On the Scalability of Description Logic Instance Retrieval

V. Haarslev[1], R. Moeller[2], M. Wessel[2]

[1]Concordia University, Montreal

[2]Hamburg University of Technology (TUHH)

Supported by EU Project
http://www.tonesproject.org/

Scenario

- Description logic reasoners for ontology-based information systems (OBIS)
- Ontological knowledge
  - TBoxes contain implication axioms between concepts
- Knowledge about individuals and their relations
  - ABoxes contain ABox assertions
- TBox + ABox = Knowledge Base (KB)
- Information access / retrieval
  - Classical instance retrieval queries
  - More expressive DL query languages
- We assume basic knowledge in DLs and DL reasoning techniques (tableau calculi)

Thesis

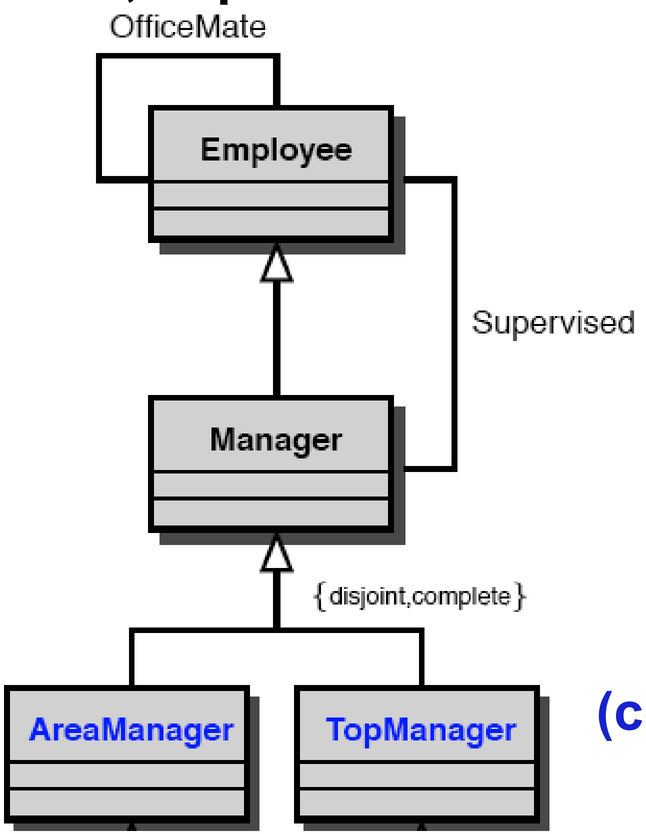
- More and more expressivity for description languages might be irresistible …
- … however, still most applications are based on “bulk data” (e.g., in rel. DBs)
- Expressivity required, but for some parts only
- DL reasoners …
  - must be good for bulk data as well (data description scalability)
    - large parts of ABoxes deterministic
  - must be expressive for special parts (expressivity scalability)
- … in order to support future OBIS

What’s different from database retrieval?

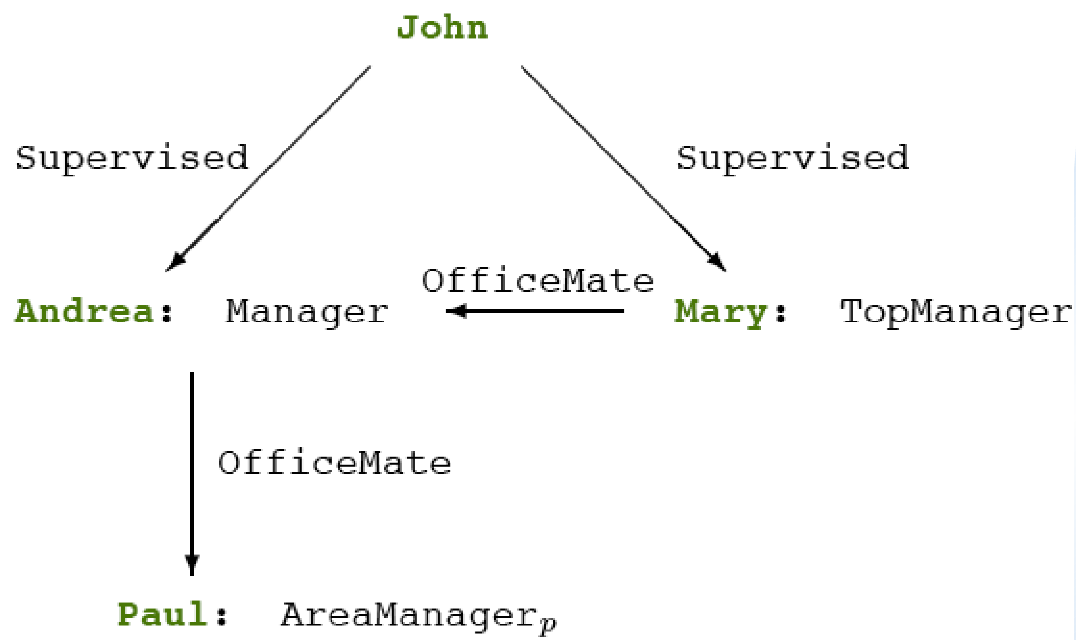
Information Society Technologies

Ontology-Based QA / Incomplete Information

TBox, depicted as class diagram



ABox, depicted as graph



(concept-instances
(some supervised
(and top-manager
(some office-mate
area-manager)))))

Instance Retrieval Queries & Conjunctive Queries (CQs)

From instance retrieval queries to CQs (-> nRQL):

```
(retrieve (?x)
      (?x (some supervised
              (and top-manager
                  (some office-mate area-manager)))))
 -> (((?x john)))  (a CQ with „Instance Retrieval Atom“)
```

A variable that appears <u>in the head</u> of a query is bound to an individual iff that binding holds in all models of the KB (is a „certain answer“)
-> ?x is a distinguished or must-bind variable

```
(retrieve (?x)
      (and (?x ?y supervised) (?y top-manager)
           (?y ?z office-mate) (?z area-manager)))
```

-> (((?x john))) with non-distinguished variables ?y, ?z

-> () with distinguished variables ?y, ?z (nRQL)

Approaches to Address the Scalability Problem

- Layered (TBox + DB)
  - Known systems, such as
    - DLDB, DL-Lite, Instance Store, LAS
  - ☑ Fast w.r.t. retrieval (due to DB)
  - – Expressivity restricted
- Integrated (TBox + ABox)
  - ☑ Expressivity
  - – Speed improvements advantageous (-> this paper / talk)

Integrated Approach

- Tableau-based approaches
  - Scalability for TBoxes empirically shown
    - Fact++, Pellet, RacerPro, others
    - Improvements always possible (in particular for less expressive languages)
  - Scalability for ABoxes on the wish list
- Other approaches
  - Disjunctive Datalog/Resolution-based
    - KAON2

Investigation

- This paper presents
  - (mainly) an investigation of optimization strategies for instance retrieval queries / atoms
- "Deterministic" KBs chosen for the investigation (next slide)
- Only if we get this part right, we will be able to adequately support OBIS application builders
- Assumption
  - ABox realization too expensive

LUBM

- LUBM = „Lehigh University Benchmark“ [Heflin et al.]
- OWL document -> DL KB is in **SH(Dn)**
- Models a university
- Benchmarking queries, e.g.

$$Q9 : ans(x, y, z) \leftarrow Student(x), Faculty(y), Course(z),$$
$$advisor(x, y), takesCourse(x, z), teacherOf(y, z)$$
$$Q12 : ans(x, y) \leftarrow Chair(x), Department(y), memberOf(x, y),$$
$$subOrganizationOf(y,' www.University0.edu')$$

LUBM TBox

- Necessary conditions for concept names
  - $A \sqsubseteq A_1 \sqcap \ldots \sqcap A_n$
- Necessary and sufficient conditions for concept names
  - $A \doteq A_1 \sqcap A_2 \sqcap \ldots \sqcap A_k \sqcap \exists R_1.B_1 \sqcap \ldots \sqcap \exists R_m.B_m$
  - $Chair \doteq Person \sqcap \exists headOf.Department$
- Moreover, transitive roles, a role hierarchy, as well as domain and range restrictions for roles are present

Basic Optimizations for Conjunctive Queries(1)

- Generators (establish variable bindings)
  - Tuple generators:
    C(x) (Instance Retrieval Atom), R(x,y)
  - Role filler generators: R(i,y), R(x,i)
- Testers (check established binding)
  - Instance tests and role tests
- Role atoms highly optimized
  - No inference required up to DL **SHI** (efficient graph traversal algorithm)

$$Q9 : ans(x, y, z) \leftarrow Student(x), Faculty(y), Course(z), advisor(x, y), takesCourse(x, z), teacherOf(y, z)$$

$$Q12 : ans(x, y) \leftarrow Chair(x), Department(y), memberOf(x, y), subOrganizationOf(y, 'www.University0.edu')$$

Basic Optimizations for Conjunctive Queries(2)

- Three well-known heuristics
  - Use low-cardinality generators first (“most constr. gen. first”)
  - Prefer filler generators over tuple generators
  - Avoid computation of more than one binding for existential variables

$$Q9: ans(x,y,z) \leftarrow Student(x), Faculty(y), Course(z),$$
$$advisor(x,y), takesCourse(x,z), teacherOf(y,z)$$
$$Q9': ans(x,y,z) \leftarrow Faculty(y), teacherOf(y,z), Course(z),$$
$$advisor^{-1}(y,x), Student(x), takesCourse(x,z)$$

ABox-Indexing for Instance Retrieval Atoms

- Exploit told information
  - Inspect ABox, analyze assertions
- Additionally exploit taxonomical information
  - Classify TBox
- Indexes usually incomplete, but complete for simple KBs (e.g. TBox is a Thesaurus)
- Used in many systems such as DLDB, Instance Store, LAS, …
- Provides „easy answers“
- Want more obvious instances?
- -> Look into the tableau completion

Tableau Provers …

- … implement a rule-based tableau calculus which decides ABox satisfiabilty
- Tableau rules are applied to the input ABox
- Rules add assertions, e.g. the AND-rule breaks up conjunctions, etc.
- Usually, one rule for each DL language constructor
- The rules are applied in a non-deterministic (but strategy-controlled way) until
  - No more rules are applicable -> a completion has been found
  - Or a contradition („Clash") has been derived
  - -> Search required
- I a completion can be derived, the ABox is satisfiable, and otherwise unsatisfiable
- A completion (finitely) represents an ABox model

Candidate Reduction (1)

- Find obvious non-instances
  - Individual „i" is a non-instance of „C" iff

    ABox' = ABox U { i : not C }

    is satisfiable
  - Computationally cheap (incomplete) test wanted for detection of obvious non-instances
  - Acquire „pseudo models" from a completion
  - ABox' is satisfiable if the so-called „pmodels" of „i" and „not C" are mergable
  - Cheap and incomplete test for satisfiability
  - Further techniques used: binary partitioning and dependency-directed partitioning, see [Haarslev&Moeller KR&R'04] for details

Candidate Reduction (2)

- Find <u>obvious</u> instances
  - Individual „i“ is an instance of „C“ iff
    ABox‘ = ABox U { i : not C }
    is not satisfiable
  - Computationally cheap (incomplete) test wanted for detection of obvious instances
  - (at least some) logically entailed (or valid) assertions must be determined from a completion for such an unsatisfiability test
    - Given a completion, identify and keep only the deterministic assertions; these are contained in every completion and are thus logically entailed = PRECOMPLETION
    - Use deterministic assertions DET(i) for computation of an approximation MSC‘ of the MSC (most specific concept) of „i“
    - Check if MSC‘(i) is subsumed by C, or
    - Check if DET(i) U DET(not C) is contradictory

Query Transformation (1)

- Insert sufficient conditions for instance retrieval atoms from the TBox

$$Q15 \quad ans(x) \leftarrow Chair(x)$$

$$Chair \doteq Person \sqcap \exists headOf.Department$$

$$ans(x) \leftarrow Person \sqcap \exists headOf.Department(x)$$

- -> Query expansion procedure (below)
- Single instance retrieval queries turn into CQs and can be optimized with the techniques just described

Query Transformation (2)

$$Q15 : ans(x) \leftarrow Person(x), headOf(x,y), Department(y)$$
$$Q15' : ans(x) \leftarrow Department(y), headOf^{-1}(y,x), Person(x)$$

- nRQL semantics for variables
  - All variables are distinguished
  - CQ is equivalent to the original instance retrieval query if precompletion = completion
  - Use also anonymous individuals (created by tableau rules) as bindings for the „fresh variables“ (here *y*)
- Reduces set of candidates for subsequent tableau-based instance retrieval proof
- -> Gives less-obvious instances

C(x) -> rewrite(tbox,C,x) (1)

Algorithm 1 $rewrite(tbox, concept, var)$:

if $meta\_constraints(tbox) \neq \emptyset \vee definition(concept) = \top$ **then**
  return $(concept(var))$
else
  $\{atom_1, \ldots, atom_n\} :=$
    $rewrite\_0(tbox, concept, definition(tbox, concept), var, \{\})$
   return $(atom_1, \ldots, atom_n)$

Algorithm 2 $rewrite\_0(tbox, concept, var, exp)$:

if $definition(concept) = \top \vee concept \in exp$ **then**
  return $\{concept(var)\}$
else
  ;; catch installs a marker to which the control flow can be thrown
  catch $not\_rewritable$
    $rewrite\_1(tbox, concept, definition(tbox, concept), var, \{concept\} \cup exp)$

C(x) -> rewrite(tbox,C,x) (2)

Algorithm 3 $rewrite\_1(tbox, concept\_name, definition, var, exp)$:

---

if $(definition = A)$ where A is an atomic concept **then**
  return $rewrite\_0(tbox, definition, var, \{definition\} \cup exp)$
else
  if $(definition = \exists R.C)$ **then**
    $filler\_var := fresh\_variable()$
    return $\{R(var, filler\_var)\} \cup rewrite\_0(tbox, C, filler\_var, exp)$
  else
    if $(definition = C_1 \sqcap \ldots \sqcap C_n)$ **then**
      return $rewrite\_1(tbox, concept\_name, C_1, var, exp)$
        $\cup \ldots \cup$
        $rewrite\_1(tbox, concept\_name, C_n, var, exp)$
    else
      ;; throw the control flow out of rewrite_1 recursion
      ;; back to the call to rewrite_1 in rewrite_0 and return {concept_name(var)}
      throw $not\_rewritable\{concept\_name(var)\}$

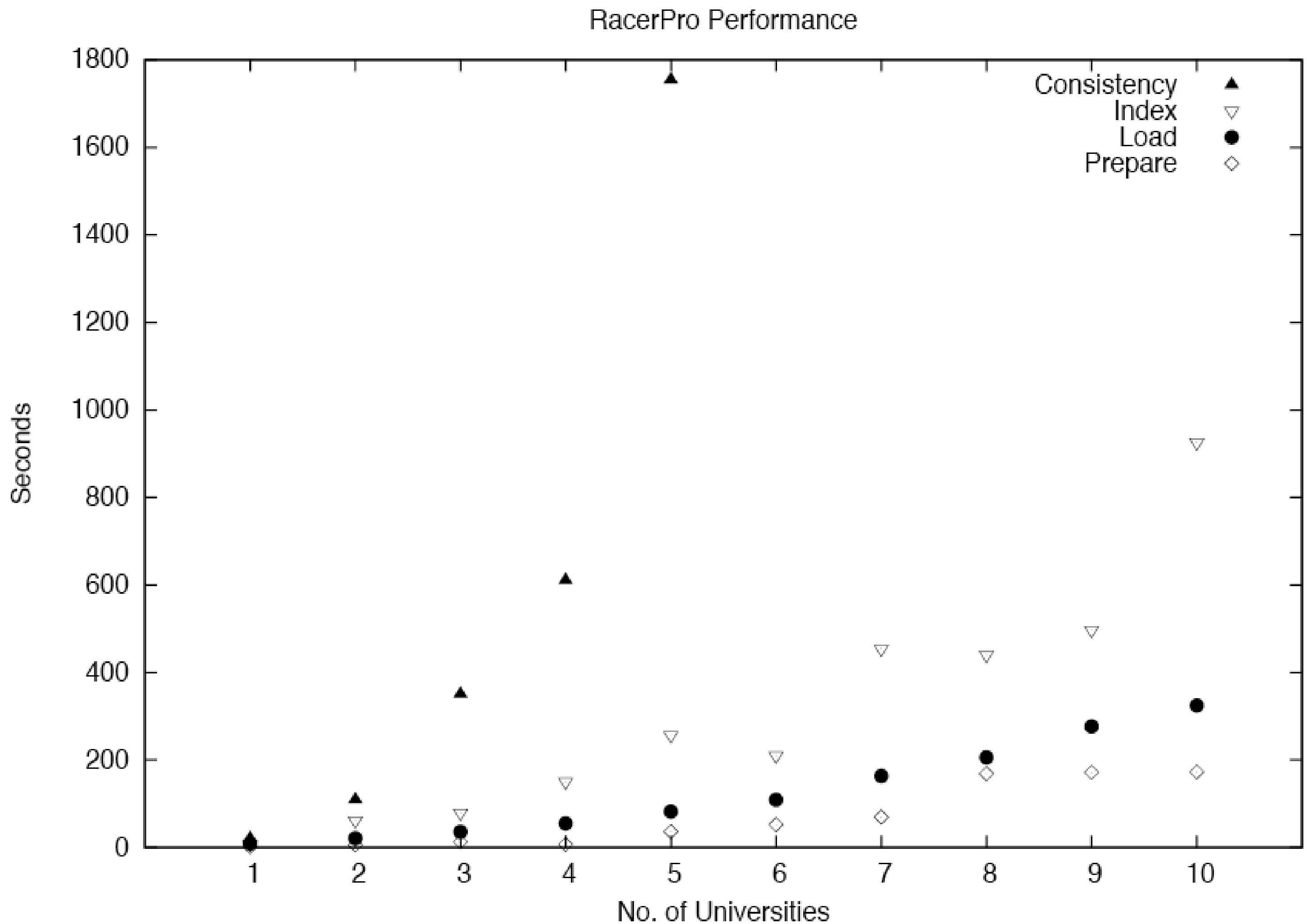
LUBM Evaluation (1)

- LUBM query set (14 queries)
- ABox sizes

Univs	Inds	Concept Assertions	Role Assertions
1	17174	53738	49336
3	55664	181324	166682
5	102368	336256	309393
10	207426	685569	630753

- Load, Consistency, Index, Prepare

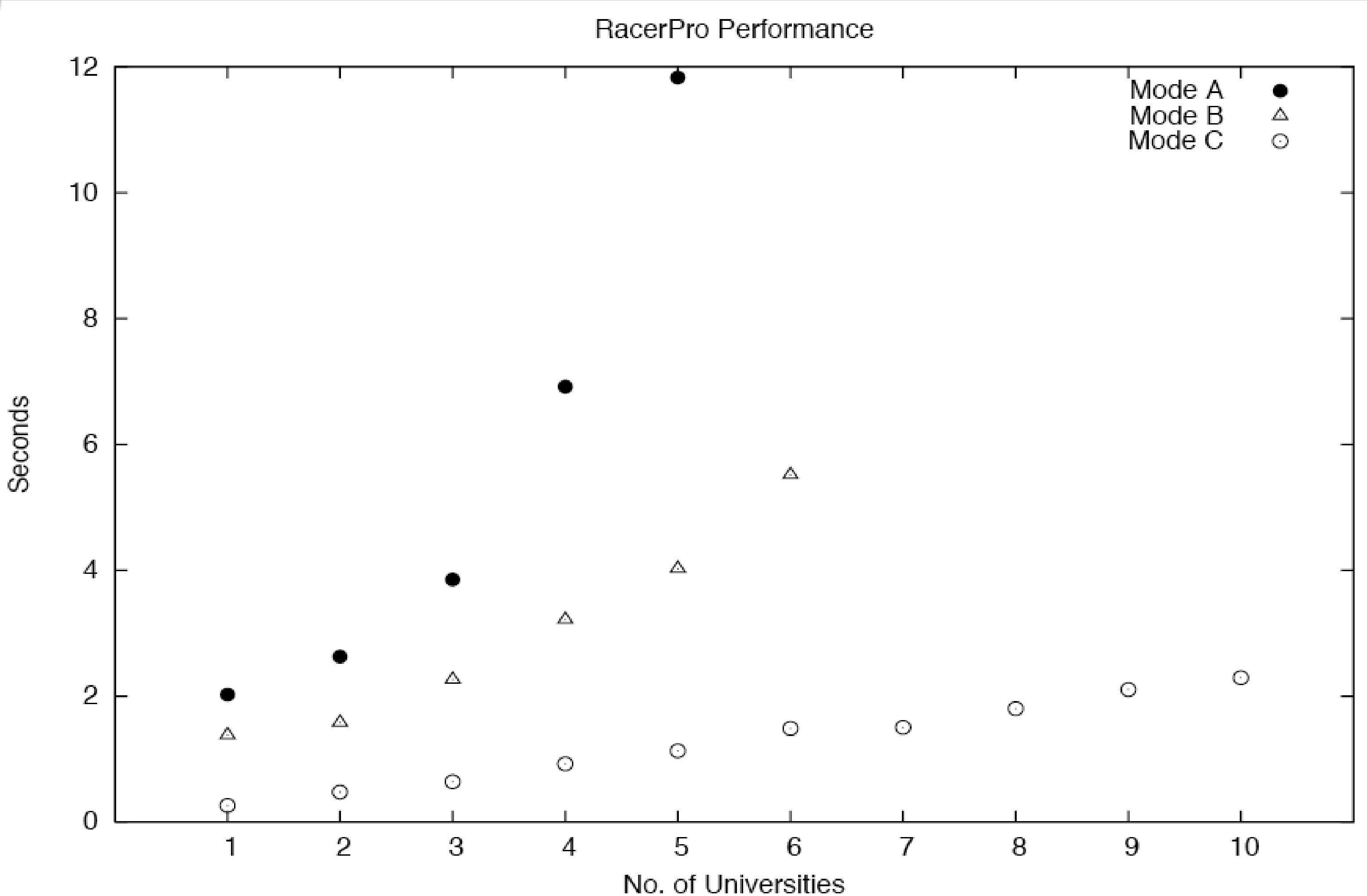
LUBM Evaluation (2)



Reasoning Modes

- A (complete)
  - Constraint reasoning for datatypes
    - Reals (incremental), Strings (incremental)
- B (complete for LUBM)
  - Told value reasoning for datatypes
- C (complete for LUBM)
  - Told value reasoning for datatypes
  - Transformation of sufficient conditions (query transformation, as explained)

LUBM Evaluation (3)



Conclusion

- Results encouraging for problems using higher expressivity
- Optimization techniques proposed might be included in any tableau-based DL prover that exists or might be built
- Memory consumption matters (see consumed time for ABox consistency checks -> GC problem)
  - -> Persistent Tableaus / ABoxes
- Thank you!