Partitioning OWL Knowledge Bases - Revisited and Revised

Sebastian Wandelt[1]

Institute for Software Systems,
TU Hamburg-Harburg,
wandelt@tuhh.de

Abstract. The development of scalable reasoning systems is one of the crucial factors determining the success of Semantic Web systems. Recently, in [GH06], an approach is proposed, which tackles the problem by splitting the assertional part of ontologies into several partitions.
In this work we provide our experiences gained by implementing and understanding the given partitioning algorithm and fix some issues which came our way. Furthermore, we propose an extension of the algorithm, which allows for assertional updates, without need to repartition the whole knowledge base. Both contributions can hopefully increase the potential success of partitioning real world ontologies..

1 Introduction

In [GH06], an approach for partitioning large OWL ABoxes with respect to an OWL TBox is given. The idea is that reasoning can be performed on each partition and the results can be combined in a particular way to obtain complete answers. The authors provide a partitioning algorithm for ontologies of expressivity SHIF, which roughly corresponds to OWL Lite. The idea to partition large ontologies into smaller parts, which hopefully fit into main memory, is promising and innovative.

Tempted by the nice results reported in [GH06], we attempted to implement the partitioning approach for further testing of real world ontologies. During our implementation, we faced several problems, which could be of interest for other possible implementators and people who are interested in efficient and scalable reasoning on description logics. This work reports our results in detail.

There were two kinds of problems, which we had to deal with during the implementation. First, some of definitions/algorithms yield incomplete partitions, i.e. after the partitioning is performed, information is lost. Second, there are (real world) cases, which will dramatically increase the average size of partitions. We propose alternatives to overcome these problems. Furthermore, we noticed that the given partitioning algorithm is subject to a, more or less, straight-forward extension for assertional updates. After an update of the assertional part of the ontology, our extension only recomputes the partitions, which were really changed. We think that this makes the partitioning idea even more interesting for practical settings.

This paper is structured as follows. Section 2 presents summarizing notions for description logics. In Section 3 we provide relevant notions and main results from [GH06].

We summarize our observations for the original partitioning algorithm in Section 4 and propose a revised version including assertional updates in Section 5. In Section 6 we give an example, which should make our extension easy to understand. We conclude the paper in Section 7 and also point at some ideas for further work.

2 Foundations: Description Logics

In the following, we briefly recall syntax and semantics of the description logic SHIF. For the details, please refer to [BCM$^+$07]. We assume a collection of disjoint sets: a set of *concept names* N_C, a set of *role names* N_{RN}, with a subset $N_{RN+} \subseteq N_{RN}$ of transitive role names, and a set of *individual names* N_I. The *set of roles* N_R is $N_{RN} \cup \{R^- | R \in N_{RN}\}$. The set of *SHIF-concepts* is given by by using the grammar:

$$\begin{aligned} C ::= & \top | \bot | D | \neg C | C_1 \sqcap C_2 | C_1 \sqcup C_2 | \forall R.C \\ & | \exists R.C | \leq_1 S | \geq_2 S, \end{aligned}$$

where $D \in N_C$, $R \in N_R$, $S \in N_R$ (but not transitive) and $n \in \mathbb{N}$. We assume the standard Tarski-style semantics with interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \bullet^{\mathcal{I}})$. A *knowledge base* $\mathcal{KB}$ consists of a 3-tuple $(\mathcal{T}, \mathcal{R}, \mathcal{A})$, where $\mathcal{T}$ is a set of terminological axioms, $\mathcal{R}$ is a set of role axioms and $\mathcal{A}$ is a set of assertional axioms.

We define a special kind of partitioning based on explicit role assertions between individuals in the ABox. Two individuals a and b belong to the same partition, if they are connected via a path of role assertions, i.e. $\exists x_1, x_2, ...x_n$ such that we have $R_1(a, x_1) \in \mathcal{A}$, $R_2(x_1, x_2) \in \mathcal{A}$, ..., $R_{n-1}(x_n, b) \in \mathcal{A}$. We call this partitioning method *partitioning based on ABox connectedness*. To the best of our knowledge this kind of partitioning is already implemented in state-of-the-art reasoner.

3 Related work

To make our work self-contained, in the following we provide a summary of the definitions and results from [GH06]. Please note that we do not intend to fully recap the whole paper. For detailed explanations and examples refer to the original work.

Definition 1. *[GH06] Given an OWL knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, a* independent ABox partitioning *of $\mathcal{A}$, is a partitioning $\cup_i \mathcal{A}_i$, such that for every concept/role assertion ϕ, with $\mathcal{K} \models \phi$, there exists an $\mathcal{A}_i$, where we have $\langle \mathcal{T}, \mathcal{A}_i \rangle \models \phi$*

Based on a Sequent Calculus inspired approach, the authors derive a set of complete inference rules for the description logic SHIF. For brevity the list of rules are omitted here. The basic idea for completeness, but possible unsoundness, is to weaken the left side of the original inference rules in [RQ93]. For this purpose the authors introduce three different notions:

Definition 2. *[GH06] $\forall$-`Possible`$(\mathcal{T}, R)$ is true if a concept $\forall S.C$ occurs on the right hand side of a general concept inclusion (GCI) in $\mathcal{T}$ and $R \sqsubseteq S$.*

```
**Function** BUILD_CHUNK_GRAPH($KB$)
**Parameter**: Knowledge base $KB = \langle \mathcal{T}, \mathcal{A} \rangle$
**Returns**: Chunk graph $G$ for $\mathcal{A}$
**Algorithm:**
    Precompute role relationsships and $\forall$-Possible, $\exists$-useful, $\leq_1$-Possible
    Update $G$ by applying chunk rules 1 and 2
    **Repeat**
        1. Update $G$ by applying chunk rule 4
        2. Update $G$ and $eq$ by applying chunk rule 3
    **until** neither $G$ nor $eq$ has been changed
    Update $G$ by applying chunk rules 5 and 6
    **return** $G$
```

Fig. 1. Building a chunk graph

Informally speaking, the rationale of the $\forall$-`Possible` constraint is as follows: if a role R is $\forall$-Possible then one has to take role assertions $R(a_i, b)$ into account when reasoning over individual b.

Definition 3. *[GH06] $\exists$-`useful`$(\mathcal{T}, R)$ is true if one of the following is true:*

1. *a concept $\exists S.C$ occurs on the left hand side of some (GCI) in $\mathcal{T}$ and $R \sqsubseteq S$*
2. *there exists S, s.t. $\forall$-Possible$(\mathcal{T}, R)$ is true and $R \sqsubseteq S$*

The rationale of the $\exists$-`useful` constraint is as follows: if a role is $\exists$-Useful then one has to take all role assertions $R(a, b_i)$ into account when reasoning over individual a.

Definition 4. *[GH06] $\leq_1$-`Possible`$(\mathcal{T}, R)$ is true if a concept $\leq_1 S$ occurs on the right hand side of a general concept inclusion (GCI) in $\mathcal{T}$ and $R \sqsubseteq S$.*

The rationale of the $\leq_1$-`Possible` constraint is as follows: if a role is $\leq_1$-`Possible` then one has to put all role assertions $R(a, b_i)$ and all concept assertions $b_i : C$ into the same partition. Using the above notions a chunk graph is constructed, which represents the dependencies between chunks (sets of ABox assertions). For completeness, we repeat the notions of a chunk graph and the chunk rules defined in [GH06].

Definition 5. *[GH06] A chunk graph G=(V,E) for an ABox $\mathcal{A}$ is as follows: 1) Each vertex of G represents a chunk, which is a subset of $\mathcal{A}$; chunks on G are disjoint. and their union is $\mathcal{A}$. 2) G is a directed graph and $(ck_1, ck_2) \in E$ means that a partition P containing chunk ck_2 must also contain chunk ck_1.*

The list of chunk rules is summarized as follows (for details and explanations refer to [GH06]):

Chunk rule 1: Create a chunk for each individual a in $\mathcal{A}$ containing $\{a : C\}_{a:C \in \mathcal{A}}$. Let $chunk(a)$ denote the chunk that holds the concept assertions of a.

Chunk rule 2: Create a chunk for each role assertion ϕ in $\mathcal{A}$. Let $chunk(\phi)$ denote the chunk that $contains(\phi)$.

Chunk rule 3: If $\leq_1$-`Possible`$(\mathcal{T}, R)$ and $ck_1 \models \langle a, b_1 \rangle : R$, $ck_2 \models \langle a, b_2 \rangle : R$ then

1. Merge ck_1 and ck_2 to ck

```
Function CONSTRUCT_PARTITIONS($G$)
Parameter: Chunk graph $G$ for $KB = \langle \mathcal{T}, \mathcal{A} \rangle$
Returns: independent partitioning $P$ of $\mathcal{A}$
Algorithm:
    1. Compact $G$ by merging every set of chunks that form a strongly connected component
    2. For every chunk $ck$ that has no outgoing links:
        (a) Create a partition $p = \bigcup_i ck_i$, where $ck$ is reachable from $ck_i$ on $G$ (including $ck$)
        (b) $P = P \cup \{p\}$
    3. return $P$
```

Fig. 2. Constructing relevant partitions

2. Merge $chunk(b_1)$ and $chunk(b_2)$ to ck'
3. Draw an arc from $chunk(a)$ to ck'
4. Draw an arc from ck to ck'
5. For every role assertion ϕ involving b_1 or b_2, draw an arc from ck to $chunk(\phi)$
6. Record the information $b_1 \approx b_2$

Chunk rule 4: For each $R \in R_+$, conduct an individual names based merging on the a set of assertions $\{\langle a, b\rangle : R_1 | R_1 \sqsubseteq R \texttt{ or } R_1 = R^-\}$ according to the following principle: For any two of the above assertions $phi_1 = \langle a, b\rangle : R_1$ and $phi_2 = \langle c, d\rangle : R_2$, where $a \approx b$ or $a \approx d$ or $b \approx c$ or $b \approx d$, merge $chunk(\phi_1)$ and $chunk(\phi_2)$

Chunk rule 5: If $\forall\texttt{-Possible}(\mathcal{T}, R)$ and $ck \models \langle a, b\rangle R$ then

1. Draw an arc from $chunk(a)$ to $chunk(b)$
2. Draw an arc from ck to $chunk(b)$

Chunk rule 6: If $\exists\texttt{-useful}(\mathcal{T}, R)$ and $ck \models \langle a, b\rangle R$ then

1. Draw an arc from $chunk(b)$ to $chunk(a)$
2. Draw an arc from ck to $chunk(a)$

Using the chunk rules, the general algorithm for building a chunk graph is shown in Figure 1. Finally, Figure 2 shows how to build an independent ABox-partitioning, given a chunk graph as input.

4 Role Partitioning - Revisited

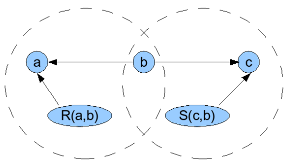
In the following section we list some observations which we made, while trying to understand and implement the proposed role partitioning algorithm in [GH06]. Each observation is accompanied by a detailed description and suggestions how to fix possible problems.

Observation 1 *The constraints $\forall\texttt{-Possible}$ and $\exists\texttt{-useful}$ are in fact equivalent to each other. That is, both constraints yield the same dependencies between assertions in an ontology ABox. Moreover, the different handling of both constraints can make the partitioning too fine and thus incomplete.*

This can be shown by a simple example knowledge base $KB_1 = \langle \mathcal{T}, \mathcal{R}, \mathcal{A} \rangle$, such that

$$\begin{aligned}
\mathcal{T} &= \{\exists R.\neg D \sqsubseteq \bot, \exists S.\neg E \sqsubseteq \bot, D \sqcap E \sqsubseteq F\} \\
\mathcal{R} &= \{\} \\
\mathcal{A} &= \{a : A, b : B, c : C, R(a,b), S(b,c)\}
\end{aligned}$$

Please note that the roles R and S are $\exists$-`useful`, but not $\forall$-`Possible`, by definition. The resulting chunk graph G and the final partitions (marked by fine-dashed circles), which are obtained by the function $CONSTRUCT\_PARTITIONS(G)$, are shown in the picture below:



The algorithm creates two distinct partitions: $P_1 = \{a : A, b : B, R(a,b)\}$ and $P_2 = \{b : B, c : C, S(c,b)\}$. It is easy to see that the original knowledge base KB_1 entails the fact $b : F$. However, none of the two computed partitions entails $b : F$. To infer $b : F$, one needs to have $R(a,b)$ and $S(c,b)$ in the same partition with $b : B$. Thus the chunk rules for $\exists$-`useful` are not yet sufficient to handle all cases in a complete way. Please note that the first two TBox-constraints can also be rewritten as $\forall$-constraints. The resulting (equivalent) TBox is

$$\mathcal{T}_b = \{\top \sqsubseteq \forall R.D, \top \sqsubseteq \forall R.E, D \sqcap E \sqsubseteq F\}.$$

It is interesting to note that the knowledge base $KB_{1b} = \langle \mathcal{T}_b, \mathcal{R}, \mathcal{A} \rangle$ (which is semantically equivalent to KB_1) is partitioned correctly. The example shows that $\exists$-constraints are not sufficiently considered, but $\forall$-`Possible`-constraints are. We fix that problem by using the following definition:

Definition 6. *$\forall_{ext}$-`Possible`$(\mathcal{T}, R)$ is true if there exists a concept inclusion $C \sqsubseteq D$ in $\mathcal{T}$, such that a concept expression $\forall S.X$ occurs in $nnf(\neg C \vee D)$, s.t. $R \sqsubseteq S$ and X is an arbitrary concept expression.*

The idea is indeed to treat $\forall$-`Possible` and $\exists$-`useful` equivalently, because each TBox-inclusion can be seen as a disjunction. And within the corresponding disjunction, roles being $\forall$-`Possible` and $\exists$-`useful` are indistinguishable. With respect to the chunk rules, we need to apply both rules (Chunk rule 5 and Chunk rule 6) whenever a role is $\forall_{ext}$-`Possible`. That is, whenever two individuals are related by a role R, s.t. $\forall_{ext}$-`Possible`$(\mathcal{T}, R)$ is true, we need to create a dependency between the role assertion and *both* individuals. For the details of these dependencies see below.

Observation 2 *The crucial point of the partitioning algorithm is the number of roles, which are $\forall$-`Possible`, $\exists$-`useful` and $\leq_1$-`Possible`. It is easy to show: if for each role R in an ontology we can find that R is either $\forall$-`Possible`, $\exists$-`useful` or*

$\leq_1$-`Possible`, then the whole partitioning algorithm degrades to ABox-connectedness partitions. On the other hand, if only few of the existing roles have these properties, then the partitioning algorithm works best. As we show below, range- and domain-restrictions play a crucial role here.

We conducted some research on public available ontologies to determine, how many roles of an *average* (randomly chosen) ontology are actually $\forall$-`Possible` and $\exists$-`useful`. The result is shown in Figure 3. For detailed information on the chosen ontologies please refer to the given references. One reason for the high amount of $\forall$-`Possible`-

Ontology	**Nr. of roles**	$\forall$-`Possible`	$\exists$-`useful`	$\leq_1$-`Possible`
LUBM [GPH05]	25	24	24	0
Galen [RRS+01]	413	136	136	150
Wine [W3C03]	13	10	10	7
GO-Daily [ABB+00]	1	0	0	0
Security [AH07]	28	27	27	7

Fig. 3. Role property statistics for common ontologies

roles are OWL range-restrictions. OWL enables the user to define that all the targets of a role are of a particular type. The TBox-equivalent for a range-restriction is $\top \sqsubseteq \forall R.C$. Thus, each role, which has an associated range-restriction, becomes $\forall$-`Possible`. It is the same with domain restrictions and $\exists$-`useful`. This shows, that a naive implementation of the partitioning algorithm can yield quite big partitions, much bigger than shown in [GH06]. In fact, it is possible to optimize the algorithm for domain/range-restrictions by use of the following definition in the next section.

Definition 7. *Role assertion $R(a, b)$ is* dom-relevant *for individual a, if we have a domain-restriction on a role S, s.t. $R \sqsubseteq S$. Role assertion $R(a, b)$ is* ran-relevant *for individual b, if we have a range-restriction on a role S, s.t. $R \sqsubseteq S$.*

Furthermore, we adjust the definition of $\forall_{ext}$-`Possible`$(\mathcal{T}, R)$ as follows:

Definition 8. *$\forall_{ext2}$-`Possible`$(\mathcal{T}, R)$ is true if there exists a concept inclusion $C \sqsubseteq D$ in $\mathcal{T}$, such that a concept expression $\forall S.X$ occurs in $nnf(\neg C \vee D)$ and $R \sqsubseteq S$, where X is an arbitrary concept expression* and $C \sqsubseteq D$ is not caused by a domain or range restriction.[1]

The usefulness of our extension can be seen in Figure 4. The columns $\forall$-`Possible` and $\exists$-`useful` show the number of roles without taking into account domain- and range restrictions. For some ontologies, we have a recognizable reduction of $\forall$-`Possible`-roles, especially for LUBM. As we stated earlier, the number of $\forall$-`Possible`-roles, and thus the granularity of the partitioning, is critical for the efficiency of the partitioning algorithm.

Ontology	**Nr. of roles**	$\forall$-Possible	$\exists$-useful	$\leq_1$-Possible
LUBM	25	4	4	0
Galen	413	136	136	150
Wine	13	7	7	7
GO-Daily	1	0	0	0
Security	28	17	17	7

Fig. 4. Role property statistics without domain/range-restrictions

```
Chunk rule A:
    If $\leq_1$-Possible$(\mathcal{T}, R)$ and $ck_1 \models \langle a, b_1 \rangle : R$, $ck_2 \models \langle a, b_2 \rangle : R$ then
      1. Merge $ck_1$ and $ck_2$ to $ck$
      2. Merge $chunk(b_1)$ and $chunk(b_2)$ to $ck'$
      3. Draw an arc from $chunk(a)$ to $ck'$
      4. Draw an arc from $ck$ to $ck'$
      5. For every role assertion $\phi$ involving $b_1$ or $b_2$, draw an arc from $ck$ to $chunk(\phi)$
      6. Record the information $b_1 \approx b_2$
Chunk rule B:
    For each $R \in R_+$, conduct an individual names based merging on the a set of assertions $\{\langle a, b \rangle : R_1 | R_1 \sqsubseteq R$ or $R_1 = R^-\}$ according to the following principle: For any two of the above assertions $\phi_1 = \langle a, b \rangle : R_1$ and $\phi_2 = \langle c, d \rangle : R_2$, where $a \approx b$ or $a \approx d$ or $b \approx c$ or $b \approx d$, merge $chunk(\phi_1)$ and $chunk(\phi_2)$
Chunk rule C:
    If $\forall_{ext2}$-Possible$(\mathcal{T}, R)$ and $ck \models \langle a, b \rangle : R$ then
      1. Merge $chunk(a)$ with $chunk(b)$
      2. Draw an arc from $ck$ to $chunk(a/b)$
```

Fig. 5. New chunk rules

5 Role Partitioning - Revised

In the following section we present a refinement of the partitioning algorithm in [GH06]. There are two major changes. First, we fix the issues identified in the previous section. Second, we enable a restricted form of ABox updates on the partitions without having to recompute the whole chunk graph and all partitions from the scratch. Let us redefine the relevant notions, e.g. chunks, and give our understanding of the partitioning.

A *Chunk* is a tuple $C = \langle RA, IND \rangle$, where $RA : 2^{N_R \times N_I \times N_I}$ yields the role assertions of a chunk and $IND : 2^{N_I \times N_C}$ is the set of individual assertions attached to a chunk. A *chunk graph* is a directed graph $G = \langle V, E \rangle$, s.t. $V \subseteq Chunk$ and $E \subseteq V \times V$. The rules for generating a chunk graph are given in Figure 5. As in the original work we assume that, initially, we have one chunk for each individual and one chunk for each role assertion in the initial ABox.

Next, we take into account the case of updating an ABox. The idea is as follows: For each operation on the knowledge base, e.g. adding a new assertion, we keep track of all possible partitions which might have been changed. Please note that in the original partitioning algorithm, we have one partition for each chunk c in the chunk graph, s.t. c has no outgoing edges. Thus, upon updating the chunk graph, we keep track of new chunks (candidates for new partitions) and also the changes applied to a chunk. For the rest of this section we denote with *anchor* a chunk with no outgoing edges.

[1] Yuanbo Guo, one of the authors of [GH06], has suggested a similar fix for domain-/range-restrictions upon personal inquiry

All interaction with the partitions takes place via a structure called PartitionManager. A *PartitionManager* is a tuple $PM = \langle partitions : Chunk \rightarrow 2^{Chunk}, candidates : 2^{Chunk} \rangle$, where the element $partitions$ encodes the partitions as follows: for each anchor c, $partitions(c)$ yields the chunks from which we can reach c in the chunk graph. The element $candidates$ is a set of possible anchor chunks. The partition manager defines a set of interface functions for interaction (see Figure 6). Informally speaking, these functions have the following semantics:

- setTerminologicalKnowledge($\mathcal{T}, \mathcal{R}$): initializes data structures for the partitioning algorithm
- addAssertion(ϕ): adds an ABox assertion to the partition manager
- removeAssertion(ϕ): removes an ABox assertion from the partition manager
- updatePartitions$()$: recompute necessary partitions and returns the newly created partitions (not the old ones which are still valid)
- checkEntailment($a : C$): check, whether the a partition of the knowledge base entails $a : C$

In the following we will go into the details of the two functions $updatePartitions()$ and $checkEntailment(a : C)$ only, since the other functions are self-explanatory with our observations and the descriptions in [GH06].

The idea of the function $updatePartitions()$ is to check for each candidate chunk $c \in candidates$, whether it is (still) a valid anchor of a partition after applying all the rules. First, we apply the three chunk rules to the chunk graph exhaustively. Please note that, usually, after small ABox updates, the rules will be applicable to a small subset of the existing chunks only (see example below). Since all three chunk rules can invalidate old partitions, e.g. by adding new arcs to the chunk graph, we need to keep track of the consequences of rule application w.r.t. our candidate partitions. This is done with two helper functions in Figure 7: $OnMerge$ is called whenever a chunk rule merges two chunks and $OnNewArc$ is called whenever a chunk rule adds a new dependency. After the rule application, the variable $candidates$ contains all chunks which are possibly anchors for additional partitions. Thus, we only need to check, whether it is an anchor chunk, i.e. the chunk has no outgoing edges, and then collect all the inverse reachable chunks (together with all dom-/ran-relevant role assertions). In the end we return all new partitions.

In function $checkEntailment(a : C)$, we make use of the fact that most of the partitions are unchanged after an update. While, already checking for entailment for all old (but valid) partitions, we compute the new partitions in the background. That is, we assume that both tasks are executed on different machines. Usually one wants to have multiple reasoning systems working on several partitions, anyways. We see two major advantages in this approach:

1. We have short response times for entailment checks, since we do not have to recompute the whole partitioning *before* we begin with the reasoning process
2. We can further distribute the workload between multiple machines during partitioning

```
Function setTerminologicalKnowledge($\mathcal{T}$, $\mathcal{R}$)
    1. Compute the role hierarchy graph for $\mathcal{R}$
    2. Compute $\forall_{ext}$-Possible and also $\leq_1$-Possible for each role occurring on $\mathcal{T}$
Function addAssertion($\phi$)
    1. If ($\phi = a : C$) then
        (a) If there exists a chunk $c$ for $a$ then add $a : C$ to $c$
        (b) else
            i. Create a new Chunk $c = \langle\{\}, \{a : C\}\rangle$
            ii. $candidates = candidates \cup \{c\}$
    2. else if ($\phi = R(a, b)$)
        (a) Create a new Chunk $c = \langle\{R(a, b)\}, \{\}\rangle$
        (b) $candidates = candidates \cup \{c\}$
Function removeAssertion($\phi$)
    1. If ($\phi = a : C$) then remove $C$ from individual $a$
    2. else if ($\phi = R(a, b)$)
        (a) Let $c = chunk(R(a, b)$
        (b) Remove $R(a, b)$ from $c$
        (c) If $c$ is empty then
            i. remove $c$ from the chunk graph
            ii. $candidates = candidates \cup \{c_n |$There exists an edge from $c_n$ to $c\}$
Function updatePartitions()
    1. Let $S = \emptyset$
    2. Apply Chunk rule A, B and C exhaustively
    3. For each $c \in candidates$ do
        (a) If $c$ has no outgoing roles then
            i. Let $t = \{c_n | c$ is reachable from $c_n\} \cup \{c\}$
            ii. $t = t\cup$
                $\{R(a, b) \in \mathcal{A} | R(a, b)$ is dom-relevant for $a$ and $a$ occurs in $t\}\}\cup$
                $\{R(a, b) \in \mathcal{A} | R(a, b)$ is ran-relevant for $b$ and $b$ occurs in $t\}\}$
            iii. Set $partitions(c) = t$
            iv. $S = S \cup t$
    4. return $S$
Function checkEntailment($a : C$)
    1. $P_o = range(partitions)$
    2. Start computing in parallel $P_u$ = updatePartitions()
    3. For each $p \in P_o$ do
        (a) If $p \models a : C$ then return true
    4. Wait for completion of $P_u$
    5. For each $p \in P_u$ do
        (a) If $p \models a : C$ then return true
    6. return false
```

Fig. 6. PartitionManager interface functions

6 Example

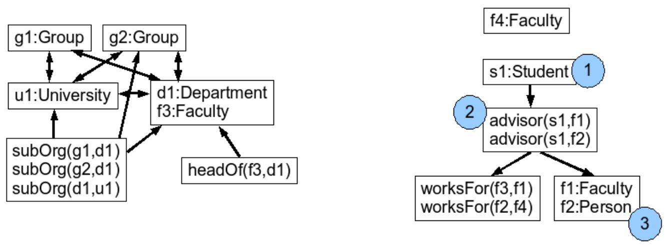
In the following we will show an example application of the extended partitioning algorithm. As example knowledge base we take the one from the original paper. Let $KB_3 = \langle\mathcal{T}_3, \mathcal{R}_3, \mathcal{A}_3\rangle$, such that

$$\begin{aligned}
\mathcal{T}_3 &= \{Student \sqsubseteq \leq_1 advisor.\top, Chair = \exists headOf.Department,\\
&\quad Department \sqsubseteq \forall subOrg.University\}\\
\mathcal{R}_3 &= \{Trans(subOrg), Trans(worksFor)\}\\
\mathcal{A}_3 &= \{u1 : University, d1 : Department, g1 : Group, g2 : Group, f1 : Faculty, f2 : Person,\\
&\quad f3 : Faculty, f4 : Faculty, s1 : Student, headOf(f3, d1), advisor(s1, f1),\\
&\quad advisor(s1, f2), worksFor(f3, f1), worksFor(f2, f4), subOrg(g1, d1),\\
&\quad subOrg(g2, d1), subOrg(d1, u1)\}
\end{aligned}$$

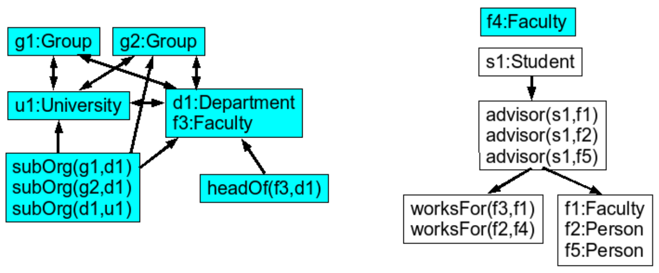
```
**Function** OnMerge(Chunk $c_1$, Chunk $c_2$)
    1. **For** all $p \in P$, such that $p$ contains either $c_1$ or $c_2$ **do**
        (a) $candidates = candidates \cup anchor(p)$
        (b) remove $p$ from $P$
    2. $candidates = candidates \cup c_1$
**Function** OnNewArc(Chunk $c_1$, Chunk $c_2$)
    1. **For** all $p \in P$, such that $p$ contains either $c_1$ or $c_2$ **do**
        (a) $candidates = candidates \cup anchor(p)$
        (b) remove $p$ from $P$
    2. $candidates = candidates \setminus \{c_1\}$
```

Fig. 7. PartitionManager helper functions

From the initial evaluation of $\mathcal{T}_3$ we can infer the facts $\forall_{ext}$-$\texttt{Possible}_2(\mathcal{T}_3, headOf)$, $\forall_{ext}$-$\texttt{Possible}_2(\mathcal{T}_3, subOrg)$ and $\leq_1$-$\texttt{Possible}(\mathcal{T}_3, advisor)$. The result of the function $redoPartitions()$, i.e. the chunk graph for $\mathcal{A}_3$, can be seen below:



There are three anchors in the chunk graph, and thus, three partitions. Assume that we want to add two ABox assertions to the Abox: $f5 : Person$ and $advisor(s1, f5)$. Executing $addAssertion(f5 : Person)$ yields an additional chunk, which is put into the set of candidates. Next we execute $addAssertion(advisor(s1, f5))$ and we obtain by chunk rule A that the chunk containing $f : Person$ is merged with chunk number 3 and $advisor(s1, f5)$ is put into chunk number 2. During these operation the partition starting at anchor chunk 1 gets invalidated. All the other partitions are untouched. The final result after repartitioning is shown below (untouched partitions are marked):



7 Conclusions and Future Work

In this work we have revisited and revised the partitioning algorithm proposed in [GH06]. Our results are twofold: First, we fix some issues with the original partitioning algorithm, which lead to incomplete partitions. Second, we propose a revised version, which allows for assertional updates of the ontology. The important point here is that we only recompute those partitions, which we really need to recompute. This does not only shorten the time needed for partitioning, but also means that we can parallelize the reasoning process and the partitioning process. We think that both kind of results can be seen as a contribution for scalable reasoning on the Semantic Web. Our test implementation of the partitioning algorithm is available for download[2] and can be investigated/used by other interested people.

For future work we intend to further investigate the applicability of partitioning techniques to more real world ontologies. However, it is hard to perform these test, without having a definition of *average real world*. Furthermore, we think that it should be possible to extend the revised partitioning algorithm from SHIF to SHIQ or maybe even further.

References

[ABB$^+$00] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet*, 25(1):25–29, May 2000.

[AH07] C. Duma A. Herzog, N. Shahmehri. An ontology of information security. *International Journal of Information Security and Privacy*, 2007.

[BCM$^+$07] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, New York, NY, USA, 2007.

[GH06] Yuanbo Guo and Jeff Heflin. A scalable approach for partitioning owl knowledge bases. In *Proc. International Workshop on Scalable Semantic Web Systems*, 2006.

[GPH05] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[RQ93] Veronique Royer and J. J Quantz. Deriving inference rules for description logics: a rewriting approach into sequent calculi. Technical report, Berlin, Germany, Germany, 1993.

[RRS$^+$01] J. Rogers, A. Roberts, D. Solomon, E. van der Haring, C. Wroe, P. Zanstra, and A. Rector. Galen ten years on: tasks and supporting tools. *Medinfo*, 10(Pt 1):256–60, 2001.

[W3C03] W3C. Owl web ontology language guide, 2003.

[2] http://www.sts.tu-harburg.de