal{M}(\tilde{\mathcal{T}}_1, \mathcal{M}_{\text{COAR}})$
- 13: $\text{CanS} := (\mathcal{M}_{\text{COARFurther}} \cap \text{C}^+(\mathcal{O})) \setminus \text{ClassifiedC}$
- 14: $\mathcal{H} := \mathcal{H} \cup \text{Classify } \mathcal{M}_{\text{COARFurther}}$ with CanS {use modified Hermit which only checks (A, B) with $A \in \text{CanS}$ }
- 15: $\text{UnclassifiedAxioms} := \text{UnclassifiedAxioms} \setminus \mathcal{M}_{\text{COARFurther}}$
- 16: $\text{UnclassifiedModuleAxioms} := \text{UnclassifiedModuleAxioms} \setminus \mathcal{M}_{\text{COARFurther}}$
- 17: $\text{ClassifiedC} := \text{ClassifiedC} \cup (\mathcal{M}_{\text{COARFurther}} \cap \text{C}^+(\mathcal{O}))$
- 18: **end while**
- 19: **end while**
- 20: **return** \mathcal{H}

4. Experiment Setting

We use Hermit version 1.3.8⁵, Konclude [29] version v0.7.0-1135⁶ as base reasoners in this paper. We use our own implementation of a MORE-like approach (see [10] page 11) which classifies \mathcal{T}_{EL} by ELK and \mathcal{T}_{RAS} by Hermit described in [10]. This makes the comparison of the algorithms more precise as they all build off the same code base.

We ran three experiments. The first two experiments are essentially expanded versions of those in [10] and demonstrate the differences of the ReAD_G and Crane approaches over ReAD and the baseline systems. In these two experiments, all the ontologies are classifiable using normal Hermit, thus the experiments explore glass-box modular reasoning as a potential marginal improvement over monolithic and black-box MORE systems.

⁵<http://www.hermit-reasoner.com>

⁶<https://github.com/konclude/Konclude>

In the final experiment, we look at several versions of SNOMED CT which enrich the \mathcal{EL}^{++} version to include disjunctions. These ontologies prove difficult to impossible for HerMiT and MORE-like so we expand our comparison to include Konclude, which is capable of classifying all version. This experiment explores whether our modular optimisations applied to HerMiT can be competitive with Konclude in an otherwise impossible for HerMiT case.

We use OWL API [30] version 3.4.3, ELK version 0.4.2 and HerMiT (modified as indicated) version 1.3.8 in all our systems. Crane also uses the module extraction code from MORE⁷.

Most of experiments have been performed on Intel(R) Core(TM) i7-6700HQ CPU 2.60GHz RAM 8GB, called Mach₁, running Java version 1.8 with an initial heap size of 1GB and a maximal heap size of 8GB. Time is measured in CPU time. All experiments related to Konclude have been performed on Intel(R) Core(TM) i7-5820K CPU 3.30GHz RAM 62GB, called Mach₂, running Java version 1.8 JDK with an initial heap size of 1GB and a maximal heap size of 60GB.

5. Experiments

Throughout these experiments, we report classification times *excluding* the time to compute modules or the AD even when interleaved in Crane, though we do report the Crane modularising time separately. This introduces a potentially misleading bias in our reporting that we strongly caution against. Indeed, if one is starting “cold”, i.e., from an unmodularised input, one should presume that all systems with an unreported modularisation time might have taken longer, even much longer. While that is not always the case, it is often enough, especially for AD-based systems that it is the right bet.

We do this because we are trying to understand *how* exploiting modules affects the classification process itself. If an approach to exploiting modules does not win then the issue of modularisation overhead is moot.

Furthermore, there is an in principle solution to high modularisation cost: modularise “offline” and amortise the cost. Since an AD based storage format and, indeed, AD-maintenance algorithms are both straightforward, it is reasonable to focus on classification itself. Plus, speeding up AD computation is ongoing with some promising efforts.

We report on Crane’s modularisation time since we had originally hoped that this coarsening time would reduce the modularisation overhead to negligible. It did not do that overall, but there are some interesting results around that.

5.1. Bioportal and HerMiT

In this section, we re-use a corpus from 2017 NCBO BioPortal ontology already described in [10] which contains 438 ontologies. We removed ABox axioms from all these ontologies, those ontologies that are empty and those ontologies for which we cannot compute an AD (6) or which HerMiT cannot handle (37);⁸ this leaves us with 308 ontologies.

⁷<https://github.com/anaphylactic/MORE>

⁸HerMiT threw OutOfMemory exceptions or timed-out after 10 hours for 11 ontologies; it failed to handle 26 ontologies due to unsupported syntax or syntax errors.

We further discarded the 164 ontologies that are either purely \mathcal{EL}^{++} (122 ontologies) or have no \mathcal{EL}^{++} modules (42 ontologies)⁹ which leaves us 63 ontologies with *non-deterministic tableaux graphs* and 81 ontologies with *deterministic tableaux graphs*; for the latter, Hermit does not check STs as the concept name satisfiability tests produce, as a side-effect, all subsumers of each concept name. We use CT to represent the classification time and H, M, R, RG, RGP, C for Hermit, MORe-like, ReAD, ReAD_G, ReAD_{GP} and Crane respectively.

5.2. Results

We show the results for deterministic ontologies in Table 1.

For the first three columns, we have roughly what we might expect: A MORe like approach shows, with some noise, some improvement over monolithic Hermit. ReAD does not show any systematic improvement over MORe. ReAD_G shows some dramatic improvements but also some odd reversals, especially for CHEBI and HRDP where it is an order of magnitude worse than the baseline! Parallelisation improves matters greatly but not quite enough to ensure dominance over baseline, at least, not with the number of cores we had available. Crane, on the other hand, dominates, and typically by two orders of magnitude. The catch is that if we include the modularisation time, Crane is hopeless. Thus, with current modularisation technology, Crane needs to offline modularisation.

Table 1

The classification time among 7 deterministic ontologies in bins (2) and (3). The time is shown in seconds. PrT is short for pre-processing time (including computing coarsened AD). Problematic times are bolded.

Ontology	CTH(\mathcal{O})	CTM(\mathcal{O})	CTR(\mathcal{O})	CTRG(\mathcal{O})	CTRGP(\mathcal{O})	CTC(\mathcal{O})	C PrT
FTC	1,445.94	1645.75	1,322.09	225.34	40.68	2.54	13,335
GO*	834.66	695.42	618.84	233.24	48.90	2	14,438
CHEBI*	66.70	47.78	49.02	530.26	96.78	4	31,870
FYPO	24.88	11.25	8.27	17.76	3.74	0.14	296
VTO	4.68	0.31	0.34	1.02	1.01	0.19	2.17
HRDO	3.88	4.04	3.86	39.91	7.73	3.31	12,579
ONL-MR-DA	2.75	2.86	3	6.11	1.33	0.06	0.19

We might expect deterministic ontologies are the best case for Hermit and the worst case for any granular approach, but the results for non-deterministic ontologies, shown in Table 2, are startling.

Hermit, MORe, and ReAD show their usual pattern with somewhat less noise. But, ReAD_G shows quite bad behavior which parallelisation only mitigates in most cases. While throwing threads at a problem to get faster overall performance is reasonable, throwing lots of threads to just catch up with baseline is not.

Crane, excluding pre-processing time, continues its near dominance, only falling behind baseline with the ONL-MSA and behind ReAD with PHAGE. While dominant, the degree of

⁹These are, indeed, comparable with a MORe system... but it would reduce to a comparison with the expressive reasoner component and thus would not test glass vs. black-box issues. Testing against these ontologies is future work.

dominance is less robustly significant. While sometimes it is 2 orders of magnitude faster, there are cases where it is merely a nice speed up or, for CAO, barely an improvement.

Table 2

The classification time among 16 non-deterministic ontologies in bins (2) and (3). The time is shown in seconds.

Ontology	CTH(\mathcal{O})	CTM(\mathcal{O})	CTR(\mathcal{O})	CTRG(\mathcal{O})	CTRGP(\mathcal{O})	CTC(\mathcal{O})	C PrT
CAO	1071.13	1057.46	666.68	2392.33	1826.35	1,039	0.02
PHAGE*	606.36	475.67	449.10	120.05	50.23	2	20,699
DCO-DEBUGIT	386.80	431.75	382.28	3150.41	1523.17	109.32	0.22
VO	78.92	94.19	85.86	35.29	9.61	0.45	1.7
NTDO	76.65	84.92	86.61	248.09	100.52	10.12	0.49
NCIT	75.73	57.38	57.25	4068.86	1349.88	38.81	857.42
ONL-MSA	9.02	9.21	9.14	33.85	18.71	19.32	0.26
CCONT	8.28	8.08	7.61	37.81	16.11	0.46	19.98
FB-CV	5.09	5.45	5.43	8.02	7.28	0.1	0.484
FOODON	2.11	0.30	0.21	0.49	0.27	0.015	3.52
NEMO	2.09	3.09	2.26	5.47	2.47	0.015	0.56
TOK	1.95	2.28	2.31	2.48	2.25	0	0.038
ECSO	1.39	1.28	1.28	4.06	2.13	0.079	1.61
HUPSON	1.30	1.34	1.16	8.72	4.05	0.047	0.28
MHC	1.23	1.14	1.14	10.81	4.36	0.046	1.95
EPO	1.11	1.02	1.07	12.39	4.57	0.329	0.13

By itself, these results do not clearly establish a case for requiring the infrastructure overhaul needed to mitigate the modularisation time, even for Crane. However, the fact that coarse-grain modules tend to dominate performance and often produce orders of magnitude speed ups suggests that investigating more ways to share information between classification of modules would be fruitful. Clearly, the finest grain modules disrupt *something*, but Crane shows that *some* level of module by module classifying is a big win.

5.3. SNOMED CT Experiment

SNOMED CT [31] is a medical terminology ontology that describes the knowledge of health information and is used in medical records as a source of unambiguous, medicine processable terms.¹⁰ The official development of SNOMED CT is constrained to be in \mathcal{EL}^{++} and is well-supported by the reasoner ELK, e.g. the 2021-Jan-28 version SNOMED CT ontology is classified by ELK in only 11 seconds.

Extending the SNOMED CT ontology to be an expressivity beyond \mathcal{EL}^{++} draws a lot of interest in both academic research [5] and industry but generally has been prohibited by practical concerns about computational difficulty.

From conversation with a SNOMED CT developer, we found that 4 minutes to reclassify was about as much as they were willing to pay. They also helped derive three sets of more expressive axioms that they found desirable from a modelling perspective. We combined these

¹⁰<https://www.snomed.org/>

with two versions of \mathcal{EL}^{++} SNOMED CT to produce a test set.¹¹ Note that this is the basic scenario that MORE was designed for: A mostly \mathcal{EL}^{++} ontology with some more expressive additions. The base versions were from 2020-July (S20July) and 2021-Jan (S21Jan). We add the 9-axiom expansion to S20July and the other two to S21Jan. For some details of all versions, see Table 5.3. Note that the length of these ontologies is much larger than its “size” indicating a large average length of axiom.

The 9 axiom extension have axioms of two forms and was part of an initial proof of concept effort:

$$\begin{aligned} 3 \text{ axioms of the form } & A_1 \equiv A_2 \sqcup A_3, \\ 6 \text{ axioms of the form } & A_4 \equiv A_5 \sqcap \exists r_1. (\exists r_2. \neg A_6 \sqcap \exists r_3. A_7 \sqcap \exists r_4. A_8 \sqcap \exists r_5. A_9) \end{aligned}$$

The modelling scenario for the remaining versions is based on importing into SNOMED CT a modified version of an Anatomy ontology to incorporate a disjunction based approach to propagation of properties through part-whole relations (see [32] for an extensive discussion of the modelling problem and different ways of solving it).

The basic move is to replace axioms in Anatomy of the form:

$$S \equiv \text{BodyStructure} \sqcap \exists \text{AllOrPartOf}. E \sqcap \exists \text{Laterality}. \text{QualifierValue} \quad (1)$$

with $S \in \text{StructureConcept}$, $E \in \text{EntireConcept}$, with axioms of the form:

$$S \equiv \text{BodyStructure} \sqcap (E \sqcup \exists \text{ProperPartOf}. E) \sqcap \exists \text{Laterality}. \text{QualifierValue}. \quad (2)$$

One then deletes the corresponding atomic subsumptions between a StructureConcept and EntireConcept in SNOMED CT and imports the modified ontology. The deleted subsumptions are then entailed.

This produces an extremely difficult ontology even if, as we did, you only convert a random sample of 1000 style 1 axioms (SA21Jan₁₀₀₀). To make an intermediately difficult version (S21Jan₇₉₀₅), we extracted all 7,905 axioms in the Anatomy ontology of the form in Equation 1, modified them, and added them directly to SNOMED CT, discarding the rest of Anatomy. We also left in the relevant atomic subsumptions to (hopefully) ease matters. Thus we have three expressive versions of SNOMED CT of ascending difficulty.

Table 3 shows the number of TBox axioms in the variants of the SNOMED CT ontologies and the length of these ontologies. We notice these SNOMED CT ontologies all have axioms with a generally complex structure since the length of these ontologies is more than 5 times larger than their size.

5.3.1. Results

As baseline, we note that ELK classifies S20July and S21Jan in 9 seconds and 12s respectively. In contrast to this, HermiT classifies these two ontologies with 5,199s (around 1.44 hours) and 5,929s (around 1.65 hours). All of the MORE descendants, including our own, show ELK like performance on these as the \mathcal{EL}^{++} module is the entire ontology.

Table 4 shows the results of classifying our three extended versions of SNOMED CT by our prior test systems (excluding non-parallel ReAD) with the addition of Konclude..

¹¹Note, for historical reasons, we did not do a full cross product.

Table 3

Different SNOMED CT extensions.

	Length	Size	DL
S20July	1,907,918	355,214	\mathcal{EL}^{++}
S20July ₉	1,907,924	355,217	\mathcal{ALC}
S21Jan	1,952,625	355,664	\mathcal{EL}^{++}
Anatomy	159,371	42,579	\mathcal{EL}^{++}
S21Jan ₇₉₀₅	2,047,125	372,114	\mathcal{ALC}
SA21Jan ₁₀₀₀	2,049,223	373,493	\mathcal{ALC}

Table 4Different SNOMED CT extensions with their classification time checked by Crane, Konclude, Hermit, MORE like approach. We use M for Memory. The time is shown in **minutes**.

	CTH(\mathcal{O})	CTM(\mathcal{O})	CTRGP(\mathcal{O})	CTC(\mathcal{O})	CTK(\mathcal{O})	C PrT	K M
S20July ₉	97	5.43	1.9	0.02	1.77	46.15	11.52GB
S21Jan ₇₉₀₅	TimeOut	2,280	30	8.3	20	123.77	22.4GB
SA21Jan ₁₀₀₀	TimeOut	TimeOut	TimeOut	87.48	191.32	412.48	29.5GB

The first observation is that only Crane and Konclude handle all three versions. We see a variably steep slowdown for all reasoners across the samples. None of the reasoners approach the 4 minute mark for the most realistic version, in spite of it being only a partial conversion.

That said, Crane beats Konclude by at least an order of magnitude on pure classification time and shows a competitive time when modularisation time is included. Moreover, Crane uses much less memory, never exceeding 8GB even for the most difficult version while Konclude needs 29GB.¹² Finally, we can, in principle, improve Crane’s performance by scaling up the cores available. The dramatic improvement of granular ReAD over more for S21Jan₇₉₀₅ suggests that its optimisations do distinguish it if not robustly.

6. Conclusion

These results suggest that light touch, glass box modularisation is a useful optimisation approach even given the inconveniences of having to manage ontologies in a decomposed form. Further tuning of the coarseness factor in Crane needs to be explored and there may be a value that is effective across the board. Of course, once your ontology is modularised and classified, modular incremental algorithms can be used which may bring the re-classification time of even the most difficult versions of SNOMED CT to the 4 minute mark. Using Konclude as the delegate reasoner for the remainder module might accomplish that directly.

Understanding the causal story of the interaction of various granularities of modularisation with other optimisations is critical. If we can identify these factors we may be able to derive a notion of module that is “right sized” for reasoning.

¹²We run experiments for Konclude with another machine with maximal heap size of 30GB since Konclude throw out of memory for 8GB heap size as we described in Section 4

Acknowledgments

Thanks to Dr Yongsheng Gao from IHTSDO for providing various SNOMED CT ontologies including how to enrich them beyond \mathcal{EL} in a realistic way.

References

- [1] B. Cuenca Grau, I. Horrocks, Y. Kazakov, U. Sattler, Modular reuse of ontologies: Theory and practice, *Journal of Artificial Intelligence Research* 31 (2008) 273–318.
- [2] U. Sattler, T. Schneider, M. Zakharyashev, Which kind of module should I extract?, in: *Proc. of DL-09*, volume 477 of *CEUR*, 2009.
- [3] F. Baader, I. Horrocks, C. Lutz, U. Sattler (Eds.), *An Introduction to Description Logic*, Cambridge University Press, 2017.
- [4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*, CUP, 2003.
- [5] A. A. Romero, B. Cuenca Grau, I. Horrocks, More: Modular combination of owl reasoners for ontology classification, in: *International Semantic Web Conference*, Springer, 2012, pp. 1–16.
- [6] D. Tsarkov, I. Palmisano, Chainsaw: a metareasoner for large ontologies., in: *Proc. of ORE 2012*, 2012.
- [7] B. Cuenca Grau, C. Halaschek-Wiener, Y. Kazakov, History matters: Incremental ontology reasoning using modules, in: *Proc. of ISWC-07*, volume 4825 of *LNCS*, 2007, pp. 183–196.
- [8] N. Matentzoglou, B. Parsia, U. Sattler, Owl reasoning: Subsumption test hardness and modularity, *Journal of automated reasoning* 60 (2018) 385–419.
- [9] H. Zhao, B. Parsia, U. Sattler, Avoiding subsumption tests during classification using the atomic decomposition, in: *DL-19*, volume 573, 2019.
- [10] H. Zhao, B. Parsia, U. Sattler, Read: Ad-based modular ontology classification, in: *European Conference on Logics in Artificial Intelligence*, Springer, 2021, pp. 210–224.
- [11] F. Baader, S. Brandt, C. Lutz, Pushing the \mathcal{EL} envelope, in: *Proc. of IJCAI-05*, 2005.
- [12] F. Baader, S. Brandt, C. Lutz, Pushing the \mathcal{EL} envelope further, in: *Proc. of OWLED 2008*, 2008.
- [13] F. Baader, C. Lutz, B. Suntisrivaraporn, Cel—a polynomial-time reasoner for life science ontologies, in: *International Joint Conference on Automated Reasoning*, Springer, 2006, pp. 287–291.
- [14] Y. Kazakov, M. Krötzsch, F. Simančík, The incredible ELK, *Journal of automated reasoning* 53 (2014) 1–61.
- [15] I. Horrocks, O. Kutz, U. Sattler, The even more irresistible *SRITQ*, in: *KR-06*, 2006, pp. 57–67.
- [16] D. Tsarkov, I. Horrocks, Fact++ description logic reasoner: System description, in: *International joint conference on automated reasoning*, Springer, 2006, pp. 292–297.
- [17] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, Y. Katz, Pellet: A practical owl-dl reasoner, *Journal of Web Semantics* 5 (2007) 51–53.

- [18] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, Hermit: an owl 2 reasoner, *Journal of Automated Reasoning* 53 (2014) 245–269.
- [19] N. A. Matentzoglou, Module-based classification of OWL ontologies, Ph.D. thesis, University of Manchester, 2016.
- [20] F. Baader, B. Hollunder, B. Nebel, H.-J. Profitlich, E. Franconi, An empirical analysis of optimization techniques for terminological representation systems: or: 'making kris get a move on', in: *3rd International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, 1992, pp. 270–281.
- [21] B. Glimm, I. Horrocks, B. Motik, R. Shearer, G. Stoilos, A novel approach to ontology classification, *Journal of Web Semantics* 14 (2012) 84–101.
- [22] C. Del Vescovo, M. Horridge, B. Parsia, U. Sattler, T. Schneider, H. Zhao, Modular structures and atomic decomposition in ontologies, *Journal of Artificial Intelligence Research* 69 (2020) 963–1021.
- [23] S. Ghilardi, C. Lutz, F. Wolter, Did I damage my ontology? A case for conservative extensions in description logics, in: *KR, AAAI Press*, 2006, pp. 187–197.
- [24] B. Konev, C. Lutz, D. Walther, F. Wolter, Semantic modularity and module extraction in description logics, in: *ECAI-08*, 2008, pp. 55–59.
- [25] C. Lutz, D. Walther, F. Wolter, Conservative extensions in expressive description logics, in: *IJCAI*, 2007, pp. 453–458.
- [26] B. Cuenca Grau, I. Horrocks, Y. Kazakov, U. Sattler, Extracting modules from ontologies: A logic-based approach, in: H. Stuckenschmidt, C. Parent, S. Spaccapietra (Eds.), *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *LNCS, SV*, 2009, pp. 159–186.
- [27] C. Del Vescovo, B. Parsia, U. Sattler, T. Schneider, The modular structure of an ontology: Atomic decomposition, in: *IJCAI*, 2011, pp. 2232–2237.
- [28] R. Shearer, I. Horrocks, Exploiting partial information in taxonomy construction, in: *International Semantic Web Conference*, Springer, 2009, pp. 569–584.
- [29] A. Steigmiller, T. Liebig, B. Glimm, Konclude: system description, *Journal of Web Semantics* 27 (2014) 78–85.
- [30] M. Horridge, S. Bechhofer, The owl api: A java api for owl ontologies, *Semantic web* 2 (2011) 11–21.
- [31] S. Schulz, B. Suntisrivaraporn, F. Baader, M. Boeker, Snomed reaching its adolescence: Ontologists' and logicians' health check, *International journal of medical informatics* 78 (2009) S86–S94.
- [32] P. Seyed, A. L. Rector, U. Sattler, B. Parsia, R. Stevens, Representation of part-whole relationships in snomed ct, in: *ICBO, Citeseer*, 2012.