Completion Graph Caching for Expressive Description Logics

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

[1] University of Ulm, Ulm, Germany, <first name>.<last name>@uni-ulm.de
[2] derivo GmbH, Ulm, Germany, liebig@derivo.de

1 Introduction

Reasoning in very expressive Description Logics (DLs) such as $\mathcal{SROIQ}$ is often hard since non-determinism, e.g., from disjunctions or cardinality restrictions, requires a case-by-case analysis and since a strict separation between intensional (TBox) and extensional (ABox) knowledge is not possible due to nominals. Current state-of-the-art reasoners for $\mathcal{SROIQ}$ are typically based on (hyper-)tableau algorithms, where higher level reasoning tasks such as classification are reduced to possibly many consistency tests. For large ABoxes and more expressive DLs, reasoning then easily becomes infeasible since the ABox has to be considered in each test.

For less expressive DLs, optimisations have been proposed that allow for considering only parts of the ABox for checking whether an individual is an instance of a specific concept [22,23]. It is, however, not clear how these optimisations can be extended to $\mathcal{SROIQ}$. Furthermore, approaches based on partitioning and modularisation require a syntactic pre-analysis of the concepts, roles, and individuals in a KB, which can be quite costly for more expressive DLs and, due to the static analysis, only queries with specific concepts are supported. Rewriting axioms such that ABox reasoning is improved also carries the risk that this negatively influences other reasoning tasks for which ABox reasoning is potentially not or less relevant.

Another possibility is the caching of the completion graph that is built by the tableau algorithm during the initial consistency check. The re-use of (parts of) the cached completion graph in subsequent tests reduces the number of consequences that have to be re-derived for the individuals of the ABox. Existing approaches based on this idea (e.g., [17]) are, however, not very suitable for handling non-determinism. Since caching and re-using non-deterministically derived facts can easily cause unsound consequences, the non-deterministically derived facts are usually simply discarded and re-derived if necessary. We address this and present a technique that allows for identifying non-deterministic facts that can safely be re-used. The approach is based on a set of conditions that can be checked locally and, thus, allows for efficiently identifying individuals step-by-step for which non-deterministic consequences have to be re-considered in subsequent tests. The presented technique can directly be integrated into existing tableau-based reasoning systems without significant adaptations and reduces reasoning effort for all tasks for which consequences from the ABox are potentially relevant. Moreover,

* The author acknowledges the support of the doctoral scholarship under the Postgraduate Scholarships Act of the Land of Baden-Wuerttemberg (LGFG).

it can directly be used for the DL $\mathcal{SROIQ}$, does not produce a significant overhead, and can easily be extended, e.g., to improve incremental ABox reasoning. Note that completion graph caching is complementary to many other optimisations that try to reduce the number of subsequent tests for higher level reasoning tasks, e.g., summarisation [2], abstraction and refinement [5], bulk processing and binary retrieval [6], (pseudo) model merging [7], extraction of known/possible instances from model abstractions [15].

The paper is organised as follows: We next introduce some preliminaries and then present the basic completion graph caching technique in Section 3. In Section 4, we present several extensions and applications to support nominals in ordinary satisfiability caching, to perform incremental reasoning for changing ABoxes, and to handle very large ABoxes by storing data in a representative way. Finally, we present an evaluation of the presented techniques in Section 5 and conclude in Section 6. Further details and an extended evaluation are available in a technical report [19].

2 Preliminaries

For brevity, we do not introduce DLs (see, e.g., [1,10]). We use (possibly with subscripts) C, D for (possibly complex) concepts, A, B for atomic concepts, and r, s for roles. We assume that a $\mathcal{SROIQ}$ knowledge base $\mathcal{K}$ is expressed as the union of a TBox $\mathcal{T}$ consisting of GCIs of the form $C \sqsubseteq D$, a role hierarchy $\mathcal{R}$, and an ABox $\mathcal{A}$, such that the effects of complex roles are implicitly encoded (e.g., based on automata [12] or regular expressions [16]). We write $\mathsf{nnf}(C)$ to denote the negation normal form of C. For r a role, we set $\mathsf{inv}(r) := r^-$ and $\mathsf{inv}(r^-) := r$.

Model construction calculi such as tableau [10,13] decide the consistency of a KB $\mathcal{K}$ by trying to construct an abstraction of a model for $\mathcal{K}$, a so-called *completion graph*. For the purpose of this paper, we use a tuple of the form $(V, E, \mathcal{L}, \dot{\neq}, \mathcal{M})$ for a completion graph G, where each node $x \in V$ (edge $\langle x, y \rangle \in E$) represents one or more (pairs of) individuals. Each node x (edge $\langle x, y \rangle$) is labelled with a set of concepts (roles), $\mathcal{L}(x)$ ($\mathcal{L}(\langle x, y \rangle)$), which the (pairs of) individuals represented by x ($\langle x, y \rangle$) are instances of. The relation $\dot{\neq}$ records inequalities, which must hold between nodes, e.g., due to at-least cardinality restrictions, and the mapping $\mathcal{M}$ tracks merging activities, e.g., due to at-most cardinality restrictions or nominals. The algorithm works by decomposing concepts in the completion graph with a set of expansion rules. For example, if $C_1 \sqcup C_2 \in \mathcal{L}(v)$ for some node v, but neither $C_1 \in \mathcal{L}(v)$ nor $C_2 \in \mathcal{L}(v)$, then the rule for handling disjunctions non-deterministically adds one of the disjuncts to $\mathcal{L}(v)$. Similarly, if $\exists r.C \in \mathcal{L}(v)$, but v does not have an r-successor with C in its label, then the algorithm expands the completion graph by adding the required successor node. If a node v is merged into a node v', $\mathcal{M}$ is extended by $v \mapsto v'$. We use $\mathsf{mergedTo}^{\mathcal{M}}(v)$ to return the node into which a node v has been merged, i.e., $\mathsf{mergedTo}^{\mathcal{M}}(v) = \mathsf{mergedTo}^{\mathcal{M}}(w)$ for $v \mapsto w \in \mathcal{M}$ and $\mathsf{mergedTo}^{\mathcal{M}}(v) = v$ otherwise.

Unrestricted application of rules for existential or at-least cardinality restrictions can lead to the introduction of infinitely many new nodes. To guarantee termination, a cycle detection technique called *(pairwise) blocking* [11] restricts the application of these rules. The rules are repeatedly applied until either the completion graph is fully expanded (no more rules are applicable), in which case the graph can be used to con-

struct a model that *witnesses* the consistency of $\mathcal{K}$, or an obvious contradiction (called a *clash*) is discovered (e.g., both C and $\neg C$ in a node label), proving that the completion graph does not correspond to a model. A knowledge base $\mathcal{K}$ is *consistent* if the rules can be applied such that they build a fully expanded and clash-free completion graph.

3 Completion Graph Caching and Reusing

In the following, we use superscripts to distinguish different versions of completion graphs. We denote with $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ the last completion graph of the initial consistency test that is obtained with only deterministic rule applications and with $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ the fully expanded (and clash-free) completion graph that possibly also contains non-deterministic choices. Obviously, instead of starting with the initial completion graph G^0 to which no rule has (yet) been applied, we can use G^d to initialise a completion graph G for subsequent consistency tests which are, for example, required to prove or refute assumptions of higher level reasoning tasks. To be more precise, we extend G^d to G by the new individuals or by additional assertions to original individuals as required for a subsequent test and then we can apply the tableau expansion rules to G. Note, in order to be able to distinguish the nodes/nominals in the different completion graphs, we assume that all nodes/nominals that are newly created for G do not occur in existing completion graphs, such as G^d or G^n.

This re-use of G^d is an obvious and straightforward optimisation and it is already used by many state-of-the-art reasoning systems to successfully reduce the work that has to be repeated in subsequent consistency tests [17]. Especially if the knowledge base is deterministic, the tableau expansion rules only have to be applied for the newly added assertions in G. In principle, also G^n can be re-used instead of G^d [17], but this causes problems if non-deterministically derived facts of G^n are involved in new clashes. In particular, it is required to do backtracking in such cases, i.e., we have to jump back to the last version of the initial completion graph that does not contain the consequences of the last non-deterministic decision that is involved in the clash. Then, we have to continue the processing by choosing another alternative. Obviously, if we have to jump back to a very early version of the completion graph, then potentially many non-deterministic decisions must be re-processed. Moreover, after jumping back, we also have to add and re-process the newly added individuals and/or assertions.

To improve the handling of non-deterministic knowledge bases, our approach uses criteria to check whether nodes in G (or the nodes in a completion graph G' obtained from G by further rule applications) are "cached", i.e., there exist corresponding nodes in the cached completion graph of the initial consistency test and, therefore, it is not required to process them again. These caching criteria check whether the expansion of nodes is possible as in the cached completion graph G^n without influencing modified nodes in G', thus, only the processing of new and modified nodes is required.

Definition 1 (Caching Criteria). *Let $G^d = (V^d, E^d, \mathcal{L}^d, \dot{\neq}^d, \mathcal{M}^d)$ be a completion graph with only deterministically derived consequences and $G^n = (V^n, E^n, \mathcal{L}^n, \dot{\neq}^n, \mathcal{M}^n)$ a fully expanded and clash-free expansion of G^d. Moreover, let G be an extension of G^d and $G' = (V', E', \mathcal{L}', \dot{\neq}', \mathcal{M}')$ a completion graph that is obtained from G by applying tableau expansion rules.*

A node $v' \in V'$ is cached *in G' if caching of the node is not invalid, where the* caching is invalid *(we then also refer to the node as non-cached) if*

C1 $v' \notin V^d$ *or* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') \notin V^n$*;*

C2 $\mathcal{L}'(v') \nsubseteq \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$*;*

C3 $\forall r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and there is an r-neighbour node w' of v' such that w' is not cached and $C \notin \mathcal{L}(w')$;*

C4 $\leqslant m\, r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and the number of the non-cached r-neighbour nodes of v' without* $\mathsf{nnf}(\neg C)$ *in their labels together with the r-neighbours in G^n of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *with C in their labels is greater than m;*

C5 $\exists r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and every r-neighbour node w' of v' with $C \notin \mathcal{L}(w')$ and $C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(w'))$ is not cached;*

C6 $\geqslant m\, r.C \in \mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v'))$ *and the number of r-neighbour nodes $w_1^n, \ldots, w_k^n$ of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *with $C \in \mathcal{L}^n(w_1^n), \ldots, C \in \mathcal{L}^n(w_k^n)$, for which there is either no node $w'_i \in V'$ with* $\mathsf{mergedTo}^{\mathcal{M}^n}(w'_i) = w_i^n$ *or w'_i with* $\mathsf{mergedTo}^{\mathcal{M}^n}(w'_i) = w_i^n$ *and $C \notin \mathcal{L}(w'_i)$ is not cached for $1 \leq i \leq k$, is less than m;*

C7 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *is a nominal node with $\leqslant m\, r.C$ in its label and there exists a blockable and non-cached* $\mathsf{inv}(r)$*-predecessor node w' of v' with* $\mathsf{nnf}(\neg C) \notin \mathcal{L}(w')$*;*

C8 $\mathsf{mergedTo}^{\mathcal{M}^n}(w')$ *is an r-neighbour node of* $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *such that w' is not cached and w' is not an r-neighbour node of v';*

C9 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *has a successor node u^n such that u^n or a descendant of u^n has a successor node* $\mathsf{mergedTo}^{\mathcal{M}^n}(w')$ *for which $w' \in V'$, w' is not cached, and there exists no node $u' \in V'$ with* $\mathsf{mergedTo}^{\mathcal{M}^n}(u') = u^n$*;*

C10 $\mathsf{mergedTo}^{\mathcal{M}^n}(v')$ *is blocked by* $\mathsf{mergedTo}^{\mathcal{M}^n}(w')$ *and $w' \in V'$ is non-cached;*

C11 *there is a non-cached node $w' \in V'$ and* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') = \mathsf{mergedTo}^{\mathcal{M}^n}(w')$*; or*

C12 *there is a node $w' \in V'$ such that $v' \dot{\neq}' w'$ and* $\mathsf{mergedTo}^{\mathcal{M}^n}(v') = \mathsf{mergedTo}^{\mathcal{M}^n}(w')$.

Conditions C1 and C2 ensure that a node also exists in the cached completion graph G^n and that its label is a subset of the corresponding label in G^n such that the same expansion is possible. C3 checks whether the expansion of a node would add a concept of the form $\forall r.C$ such that it could propagate C to a non-cached neighbour node. Analogously, C4 checks for potentially violated at-most cardinality restrictions by counting the new or modified neighbours in G' and the neighbours in G^n. C5 and C6 verify that existential and at-least cardinality restrictions are still satisfied if the cached nodes are expanded identically. C7 checks whether the NN-rule of the tableau algorithm would be applicable after the expansion, i.e., we check whether all potentially relevant neighbours in G' are nominal nodes. C8 checks whether the expansion would add additional roles to edge labels between cached and non-cached nodes, which could be problematic for disjoint roles. For C9: If a node w', for which caching is invalid, is connected to nodes in G^n that are only available in G^n (e.g., non-deterministically created ones), then we have to identify caching as invalid for those ancestors of these nodes that are also in G' such that these connections to w' can be re-built. Otherwise, we would potentially miss new consequences that could be propagated from w'. C10 is used to reactivate the processing of nodes for which the caching of the blocker node is invalid. C11 and C12 ensure that merging is possible as in G^n: C11 checks whether the node into which the node is merged is also cached and C12 ensures that there is no additional entry for $\dot{\neq}'$ that would cause a clash if the nodes were merged as in G^n.

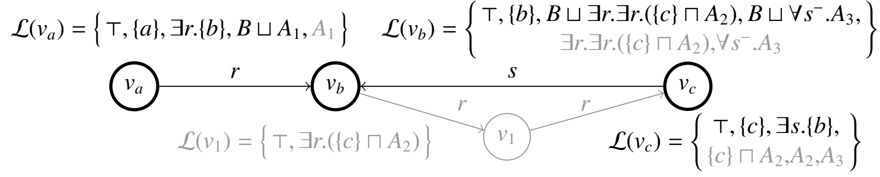

Fig. 1. Constructed and cached completion graph for Example 1 with deterministically (coloured black) and non-deterministically (coloured grey) derived facts

Since only those nodes can be cached, which are available in the "deterministic" completion graph G^d, it is important to maximise the deterministic processing of the completion graph. This can, for example, be achieved by processing the completion graph with deterministic rules only until the generation of new nodes is subset blocked. Subset blocking is not sufficient for more expressive DLs, but it prevents the expansion of too many successor nodes in case non-deterministic rule applications merge and prune some parts of the completion graph. Absorption techniques (e.g., [14,18,21]) are also essential since they reduce the overall non-determinism in an ontology.

Example 1. Assume that the tableau algorithm builds a completion graph as depicted in Figure 1 for testing the consistency of a knowledge base $\mathcal{K}$ containing the axioms

$$\{a\} \sqsubseteq B \sqcup A_1 \qquad \{b\} \sqsubseteq B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2) \qquad \{c\} \sqsubseteq \exists s.\{b\}$$
$$\{a\} \sqsubseteq \exists r.\{b\} \qquad \{b\} \sqsubseteq B \sqcup \forall s^-.A_3.$$

The deterministic version of the completion graph, G^d, contains the elements coloured in black in Figure 1 and the non-deterministic version, G^n, additionally contains the elements coloured in grey. If we want to determine which individuals are instances of the concept $\exists r.\top$, then we have to check, for each individual i in $\mathcal{K}$, the consistency of $\mathcal{K}$ extended by $\mathsf{nnf}(\neg\exists r.\top)(i)$, which is equivalent to adding the axiom $\{i\} \sqsubseteq \forall r.\bot$. The individual a is obviously an instance of this concept, which can also be observed by the fact that expanding the extended completion graph adds $\forall r.\bot$ to $\mathcal{L}^d(v_a)$, which immediately results in a clash. In contrast, if we extend $\mathcal{L}^d(v_b)$ by $\forall r.\bot$, we have to process the disjunctions $B \sqcup \exists r.\exists r.(\{c\} \sqcap A_2)$ and $B \sqcup \forall s^-.A_3$. The disjunct $\exists r.\exists r.(\{c\} \sqcap A_2)$ would result in a clash and, therefore, we choose B, which also satisfies the second disjunction. Note that v_a and v_c do not have to be processed since v_a is cached, i.e., its expansion is possible in the same way as in G^n, and v_c does not have any concepts for which processing is required. Last but not least, we have to test whether $\mathcal{L}^d(v_c)$ extended by $\forall r.\bot$ is satisfiable. Now, the caching of v_c is obviously invalid (due to C2) and, therefore, also the caching of v_b is invalid: C3 can be applied for $\forall s^-.A_3$ and C9 for the successor node v_1 of v_b in G^n which also has the non-cached successor node v_c. Since v_a is still cached, we only have to process v_b again, which does, however, not result in a clash. Hence, only a is an instance of $\exists r.\top$.

Most conditions can be checked locally: For a non-cached node, we simply follow its edges w.r.t. G' and G^n to (potentially indirect) neighbour nodes that are also available in G^d. Exceptions are Conditions C10, C11, and C12, for which we can, however, simply trace back established blocking relations or $\mathcal{M}^n$ to find nodes for which the caching

criteria have to be checked. The implementation can also be simplified by keeping the caching of a node invalid once it has been identified as invalid (even if the caching becomes possible again after the node has been expanded identically). Instead of directly checking the caching criteria, the relevant nodes can also be stored in a set/queue to check the conditions when it is (more) convenient. Clearly, it depends on the ontology, whether it is more beneficial to test the caching criteria earlier or later. If the criteria are tested early, then we could unnecessarily re-process some parts of the cached completion graph since the application of (non-deterministic) expansion rules can satisfy some conditions. A later testing could cause a repeated re-processing of the same parts if a lot of backtracking is required and consequences of re-activated nodes are involved in clashes. Alternatively, one could learn for the entire ontology or for single nodes, whether the criteria should be checked earlier or later. Unfortunately, this cannot easily be realised for all reasoning systems since it requires that dependencies between derived facts are precisely tracked in order to identify nodes that are often involved in the creation of clashes and for which the criteria should be checked early.

It is also possible to non-deterministically re-use derived consequences from G^n, i.e., if the caching is invalid for a node v and $\mathsf{mergedTo}^{\mathcal{M}^n}(v)$ is in G^n, then we can non-deterministically add the missing concepts from $\mathcal{L}^n(\mathsf{mergedTo}^{\mathcal{M}^n}(v))$ to $\mathcal{L}(v)$. Since the resulting completion graph is very similar to the cached one, caching can often be established quickly for many nodes. Of course, if some of the non-deterministically added concepts are involved in clashes, then we potentially have to backtrack and process the alternative where this node is ordinarily processed. Another nice side effect of storing G^n is that we can use the non-deterministic decisions from G^n as an orientation in subsequent consistency tests. By prioritising the processing of the same non-deterministic alternatives as for G^n, we can potentially find a solution that is very similar to G^n without exploring much of the search space.

4 Caching Extensions and Applications

In this section, we sketch additional applications of the caching technique, which allow for supporting nominals for the satisfiability caching of node labels and for reducing the incremental reasoning effort for changing ABoxes. Furthermore, we describe an extension that allows for caching (parts of) the completion graph in a representative way, whereby an explicit representation of many nodes in the completion graph can be avoided and, therefore, the memory consumption can be reduced.

Satisfiability Caching with Nominals: Caching the satisfiability status of labels of (blockable) nodes in completion graphs is an important optimisation technique for tableau-based reasoning systems [3,4]. If one obtains a fully expanded and clash-free completion graph, then the contained node labels can be cached and, if identical labels occur in other completion graphs, then their expansion (i.e., the creation of required successor nodes) is not necessary since their satisfiability has already been proven. Of course, for more expressive DLs that include, for example, inverse roles and cardinality restrictions, we have to consider pairs of node labels as well as the edge labels between these nodes. Unfortunately, this kind of satisfiability caching does not work for DLs with nominals. In particular, connections to nominal nodes can be used to propagate

new concepts from one blockable node in the completion graph to any other blockable node, whereas for DLs without nominals, the consequences can only be propagated from or to successor nodes. Hence, the caching of node labels is not easily possible and many reasoners deactivate this kind of caching for knowledge bases with nominals.

However, in combination with completion graph caching, we re-gain the possibility to cache some labels of (blockable) nodes for knowledge bases with nominals. Roughly speaking, we first identify which nodes "depend on" which nominals, i.e., which nominal nodes are existentially quantified as a successor/descendant for a node. The labels of such nodes can then be cached together with the dependent nominals, i.e., with those nominals on which the nodes depend, if all nodes for the dependent nominals are still cached w.r.t. the initial completion graph. These cache entries can be re-used for nodes with identical labels in subsequent completion graphs as long as the completion graph caching of the nodes for the dependent nominals is not invalid. Of course, the blocked processing of nodes due to matching cache entries has to be reactivated if the completion graph caching of a node for a dependent nominal becomes invalid. Moreover, we cannot cache the labels of blockable nodes that depend on newly generated nominals since possible interactions over these nodes are not detected.

Incremental Reasoning for Changing ABoxes: Many reasoning systems re-start reasoning from scratch if a few axioms in the knowledge base have been changed. However, it is not very likely that a few changes in the ABox of a knowledge base have a huge impact on reasoning. In particular, many ABox assertions only propagate consequences to the local neighbourhood of the modified individuals and, therefore, the results of reasoning tasks such as classification are often not affected, even if nominals are used in the knowledge base. With the presented completion graph caching, we can easily track which nodes from the cached completion graph are modified in subsequent tests and for which caching is invalidated to perform higher level reasoning tasks. Hence, if the changed ABox assertions have only a known, locally limited influence, then the reasoning effort for many (higher level) reasoning tasks can be reduced by checking whether a tracked node is affected by the changes. To detect the influences of the changes, an incremental consistency check can be performed where all modified individuals and their neighbours are deterministically re-constructed step-by-step until "compatibility" with the previous deterministic completion graph is achieved, i.e., the same deterministic consequences are derived for the nodes as in the previous completion graph. The non-deterministic version of the new completion graph can then be obtained by continuing the (non-deterministic) processing of the re-constructed nodes and by using the presented completion graph caching for all remaining nodes. Hence, we can identify the influenced nodes by comparing the newly obtained completion graph with the previous one. Compared to other incremental consistency checking approaches (e.g., [9]), the re-construction of changed parts supported by the completion graph caching enables a better handling of non-determinism and does not require a memory intensive tracking of which consequences are caused by which ABox assertions. The idea of tracking those parts of a completion graph that are potentially relevant/used for the calculation of higher level reasoning tasks and comparing them with the changed parts in new completion graphs has already been proposed for answering conjunctive queries under incremental ABox updates [8], but the completion graph caching simplifies the

Table 1. Ontology metrics for selected benchmark ontologies (A stands for Axioms, C for Classes, P for Properties, I for Individuals, CA for Class Assertions, OPA for Object Property Assertions, and DPA for Data Property Assertions)

Ontology	Expressivity	#A	#C	#P	#I	#CA	#OPA	#DPA
OGSF	$\mathcal{SROIQ}(\mathcal{D})$	1,235	386	179	57	45	58	20
Wine	$\mathcal{SHOIN}(\mathcal{D})$	1,546	214	31	367	409	492	2
DOLCE	$\mathcal{SHOIN}$	1,667	209	317	42	101	36	0
OBI	$\mathcal{SROIQ}(\mathcal{D})$	28,770	3,549	152	161	273	19	1
USDA-5	$\mathcal{ALCIF}(\mathcal{D})$	1,401	30	147	1,214	1,214	12	0
COSMO	$\mathcal{SHOIN}(\mathcal{D})$	29,655	7,790	941	7,817	8,675	3,240	665
DPC1	$\mathcal{ALCIF}(\mathcal{D})$	55,020	1,920	94	28,023	15,445	39,453	0
UOBM-1	$\mathcal{SHOIN}(\mathcal{D})$	260,728	69	44	25,453	46,403	143,549	70,628

realisation of this technique and significantly reduces the overhead for identifying those parts of higher level reasoning tasks that have to be re-computed. Moreover, with the completion graph caching, also very expressive DLs such as $\mathcal{SROIQ}$ can be supported.

Representative Caching: In order to reduce the memory consumption for caching the completion graph, the technique can be adapted such that all relevant data is stored in a representative way, which allows for building "local" completion graphs for small subsets of the entire ABox until the existence of a complete completion graph considering all individuals can be guaranteed. To be more precise, if a fully expanded and clash-free completion graph is constructed for a subset of the ABox (e.g., a subset of all individuals and their assertions), then we extract and generalise information from the processed individuals and store them in a representative cache. If we then try to build a completion graph for another subset of the ABox that has some overlapping with a previously handled subset (e.g., role assertions for which edges to previous handled individuals have to be created), then we load the available data from the cache and continue the processing of the overlapping part until it is "compatible", i.e., the expansion of the remaining individuals in the cache can be guaranteed as in the previously constructed completion graphs. Of course, this only works well for knowledge bases for which there is not too much non-deterministic interaction between the separately handled ABox parts. Moreover, compared to the ordinary completion graph caching, we are clearly trading a lower memory consumption against an increased runtime since more work potentially has to be repeated to establish compatibility.

5 Implementation and Evaluation

The completion graph caching introduced in Section 3 is integrated in our reasoning system Konclude [20] and we selected several well-known benchmark ontologies (cf. Table 1) for the evaluation of the presented techniques. The evaluation was carried out on a Dell PowerEdge R420 server running with two Intel Xeon E5-2440 hexa core processors at 2.4 GHz with Hyper-Threading and 144 GB RAM under a 64bit Ubuntu 12.04.2 LTS. In order to make the evaluation independent of the number of CPU cores, we used only one worker thread for Konclude. We ignored the time spent for parsing ontologies and writing results and we used a time limit of 5 minutes, i.e., 300 seconds.

Table 2. Reasoning times for different completion graph caching techniques (in seconds)

Ontology	Prep.+ Cons.	Classification				Realisation			
		No-C	Det-C	ET-C	LT-C	No-C	Det-C	ET-C	LT-C
OGSF	0.0	3.8	1.0	0.2	0.2	0.1	0.0	0.0	0.0
Wine	0.0	49.5	29.6	0.8	0.8	49.1	25.8	0.2	0.1
OBI	0.2	65.9	19.2	1.5	1.5	2.2	2.0	0.0	0.1
DOLCE	0.0	6.7	1.1	0.2	0.2	≥ 300.0	5.2	0.1	0.1
USDA-5	4.1	0.8	1.0	1.0	0.8	≥ 300.0	38.7	20.5	20.2
COSMO	0.6	≥ 300.0	≥ 300.0	42.2	11.2	*n/a*	*n/a*	11.2	19.9
DPC1	6.5	0.1	0.2	0.1	0.1	≥ 300.0	53.3	19.4	20.5
UOBM-1	6.7	240.6	4.8	1.3	1.1	≥ 300.0	≥ 300.0	≥ 300.0	≥ 300.0

Table 2 shows the reasoning times for consistency checking (including preprocessing), classification, and realisation (in seconds) with different completion graph caching techniques integrated in Konclude. Please note that the class hierarchy is required to realise an ontology, i.e., classification is a prerequisite of realisation, and, analogously, consistency checking as well as preprocessing are prerequisites of classification. Thus, realisation cannot be performed if the time limit is already reached for classification.

If no completion graph caching is activated (No-C), then the realisation and classification can often require a large amount of time since Konclude has to re-process the entire ABox for all instance and subsumption tests (if the ontology uses nominals). For several ontologies, such as Wine and DOLCE, the caching and re-use of the deterministic completion graph from the consistency check (Det-C) already leads to significant improvements. Nevertheless, with the two variants ET-C and LT-C of the presented completion graph caching technique, where ET-C uses an "early testing" and LT-C a "late testing" of the defined caching criteria, Konclude can further reduce the reasoning times. In particular, with both completion graph caching techniques, all evaluated ontologies can easily be classified and also the realisation can be realised efficiently for all but UOBM-1. Table 2 also reveals that only for COMSO there is a remarkable difference between ET-C and LT-C, where this difference can be explained by the fact that there is often more interaction with the individuals from the ABox for instance tests than for satisfiability and subsumption tests and, therefore, ET-C can be better for realisation due to the potentially lower effort in combination with backtracking.

The effects of the different completion graph caching techniques can also be observed for the OWL DL Realisation dataset of the ORE 2014 competition,[3] which contains several ontologies with non-deterministic language features and non-trivial ABoxes. By excluding $1,331$ s spent for preprocessing and consistency checking, the accumulated classification times over the contained 200 ontologies are $3,996$ s for the version No-C, $2,503$ s for Det-C, $1,801$ s for ET-C, and $1,606$ s for LT-C. Clearly, the dataset also contains many ontologies for which completion graph caching does not seem to have a significant impact, for example, if the ontologies can be processed deterministically or the ontologies are too difficult to even perform consistency checking (which is the case for 3 ontologies). Nevertheless, our completion graph caching improves the classification time with similar performances for LT-C and ET-C. By using

[3] http://www.easychair.org/smart-program/VSL2014/ORE-index.html

Table 3. Incremental reasoning effort for different reasoning tasks on changed ABoxes

Ontology	$\frac{\lvert\Delta\rvert \cdot 100}{\lvert\mathcal{K}\rvert}$	Consistency			Classification			Realisation		
		Time [s]		changed	Time [s]		reclassified	Time [s]		recomp. individuals [%]
		$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	nodes [%]	$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	classes [%]	$\mathcal{K}$	$\mathcal{K}^{\mp\Delta}$	
USDA-5	1	3.1	0.0	0.0	0.9	–	–	18.2	0.0	1.0
USDA-5	2	3.2	0.0	0.0	0.8	–	–	14.9	0.0	2.0
USDA-5	4	3.2	0.1	0.0	0.8	–	–	17.0	0.0	3.9
COSMO	1	0.4	0.0	3.1	24.4	17.4	73.0	0.4	0.2	3.1
COSMO	2	0.4	0.1	5.5	25.9	18.0	73.0	0.5	0.3	5.6
COSMO	4	0.4	0.1	9.9	25.2	18.6	72.8	0.6	0.4	10.0
DPC1	1	5.0	0.6	1.5	0.1	–	–	29.1	14.0	10.0
DPC1	2	4.9	1.1	2.6	0.1	–	–	27.7	19.9	16.9
DPC1	4	5.0	2.0	4.4	0.1	–	–	29.3	26.7	26.5
UOBM-1	1	3.6	2.3	10.0	1.3	0.8	5.9	$\geq$ 300.0		*n/a*
UOBM-1	2	3.7	2.9	14.0	1.4	1.1	7.9	$\geq$ 300.0		*n/a*
UOBM-1	4	3.7	3.5	18.2	1.4	1.7	11.3	$\geq$ 300.0		*n/a*

the satisfiability caching extension for nominals, as presented in Section 4, the accumulated classification time can be further improved to 725 s. Similar results are also achieved for the realisation of these ontologies. By excluding the times for all prerequisites, the accumulated realisation times over all 200 ontologies are $1,740$ s for No-C, $1,498$ s for Det-C, $1,061$ s for ET-C, $1,256$ s for LT-C, and 923 s for the version where the satisfiability caching extension for nominals is additionally activated.

Incremental Reasoning Experiments: To test the incremental reasoning based on the presented completion graph caching, we used those ontologies of Table 1 that have a large amount of ABox assertions and for which Konclude still has a clearly measurable reasoning time, i.e., USDA-5, COSMO, DPC1, and UOBM-1. We simulated a changed ABox for these ontologies by randomly removing a certain amount of assertions from the ontology (denoted by $\mathcal{K}$) and by re-adding the removed assertions and removing new assertions (denoted by $\mathcal{K}^{\mp\Delta}$). For each ontology, we evaluated 10 random modifications that have 1, 2, and 4 % of the size of the ontology's ABox. For simplicity, all modifications have the same amount of removed and added assertions. The obtained results for the presented incremental reasoning approach are shown in Table 3.

For consistency, the first two columns show the (incremental) consistency checking time (in seconds) for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively, and the third column shows the percentage of the nodes in the completion graph for $\mathcal{K}$ that has been changed for the application of the modification. It can be observed that, especially for smaller modifications, the incremental consistency check often requires much less time than the initial consistency check. In particular, the individuals of USDA-5 are sparsely connected via object properties and, therefore, often only the modified individuals have to be re-built. For larger modifications, the incremental consistency checking time increases significantly for some ontologies, e.g., UOBM-1, and does almost catch up to the consistency checking time for the initial ontology. On the one hand, our incremental consistency checking approach clearly has some additional overhead due to the fact that nodes are re-built step by step until an expansion as for the initial completion graph can be guaran-

teed, but, on the other hand, our prototypical implementation has still a lot of room for improvements. For example, we currently also re-build nodes for individuals for which only new assertions have been added although it would be sufficient to simply extend the nodes of the previous deterministic completion graph by the new consequences.

For classification, the first two columns show analogously the (incremental) classification time for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively, and the third column represents the percentage of the classes for which satisfiability and subsumption tests were re-calculated. At the moment, the used/modified nodes from the cached completion graph are tracked together for all satisfiability and subsumption tests and we mark those classes for which nodes have been tracked. It can be observed that for the ontologies with nominals (e.g., UOBM-1), only a few classes have to be re-classified and, in several cases, reclassification is not required at all.

Also for realisation, the first two columns show the (incremental) realisation time for $\mathcal{K}$ and $\mathcal{K}^{\mp\Delta}$, respectively, and the last column shows the percentage of the number of individuals that are potentially affected by the changes and for which the (possible) types have to be re-computed. For this, we separately track, for each individual, the nodes of the cached completion graph that are used/modified by the instance tests.

Representative Caching Experiments: We integrated a first prototypical version of the presented representative caching in our reasoning system Konclude, which is, however, not yet compatible with all other integrated features and optimisations. As of now, the integrated representative caching is primarily used for “simple individuals” that do not have too much interaction with other individuals in the ABox. In cases where representative caching could easily cause performance deficits (e.g., through the intensive use of nominals), Konclude caches the relevant parts of such completion graphs by using the ordinary technique. Moreover, data property assertions are, at the moment, internally transformed into class assertions and, as a consequence, nodes for individuals with data property assertions can currently not be representatively cached. However, first experiments are very encouraging. For example, Homo_sapiens is a very large $\mathcal{SROIQ}$ ontology from the Oxford ontology library with $244,232$ classes, 255 object properties, and $289,236$ individuals for which Konclude requires $10,211$ MB in total to check the consistency by using representative caching, whereas $19,721$ MB are required by Konclude for consistency checking with a fully expanded completion graph. Note that a large amount ($9,875$ MB) of the required memory is used for the (unoptimised) internal representation, the data from the preprocessing, and the parsed OWL objects which are kept in memory to facilitate the handling of added/removed axioms.

6 Conclusions

We have presented a refinement of the completion graph caching technique that improves ABox reasoning also for very expressive Description Logics through a more sophisticated handling of non-deterministic consequences. In addition, we sketched extensions and applications of the caching technique, which allow for supporting nominals for the satisfiability caching of node labels, for reducing the incremental reasoning effort for changing ABoxes, and for handling very large ABoxes by storing partially processed parts of the completion graph in a representative way.

References

1. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, second edn. (2007)
2. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Scalable highly expressive reasoner (SHER). J. of of Web Semantics 7(4), 357–361 (2009)
3. Donini, F.M., Massacci, F.: EXPTIME tableaux for $\mathcal{ALC}$. J. of Artificial Intelligence 124(1), 87–138 (2000)
4. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: Hermit: An OWL 2 reasoner. J. of Automated Reasoning 53(3), 1–25 (2014)
5. Glimm, B., Kazakov, Y., Liebig, T., Tran, T.K., Vialard, V.: Abstraction refinement for ontology materialization. In: Proc. 13th Int. Semantic Web Conf. (ISWC'14). LNCS, vol. 8797, pp. 180–195. Springer (2014)
6. Haarslev, V., Möller, R.: On the scalability of description logic instance retrieval. J. of Automated Reasoning 41(2), 99–142 (2008)
7. Haarslev, V., Möller, R., Turhan, A.Y.: Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In: Proc. 1st Int. Joint Conf. on Automated Reasoning (IJCAR'01). LNCS, vol. 2083, pp. 61–75. Springer (2001)
8. Halaschek-Wiener, C., Hendler, J.: Toward expressive syndication on the web. In: Proc. 16th Int. Conf. on World Wide Web (WWW'07). ACM (2007)
9. Halaschek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: Proc. 4th Confederated Int. Conf. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE(OTM'06), LNCS, vol. 4275, pp. 722–737. Springer (2006)
10. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible $\mathcal{SROIQ}$. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 57–67. AAAI Press (2006)
11. Horrocks, I., Sattler, U.: A description logic with transitive and inverse roles and role hierarchies. J. of Logic and Computation 9(3), 385–410 (1999)
12. Horrocks, I., Sattler, U.: Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms. Artificial Intelligence 160(1), 79–104 (2004)
13. Horrocks, I., Sattler, U.: A tableau decision procedure for $\mathcal{SHOIQ}$. J. of Automated Resoning 39(3), 249–276 (2007)
14. Hudek, A.K., Weddell, G.E.: Binary absorption in tableaux-based reasoning for description logics. In: Proc. 19th Int. Workshop on Description Logics (DL'06). vol. 189. CEUR (2006)
15. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. of Artificial Intelligence Research 48, 253–303 (2013)
16. Simančík, F.: Elimination of complex RIAs without automata. In: Proc. 25th Int. Workshop on Description Logics (DL'12). CEUR Workshop Proceedings, vol. 846. CEUR-WS.org (2012)
17. Sirin, E., Cuenca Grau, B., Parsia, B.: From wine to water: Optimizing description logic reasoning for nominals. In: Proc. 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'06). pp. 90–99. AAAI Press (2006)
18. Steigmiller, A., Glimm, B., Liebig, T.: Optimised absorption for expressive description logics. In: Proc. 27th Int. Workshop on Description Logics (DL'14). vol. 1193. CEUR (2014)
19. Steigmiller, A., Glimm, B., Liebig, T.: Completion graph caching extensions and applications for expressive description logics. Tech. rep., Ulm University, Ulm, Germany (2015), available online at `https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.090/Publikationen/2015/DL2015CGCTR.pdf`

20. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: system description. J. of Web Semantics 27(1) (2014)
21. Tsarkov, D., Horrocks, I.: Efficient reasoning with range and domain constraints. In: Proc. 17th Int. Workshop on Description Logics (DL'04). vol. 104. CEUR (2004)
22. Wandelt, S., Möller, R.: Towards ABox modularization of semi-expressive description logics. Applied Ontology 7(2), 133–167 (2012)
23. Wu, J., Hudek, A.K., Toman, D., Weddell, G.E.: Absorption for ABoxes. J. of Automated Reasoning 53(3), 215–243 (2014)