Sequoia: A Consequence-Based Reasoner for $\mathcal{SROIQ}$

David Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks

University of Oxford, Oxford OX1 3QD, UK
david.tena.cucala@cs.ox.ac.uk
bernardo.cuenca.grau@cs.ox.ac.uk
ian.horrocks@cs.ox.ac.uk

Abstract. We describe and evaluate a new extension of Sequoia, the consequence-based reasoner for $\mathcal{SRIQ}$ presented in [4], which introduces support for nominals. Our reasoner implements the calculus for the logic $\mathcal{ALCHOIQ}^+$ presented in our recent work [30]. By using well-known ontology preprocessing methods, Sequoia can be used to classify ontologies in $\mathcal{SROIQ}$ —a rich DL covering all features of OWL 2 DL apart from datatypes. We show that Sequoia offers competitive performance and thus it provides an important addition to the repertoire of practical implementation techniques for reasoning in expressive DLs.

1 Introduction

Consequence-based (CB) reasoning [1] is a promising approach to DL reasoning which combines features of (hyper)tableau [6, 9, 10, 27, 28, 31] and resolution-based [11, 14, 18, 21] algorithms. On the one hand, similarly to resolution, CB calculi derive formulae entailed by the ontology (thus avoiding the explicit construction of large models), and they are typically worst-case optimal. On the other hand, CB calculi organise clauses into *contexts* arranged as a graph structure reminiscent of that used for model construction in (hyper)tableau; this prevents CB calculi from drawing many unnecessary inferences and yields a nice goal-oriented behaviour. Furthermore, in contrast to both resolution and (hyper)tableau, CB calculi can verify a large number of subsumptions in a single execution, allowing for one-pass classification. Finally, CB calculi have proved highly effective in practice. Leading reasoners for lightweight DLs such as ELK [17] or Snorocket [20] are based on CB calculi. Furthermore, prototypical implementations of CB calculi for more expressive languages, such as Sequoia [4] or Avalanche [32], have shown promising results.

CB calculi were first proposed for the $\mathcal{EL}$ family of DLs [1, 7], and later extended to more expressive logics such as Horn-$\mathcal{SHIQ}$ [16], Horn-$\mathcal{SROIQ}$ [22], and $\mathcal{ALCH}$ [26]. Simančík et al. proposed a unifying framework for CB reasoning in $\mathcal{ALCHI}$, which explicitly introduced for the first time the notion of *context* as a mechanism for constraining resolution inferences and make reasoning goal-directed [25]. This framework was subsequently extended to $\mathcal{ALCHIQ}^+$, which supports number restrictions and inverse roles, as well as self-reflexivity and disjoint roles [5]. The framework was also extended to $\mathcal{ALCHOI}$, which supports inverse roles and nominals [29], and $\mathcal{ALCHOQ}$, which supports nominals and number restrictions [15]. Finally, in our recent work we extended the framework to $\mathcal{ALCHOIQ}^+$, which supports all of the aforementioned constructs [30].

This paper describes an extension of the CB reasoner Sequoia [4] with full support for nominals and novel optimisations. Our reasoner implements the calculus in [30] and supports all of OWL 2 DL with the exception of datatypes.[1]

In Section 2 we recapitulate the calculus and the classification algorithm underpinning our extension of Sequoia. In Section 3 we present the system's architecture and discuss implementation techniques and optimisations. Finally, in Section 4 we present the results of an evaluation of Sequoia against state-of-the-art reasoners over a corpus of 777 DL ontologies. Our evaluation shows that Sequoia was able to classify more ontologies than any other evaluated reasoner, and its overall performance was highly competitive. Furthermore, the performance of Sequoia was also comparable to that of ELK on OWL 2 EL ontologies.

2 Consequence-Based Reasoning in Sequoia

In this section we outline the classification algorithm in Sequoia and the CB calculus for $\mathcal{ALCHOIQ}^+$ that underpins it. A complete description of the classification algorithm, the calculus, and their properties, can be found in [30].

The calculus implemented in Sequoia is based on the framework for CB reasoning in [25]. The framework defines a *context structure* for a given ontology $\mathcal{O}$. Intuitively, a context structure is a graph having *contexts* as nodes, where each context summarises a set of elements in a model of $\mathcal{O}$. Each context has a unique *core*—a condition satisfied by all domain elements represented by the context; for instance, a context v with core $A(x)$ represents all elements interpreted as instances of A in a model of $\mathcal{O}$.

Consequences derived by the calculus are distributed among contexts. Consequences in a context v are expressed as *context clauses*, which are Skolemised first-order clauses restricted to a form wherein variables x and y receive a special treatment. Rather than allowing x and y to range over the entire domain, variable x ranges only over elements represented by v and, for any such element t, variable y may range only over elements related to t via particular roles. For instance, in a context v with core $A(x)$, context clause $R(y,x) \rightarrow B(x) \vee C(x)$ represents that, for every instance t of A in a model, if $R(s,t)$ holds for some s in the model, then either $B(t)$ or $C(t)$ holds as well.

The calculus in Sequoia has been designed for $\mathcal{ALCHOIQ}^+$ ontologies expressed as a set of DL-clauses of a particular form; any $\mathcal{SROIQ}$ ontology can be (exponentially) reduced to such clauses using well-known techniques [8, 23, 24]. To classify an ontology $\mathcal{O}$, the calculus creates a context structure with a fresh context v_B for each atomic concept B in $\mathcal{O}$. This context will gather all

[1] `http://www.cs.ox.ac.uk/isg/tools/Sequoia/`

relevant consequences expressing constraints on instances of B. Next, the rules of the calculus are applied until the context structure becomes saturated. Each of the following inference rules can be applied to a context v.

– Rule Core ensures that all elements represented by v satisfy the condition expressed by the core of the context. For instance, if the core of v is $A(x)$, the rule will add context clause $\top \rightarrow A(x)$ to v.
– Rule Hyper applies hyperresolution [3] using context clauses in v as side premises and a clause in $\mathcal{O}$ as main premise; e.g., the rule adds clause $S(y,x) \rightarrow A(x) \vee C(y)$ to v if clauses $\{S(y,x) \rightarrow S(y,x), \top \rightarrow A(x) \vee B(x)\}$ appear in v, and $S(z,x) \wedge B(x) \rightarrow C(z) \in \mathcal{O}$; inferences by Hyper can only unify variable x in $\mathcal{O}$ with variable x in context clauses; this ensures that consequences preserve the form of context clauses.
– Rule Eq uses an equality in the head of a context clause in v to make a substitution in the head of another context clause in v. For instance, if we have context clauses $\{\top \rightarrow R(x, f(x)), \top \rightarrow f(x) \approx g(x)\}$ in a context v, rule Eq will write the context clause $\top \rightarrow R(x, g(x))$ in v. This rule, together with the next two, implement a form of paramodulation reasoning [2].
– Rule Ineq eliminates literals of the form $t \not\approx t$ from heads of context clauses in v. For instance, if a context clause $A(x) \rightarrow f(x) \not\approx f(x)$ appears in v, the rule will write $A(x) \rightarrow \bot$ in this context.
– Rule Factor replaces two equalities in the head of a context clause in v by two equivalent equations; this step is required for completeness. For instance, if there is in v a context clause $\top \rightarrow g(x) \approx o \vee g(x) \approx f(x)$, this rule will add in v the context clause $\top \rightarrow f(x) \not\approx o \vee g(x) \approx f(x)$.
– Rule Elim eliminates redundant context clauses, such as those subsumed by other context clauses in v.

In addition to these rules, which can be independently applied to each context, the calculus provides rules that may involve two neighbouring contexts in the context structure, where an edge (v, w) labelled with function f indicates that, for each element t represented by v, element $f(t)$ is represented by w.

– Rule Succ ensures that literals with function symbols can participate in inferences. If v contains $\top \rightarrow B(f(x))$ and $B(x) \rightarrow C(x) \in \mathcal{O}$, we cannot apply Hyper to infer $\top \rightarrow C(f(x))$ in v; instead, we apply Succ to derive a context clause $B(x) \rightarrow B(x)$ that can participate in such inference. This clause is written in a fresh context w or in a suitable context w chosen from those already in the context structure; in either case, an edge (v, w) labelled "f" is added if not already in place. The policy for choosing w (or deciding to create it fresh) is a parameter of the calculus called the *expansion strategy.*
– Rule Pred applies a hyperresolution step involving clauses spread between two neighbouring contexts. For instance, if context clause $\top \rightarrow R(x, f(x))$ appears in v, and context clause $R(y, x) \rightarrow F(y)$ appears in w, and there is an edge (v, w) labelled f, this rule will add $\top \rightarrow F(x)$ in v.

To deal with nominals, the calculus allows for ground atoms in context clauses, and it introduces a special "root context" v_r, where we allow applications of Hyper that unify variable x with a constant; most inferences on named individuals take place in this context. The calculus also introduces nominal-specific rules which allow us to easily propagate ground atoms across contexts.

– Rule r-Succ is analogous to Succ, and propagates literals from a context v to the root context; e.g., the rule can be applied to a clause $\top \rightarrow B(o)$ in v to generate clause $B(o) \rightarrow B(o)$ in the root context.
– Rule r-Pred is analogous to Pred, and it is used to perform hyperresolution involving clauses spread between the root context and some other context; e.g., if rule $R(y, o) \rightarrow F(y)$ appears in the root context, and context clause $\top \rightarrow R(x, o)$ appears in a context v connected to the root context by an edge labelled "o", the rule generates $\top \rightarrow F(x)$ in v.
– Rule Join applies ground resolution to clauses in the same context; e.g., it derives $\top \rightarrow F(x)$ in v if it contains $\top \rightarrow F(x) \vee A(o)$ and $A(o) \rightarrow F(x)$.
– Rule Nom generates fresh constants to represent successors of named individuals by inverse roles constrained by an "at-most" number restriction. Introducing fresh constants captures the nominal-like behaviour of these elements explicitly. For instance, if $S(o, y) \rightarrow S(o, y)$ appears in the root context, and S is a functionally restricted role according to $\mathcal{O}$, Nom adds $S(o, y) \rightarrow y \approx o_S$ to the root context. Applications of this rule are suitably restricted to ensure termination.

When the context structure becomes saturated under all rules of the calculus, each relevant subsumption for concept B can be read directly from context v_B. In particular, by choosing appropriate parameters for the calculus (such as a suitable literal order), soundness and completeness of the calculus ensure that v_B will contain a context clause $\top \rightarrow A(x)$ if and only if $\mathcal{O} \models B(x) \rightarrow A(x)$.

3 System Architecture

Figure 1 summarises the architecture of Sequoia. Version 0.7.0 is implemented in Scala and available through the OWL API interface [12].[2] The *Sequoia Reasoning Engine* is the core component of the system, and it currently supports consistency checking and classification as reasoning tasks. We next describe the key components of the Sequoia Reasoning Engine.

Ontology loading, clausification, and indexing. Sequoia accepts as input OWL 2 DL ontologies without datatypes and `HasKey` axioms; by default, Sequoia will throw an exception if the input ontology is not supported, but the reasoner can be configured to work in best-effort mode and ignore unsupported axioms. Sequoia converts OWL axioms to normalised DL-clauses; for this, it first encodes away role inclusion axioms using a variant of the algorithm in [13] (see [4] for

[2] Version 0.6.2, which does not support nominals, is also available as a Protégé plugin and a command line utility. These will soon be updated to the latest version.

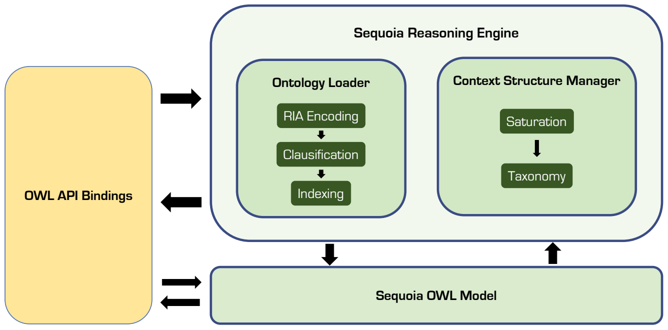

Fig. 1. Architecture of Sequoia. Boxes represent the key components of the reasoner and their sub-components. Arrows represent flows of information.

details), and then it applies the *clausification* sub-routine, which uses a variant of structural transformation to produce a set of DL-clauses of the form required by the calculus. Like first-order theorem provers, Sequoia uses several indexes to efficiently identify clauses that can participate in an inference or a simplification rule. Sequoia uses custom indexing techniques adapted to the special syntax of ontology and context clauses. The version of Sequoia described in this paper uses very similar index structures to those described in [4].

Reasoning with a context structure. Once an ontology $\mathcal{O}$ has been loaded, the Sequoia Reasoning Engine can check its consistency and classify it. The Context Structure Manager creates, saturates, and interprets the contexts of a context structure. For consistency checking, it initialises a single context with an empty core. For classification, it initialises a context v_A with core $A(x)$ for each atomic concept A in $\mathcal{O}$ (excluding auxiliary predicates created during clausification), and configures the context parameters (such as the literal ordering) to ensure that for each atomic concept B, inclusion $\top \rightarrow B(x)$ will be derived in v_A if and only if $\mathcal{O} \models A(x) \rightarrow B(x)$. Each context in the context structure is implemented as a fibre. When a context v is created, an empty set $\mathcal{S}_v$ of context clauses is initialised, together with an auxiliary (empty) set of *unprocessed* clauses $\mathcal{U}_v$. Then, Sequoia applies the steps below to v; since the derivation of new inferences within each context is independent from the state of other contexts, this part of the algorithm is easily parallelisable (see [4] for details).

1. Apply the Core rule; add any resulting clauses to $\mathcal{U}_v$ and $\mathcal{S}_v$.
2. Apply the Hyper rule using all ontology clauses of the form $\top \rightarrow \Delta$; add all derived clauses to $\mathcal{U}_v$ and $\mathcal{S}_v$.
3. While $\mathcal{U}_v$ is not empty:
   (a) Pick a clause C from $\mathcal{U}_v$; let $\mathcal{L}$ be the set of maximal literals in C.

- (b) Apply all inferences with the Hyper rule involving a literal in $\mathcal{L}$, ontology clauses, and clauses in $\mathcal{S}_v$. Add inferences to sets $\mathcal{U}_v$ and $\mathcal{S}_v$.
- (c) Apply all inferences with the Pred rule involving a literal in $\mathcal{L}$, a clause in a neighbour context, and clauses in $\mathcal{S}_v$. Add inferences to $\mathcal{U}_v$ and $\mathcal{S}_v$.
- (d) Apply all inferences with the r-Pred rule involving a literal in $\mathcal{L}$ and clauses in $\mathcal{S}_v$ and the root context. Add inferences to $\mathcal{U}_v$ and $\mathcal{S}_v$.
- (e) Apply all inferences with the Eq and Factor rules involving a literal in $\mathcal{L}$ and clauses in $\mathcal{S}_v$; add inferences to sets $\mathcal{U}_v$ and $\mathcal{S}_v$.
- (f) Apply all inferences with the Nom rule involving a literal in $\mathcal{L}$, clauses in $\mathcal{S}_v$, and a clause in $\mathcal{O}$; add inferences to sets $\mathcal{U}_v$ and $\mathcal{S}_v$.
- (g) Erase C from $\mathcal{U}_v$.

4. For each new clause added to $\mathcal{S}_v$, propagate the relevant inference(s) to neighbour contexts using Succ or r-Succ. Clauses are propagated through communication channels between contexts.
5. For every new clause added to $\mathcal{S}_v$, if this clause may trigger Pred or r-Pred in a neighbour context w, propagate this clause to a look-up set $\mathcal{P}_w$ in w. The context w will use this set in Steps 3.(c)-(d).
6. *Sleep* until a clause C is received through an incoming communication channel. If this happens, add C to $\mathcal{U}_v$ and go to Step 3.

The Join rule is combined with Pred and r-Pred. The Ineq rule is applied as a simplification after each clause C is derived. Set $\mathcal{S}_v$ uses a redundancy index to ensure that no clause C is added to $\mathcal{S}_v$ if it is subsumed by a clause $C' \in \mathcal{S}_v$. Furthermore, clauses in $\mathcal{S}_v$ subsumed by C are dropped when C is added to $\mathcal{S}_v$.

When all contexts are sleeping, the context structure has become saturated. Then, Sequoia runs the *Taxonomy* sub-routine to read all relevant concept inclusions from the saturated context structure and compute their transitive reduction. The result can be accessed using standard OWL API methods.

Optimisations Sequoia implements a number of optimisation techniques. We discuss those introduced in this version and refer the reader to [4] for the rest.

– We have divided the saturation phase into two consecutive phases. The first phase derives only Horn context clauses until the context structure becomes saturated. The second phase then completes the context structure by deriving all the remaining (non-Horn or Horn) context clauses. This prevents the generation of context clauses with many disjuncts in the head (a known source of inefficiency in Sequoia) which turn out to be subsumed by Horn clauses derived in the first phase.

– The concentration of inferences involving named individuals in the root context prevents the possibility of parallelising such inferences. To address this problem, we split the root context into a separate nominal context v_o for each individual o. In nominal context v_o, we freely swap the constant o for variable x, and disallow applications of the Hyper rule that unify variable x in ontology clauses with constant o in context clauses. To retain completeness, we have suitably modified the calculus rules interacting with the root context;

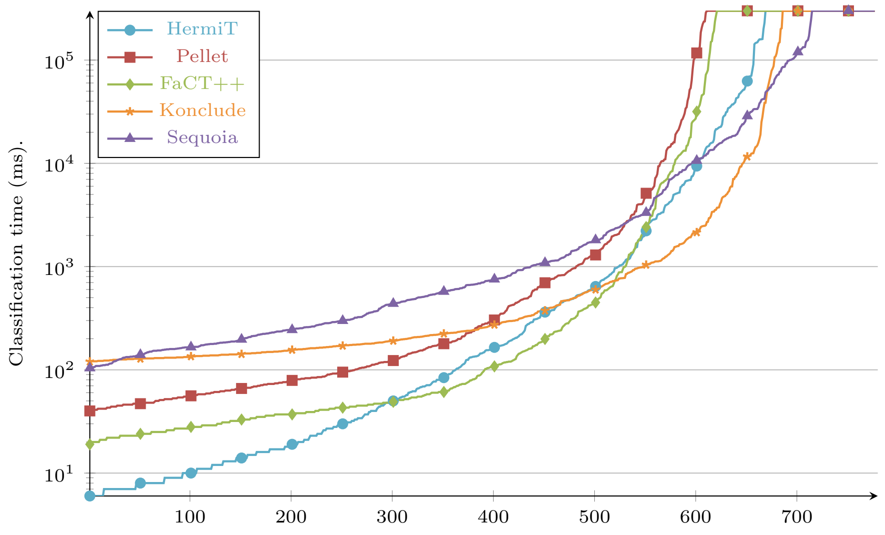

Fig. 2. Classification Times for All Ontologies

for instance, the rule r-Succ propagates clauses of the form $B(o) \to B(o)$ to v_o. Furthermore, we have added rules to ensure that relevant clauses are propagated across nominal contexts.

– Many context clauses of the form $\Gamma \to \Delta \vee B(o)$ turn out to be redundant due to the existence of a clause of the form $\top \to B(o)$ in the root context. The generation of such clauses can often be prevented if we propagate $\top \to B(o)$ immediately after it is derived to all contexts which mention o; to make this task easier, each context keeps track of the constants that have appeared so far in its context clauses.
– For classification tasks, Sequoia requires all atomic concepts to be incomparable on contexts introduced at initialisation. This can be a source of inefficiency: if we have a context clause $\top \to A_1(x) \vee A_2(x) \vee \cdots \vee A_n(x)$ and clauses $A_i(x) \to A(x) \in \mathcal{O}$ for each $1 \leq i \leq n$, the calculus will derive approximately 2^n clauses. However, it is possible to introduce an ordering between these atoms and preserve completeness by allowing *both* maximal atoms and *second-maximal* atoms (i.e. atoms smaller only than maximal atoms) in clause heads to participate in inferences in those contexts. By using this strategy in the example above, our calculus derives $\sim n^3$ clauses.

4 Evaluation

Methodology We evaluated the performance of classification in Sequoia version 0.7.0 against other DL reasoners using the methodology described in [28]. We

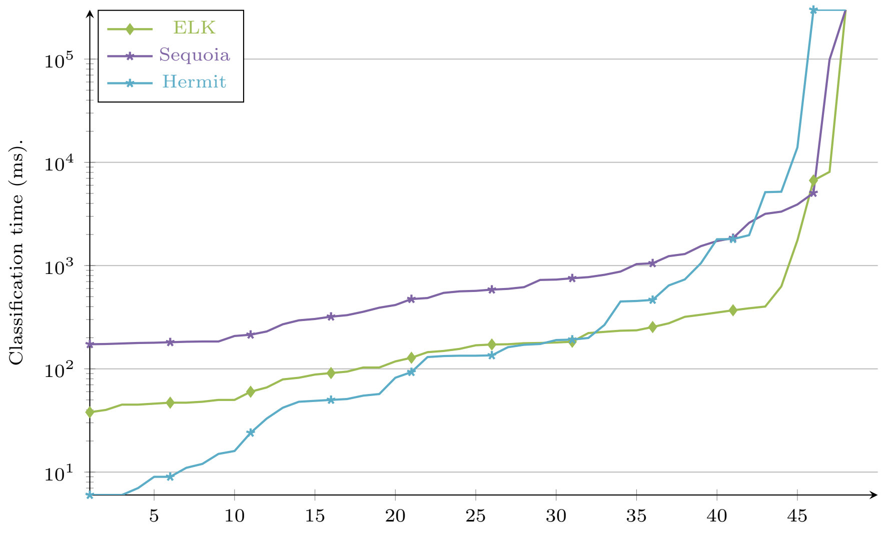

Fig. 3. Classification Times for ELK-supported Ontologies

used the Oxford Ontology Repository as our corpus,[3] and we preprocessed the ontologies according to the following steps:

– Seven ontologies were not in OWL 2 DL due to violations of restrictions on the use of non-simple object properties; we removed all axioms involved in such violations. Moreover, we manually fixed syntax errors on 9 ontologies.
– Sequoia does not yet support datatypes; we have replaced each data range by a fresh class, each data property by a fresh object property, and each literal by a fresh named individual.

As a result, we obtained 777 ontologies, out of which 77 contain nominals in the TBox (that is, excluding ABox assertions). A total of 48 ontologies in the corpus are captured by the fragment of OWL 2 EL supported by ELK.

We ran all experiments on a Dell server with 512 GB of RAM and two Intel CPU E5-2640 V3 2.60 GHz processors, with eight cores per processor and two threads per core. The system OS was Fedora 26, kernel version 4.11.9-300.fc26.x86_64, and Java 1.8.0 update 151. The other evaluated reasoners were HermiT 1.3.8, Konclude 0.6.2, FaCT++ 1.6.5, Pellet 2.4.0, and ELK 0.4.0. To streamline the tests, we accessed all reasoners through the OWL API interface. Konclude does not offer direct access through this interface, so we accessed it via the OWLlink OWL API adapter; we do not expect this to have had a significant impact on performance since axioms were loaded to the OWLlink server before the test started. Reasoners such as ELK and Konclude offer support for parallel reasoning; however, this feature is not yet available in the current version of Sequoia, so we compared all reasoners in single-threaded mode.

[3] `http://www.cs.ox.ac.uk/isg/ontologies/`

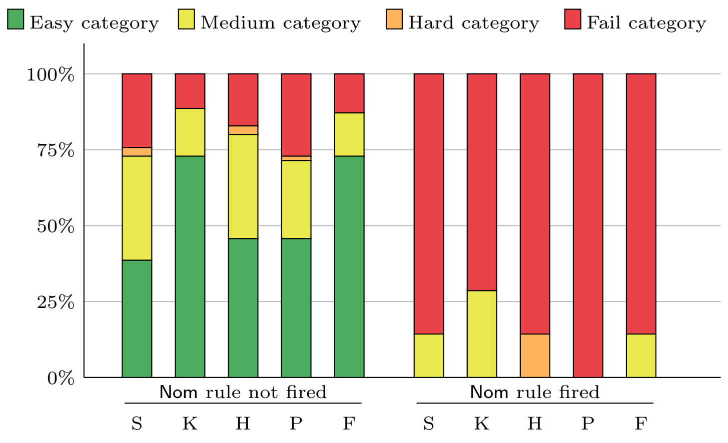

Fig. 4. Percentage per Category of Easy, Medium, Hard, and Fail Ontologies with Nominals, for Sequoia (S), Konclude (K), HermiT (H), Pellet (P), and FaCT++ (F)

For each ontology in the (preprocessed) corpus and for each reasoner, we created a fresh process that used the OWL API to: (i) load the ontology, (ii) create a new instance of the reasoner, (iii) load the ontology to the reasoner, and (iv) ask the reasoner to classify the ontology. The process was allowed to run for 5 minutes; if this threshold was reached, the process was destroyed and we recorded a timeout. Otherwise, we created another fresh process to repeat the task up to three times. We measured the wall-clock duration of the classification task in each repetition. If all repetitions finished without reaching timeout, we recorded the average classification time. Taxonomies were hashed to verify correctness; in all cases, the results of Sequoia matched those of at least one other reasoner. The systems, ontologies, and results of the experiment are available online.[4]

Results The (averaged) classification times for the full corpus are summarised in Figure 2. For each reasoner, we sorted the classification times in ascending order; a value point (n, m) in the figure represents that the n-th smallest average classification time for that reasoner was m milliseconds. Points where $m = 300s$ represent timeouts. We excluded the results for ELK from Figure 2 and included them in Figure 3, which considers only the 48 ontologies supported by ELK. In all cases, we checked correctness using a hash function on the taxonomy and verified that Sequoia agreed with at least one other reasoner.

Figure 4 considers the 77 ontologies with nominals in the TBox and divides them into two buckets. The first one represents the 70 ontologies where the Nom rule did not fire during classification, and the second bucket contains the remaining 7 ontologies. For each bucket and reasoner, we further divided these

[4] `http://krr-nas.cs.ox.ac.uk/2019/DL-sequoia/`

ontologies into four categories: Easy (classification time $< 1s$), Medium ($\geq 1s$ and $< 60s$), Hard ($\geq 60s$ and $< 300s$), and Fail (timeout).

Discussion Sequoia could classify more ontologies than any other reasoner, and it performed especially well compared to others on ontologies requiring at least $30s$ to classify. Sequoia, however, incurred in a measurable overhead on ontologies with classification times up to 10s, which we believe is related to the management of the context structure and the communication between contexts. Timeouts mostly occurred on non-Horn ontologies where saturation produced many clauses with large numbers of disjuncts in the head, which is a well-known source of performance problems in CB reasoning [4]. These are exacerbated for ontologies where an at-most number restriction is applicable to a context with a large number of successors for the same role; this leads to the generation of an exponential number of clauses with quadratically many equalities in the head.

Figure 3 provides evidence of the theoretical pay-as-you-go properties of the calculus, as the scalability of Sequoia is in line with that of ELK (with a roughly constant overhead, maybe due to the choice of data structures). It was not possible to compare the handling of nominals in Sequoia against ELK, for none of the 77 ontologies with nominals in the TBox was fully supported by ELK.

Figure 4 shows that reasoning with nominals is still hard for Sequoia, and further optimisation is needed. This is due to the fact that the global nature of reasoning with nominals does not combine well with the focus on local consequences in CB reasoning [19]. It is also interesting to notice that 7 ontologies required the generation of fresh nominals by the Nom rule during classification; this supports the widespread belief within the community that such occurrences are relatively rare. Interestingly, these 7 ontologies were very hard for all reasoners. Furthermore, note that the calculus in Sequoia is not worst-case optimal if the Nom rule is triggered (the calculus can then run in triple exponential time); however, Sequoia's performance on these ontologies seems to be in line with that of the other reasoners.

5 Conclusion

We have presented a new version of the consequence-based reasoner Sequoia, which supports the whole of OWL 2 DL with the exception of datatypes. Our evaluation shows that the performance of Sequoia is very competitive and that the pay-as-you-go theoretical properties of the underpinning calculus manifest themselves in practice. There is still plenty of room for optimisation. In particular, Sequoia still struggles with several non-Horn ontologies with number restrictions. Nominals can also be a source of problems for Sequoia due to the derivation of large numbers of ground clauses in many contexts. In addition to extending the system to support datatypes, we will be working on further optimisation techniques to address the outstanding practical limitations.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ Envelope. In: IJCAI (2005)
2. Bachmair, L., Ganzinger, H.: Equational Reasoning in Saturation-Based Theorem Proving. In: Bibel, W., Schmidt, P. (eds.) Automated Deduction—A Basis for Applications, vol. I, pp. 353–397. Kluwer (1998)
3. Bachmair, L., Ganzinger, H.: Resolution Theorem Proving. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, vol. I, chap. 2. MIT (2001)
4. Bate, A., Motik, B., Cuenca Grau, B., Tena Cucala, D., Simancik, F., Horrocks, I.: Consequence-based reasoning for description logics with disjunctions and number restrictions. J. Artif. Intell. Res. **63**, 625–690 (2018)
5. Bate, A., Motik, B., Grau, B.C., Simancik, F., Horrocks, I.: Extending consequence-based reasoning to $\mathcal{SRIQ}$. In: KR. pp. 187–196 (2016)
6. Baumgartner, P., Furbach, U., Niemelä, I.: Hyper Tableaux. In: JELIA. pp. 1–17. No. 1126 in LNAI, Springer, Évora, Portugal (1996)
7. Brandt, S.: On subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general TBoxes. In: DL (2004)
8. Demri, S., de Nivelle, H.: Deciding Regular Grammar Logics with Converse Through First-Order Logic. J. Log. Lang. Inf. **14**(3), 289–329 (2005)
9. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: An OWL 2 reasoner. J. Autom. Reasoning **53**(3), 245–269 (2014)
10. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro knowledge representation and reasoning system. Semantic Web **3**(3), 267–277 (2012)
11. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In: ISWC. pp. 383–397 (2005)
12. Horridge, M., Bechhofer, S.: The OWL API: A Java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011)
13. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: KR (2006)
14. Hustadt, U., Schmidt, R.A.: Issues of decidability for description logics in the framework of resolution. In: Caferra, R., Salzer, G. (eds.) Automated Deduction in Classical and Non-Classical Logics. LNCS, vol. 1761. Springer (2000)
15. Karahroodi, N.Z., Haarslev, V.: A Consequence-based Algebraic Calculus for $\mathcal{SHOQ}$. In: Artale, A., Glimm, B., Kontchakov, R. (eds.) DL. CEUR 1879 (2017)
16. Kazakov, Y.: Consequence-Driven Reasoning for Horn $\mathcal{SHIQ}$ Ontologies. In: IJCAI. pp. 2040–2045 (2009)
17. Kazakov, Y., Krötzsch, M., Simančík, F.: The incredible ELK. Journal of Automated Reasoning **53**(1), 1–61 (2014)
18. Kazakov, Y., Motik, B.: A resolution-based decision procedure for $\mathcal{SHOIQ}$. Journal of Automated Reasoning **40**(2-3), 89–116 (2008)
19. Krötzsch, M.: Efficient rule-based inferencing for OWL EL. In: IJCAI (2011)
20. Metke-Jimenez, A., Lawley, M.: Snorocket 2.0: Concrete Domains and Concurrent Classification. In: ORE. CEUR, vol. 1015, pp. 32–38 (2013)
21. de Nivelle, H., Schmidt, R.A., Hustadt, U.: Resolution-Based Methods for Modal Logics. Logic Journal of the IGPL **8**(3), 265–292 (2000)
22. Ortiz, M., Rudolph, S., Simkus, M.: Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In: KR (2010)
23. Schmidt, R.A., Hustadt, U.: The Axiomatic Translation Principle for Modal Logic. ACM Transactions on Computational Logic **8**(4) (2007)
24. Simančík, F.: Elimination of complex RIAs without automata. In: DL (2012)

25. Simančík, F., Motik, B., Horrocks, I.: Consequence-Based and Fixed-Parameter Tractable Reasoning in Description Logics. Artificial Intelligence **209**, 29–77 (2014)
26. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-Based Reasoning beyond Horn Ontologies. In: IJCAI. pp. 1093–1098 (2011)
27. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics **5**(2), 51–53 (2007)
28. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: System description. Journal of Web Semantics **27**, 78–85 (2014)
29. Tena Cucala, D., Cuenca Grau, B., Horrocks, I.: Consequence-based reasoning for description logics with disjunction, inverse roles, and nominals. In: DL (2017)
30. Tena Cucala, D., Cuenca Grau, B., Horrocks, I.: Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. In: IJCAI. pp. 1970–1976 (2018)
31. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: IJCAR. LNAI, vol. 4130, pp. 292–297. Springer (2016)
32. Vlasenko, J., Daryalal, M., Haarslev, V., Jaumard, B.: A Saturation-based Algebraic Reasoner for $\mathcal{ELQ}$. In: PAAR. CEUR, vol. 1635, pp. 110–124 (2016)