Querying the Semantic Web with Racer + nRQL

Applications on Description Logics ’04

Volker Haarslev, Ralf Möller, Michael Wessel

Software Systems Group
Hamburg University of Science and Technology
`r.f.moeller@tuhh.de`

Overview of Talk

- Introducing nRQL

Overview of Talk

- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics

Overview of Talk

- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics
- Querying Racer ABoxes with nRQL

Overview of Talk

- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics
- Querying Racer ABoxes with nRQL
  - Example Session

Overview of Talk

- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics
- Querying Racer ABoxes with nRQL
  - Example Session
- Benchmarking Racer + nRQL

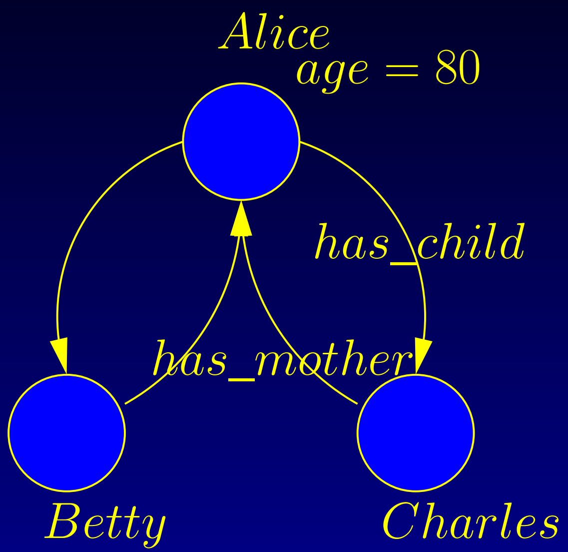
Overview of Talk

- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics
- Querying Racer ABoxes with nRQL
  - Example Session
- Benchmarking Racer + nRQL
  - The Lehigh University Benchmark (LUBM)
  - Evaluation

Overview of Talk

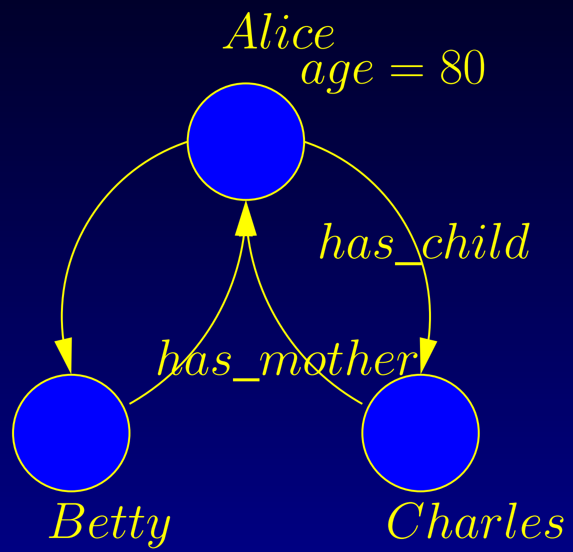
- Introducing nRQL
  - Introductory Example
  - Overview of Features
  - Query Processing Modes
  - Syntax & Semantics
- Querying Racer ABoxes with nRQL
  - Example Session
- Benchmarking Racer + nRQL
  - The Lehigh University Benchmark (LUBM)
  - Evaluation
- Conclusion & Future Work

Motivating Simple Example



$mother(alice), age(alice) = 80,$
$has\_mother(betty, alice),$
$has\_mother(charles, alice),$
$mother(betty), mother(betty)$

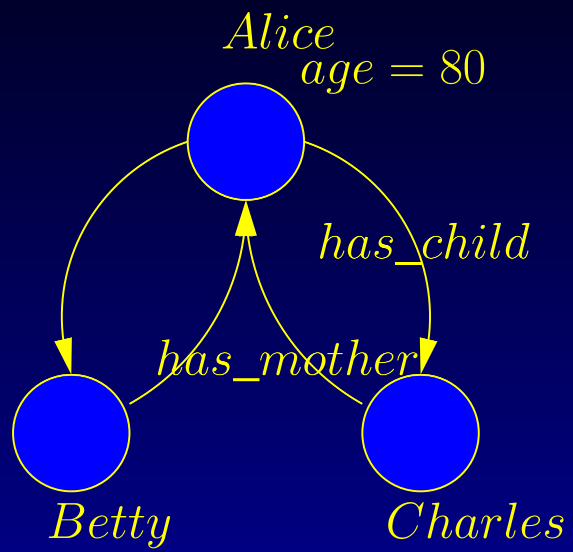
Motivating Simple Example



$$mother(alice), age(alice) = 80,$$
$$has\_mother(betty, alice),$$
$$has\_mother(charles, alice),$$
$$mother(betty), mother(betty)$$

- How do we retrieve <u>pairs</u> of siblings (e.g., $\{(betty, charles)\}$)?

Motivating Simple Example

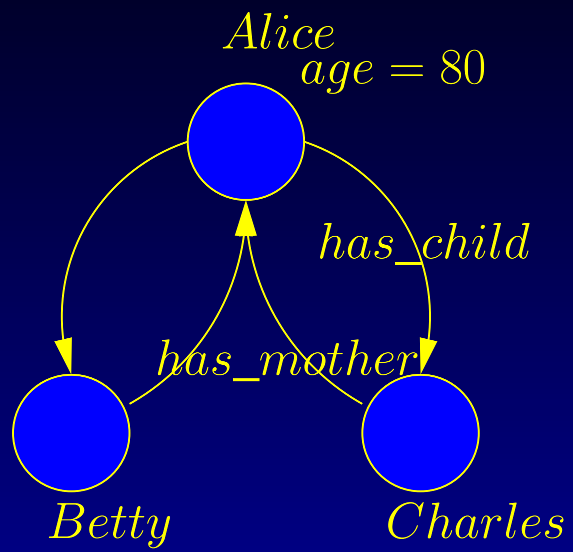


$mother(alice), age(alice) = 80,$
$has\_mother(betty, alice),$
$has\_mother(charles, alice),$
$mother(betty), mother(betty)$

- How do we retrieve pairs of siblings (e.g., $\{(betty, charles)\}$)?

⊖ write a “search program” (not declarative)

Motivating Simple Example



$mother(alice), age(alice) = 80,$
$has\_mother(betty, alice),$
$has\_mother(charles, alice),$
$mother(betty), mother(betty)$

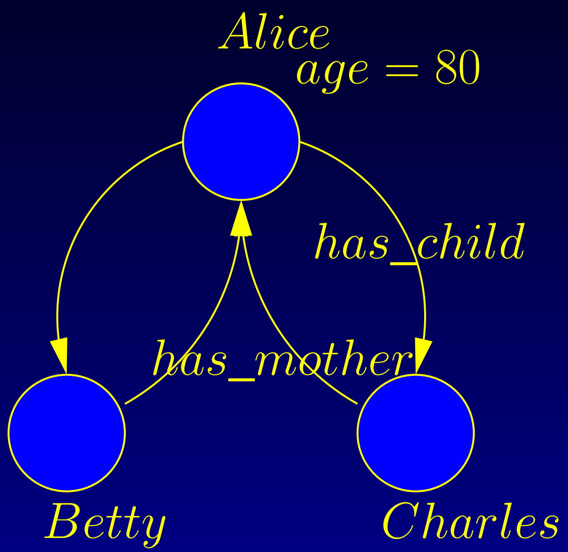
- How do we retrieve pairs of siblings (e.g., $\{(betty, charles)\}$)?

⊖ write a “search program” (not declarative)

⊕ use nRQL:

```
(retrieve (?x ?y)
          (and (has-child ?z ?x)
               (has-child ?z ?y)))
```

Motivating Simple Example



$mother(alice)$, $age(alice) = 80$,
$has\_mother(betty, alice)$,
$has\_mother(charles, alice)$,
$mother(betty)$, $mother(betty)$

- How do we retrieve pairs of siblings (e.g., $\{(betty, charles)\}$)?

⊖ write a “search program” (not declarative)

⊕ use nRQL:

$\Rightarrow$
```
(((?X CHARLES) (?Y BETTY))
 ((?X BETTY) (?Y CHARLES)))
```

nRQL Language – Features

- Concept and role query atoms

nRQL Language – Features

- Concept and role query atoms
  - With variables:
    ```
    (retrieve (?x) (?x woman))
    ```
  - Without variables:
    ```
    (retrieve () (betty woman))
    ```
  - With individuals:
    ```
    (retrieve (betty) (betty woman))
    =
    (retrieve (?betty-var)
        (and (?betty-var woman)
             (same-as ?betty-var betty)))
    ```

nRQL Language – Features

- Concept and role query atoms
  - ```
    (retrieve (?x ?y)
              (?x ?y has-child))
    ```
  - ```
    (retrieve (betty ?y)
              (betty ?y has-child))
    ```
  - ...

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
  - ```
    (retrieve (?x)
        (?x ?x (:constraint
            (has-child has-father age)
            (has-child has-mother age)
            (> age-1 (+ 10 age-2)))))
    ```
    ```

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
  - `(retrieve (?x) (neg (?x woman)))`
  - `(retrieve (?x ?y)`
    `(neg (?x ?y has-child)))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
    - NAF for atoms with individuals can be tricky:

```
(retrieve (betty)
          (neg (betty woman)))
=
(retrieve (?betty-var)
     (neg (and (?betty-var woman)
               (same-as ?betty-var betty))))
```

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
  - NAF for atoms with individuals can be tricky:

```
(retrieve (betty)
          (neg (betty woman)))
=
(retrieve (?betty-var)
    (UNION (neg (?betty-var woman))
           (neg (same-as ?betty-var betty))))
```

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
  - `(retrieve (?x) (?x (not woman)))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
  - `(retrieve (?x) (?x (not woman)))`
  - `(retrieve (?x ?y)`
    `(?x ?y (not has-child)))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
  - ```
    (retrieve (?x (AGE ?x))
        (?x (and human (an age))))
    ```
    ```

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
    - `(retrieve (?x (FILLERS (AGE ?x)))`
      `(?x (and human (an age))))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
- Extended Racer concept syntax for OWL datatype properties

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
- Extended Racer concept syntax for OWL datatype properties
  - `(retrieve ((fillers (OWL-DTP ?x))) (?x (MIN OWL-DTP 10)))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
- Extended Racer concept syntax for OWL datatype properties
- “pseudo-nominals” for concept expressions

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
- Extended Racer concept syntax for OWL datatype properties
- “pseudo-nominals” for concept expressions
  - `(retrieve (?x) (?x (some has-child BETTY)))`

nRQL Language – Features

- Concept and role query atoms
- Concrete domain (CD): constraint query atoms (with role chains ended by CD attribute)
- Negation as failure (NAF)
- True negation (also in role query atoms!)
- Projection operators to fillers of CD attributes (OWL datatype properties)
- Projection to told values of the CD
- Extended Racer concept syntax for OWL datatype properties
- "pseudo-nominals" for concept expressions

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- Set-at-a-time mode ("Get all tuples")

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- <u>Set-at-a-time mode</u> (“Get all tuples”)
- <u>Tuple-at-a-time mode</u> (“Get next tuple”)

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- <u>Set-at-a-time mode</u> (“Get all tuples”)
- <u>Tuple-at-a-time mode</u> (“Get next tuple”)
    - Lazy: compute next tuple if requested
    - Eager: precompute next tuples (proactive)

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- Set-at-a-time mode ("Get all tuples")
- Tuple-at-a-time mode ("Get next tuple")

$\Rightarrow$ **Incremental, concurrent querying!**

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- Set-at-a-time mode ("Get all tuples")
- Tuple-at-a-time mode ("Get next tuple")

⇒ **Incremental, concurrent querying!**

- "Max. no. of tuples" bound and timeout settable, cancelable

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- Set-at-a-time mode (“Get all tuples”)
- Tuple-at-a-time mode (“Get next tuple”)

$\Rightarrow$ **Incremental, concurrent querying!**

- “Max. no. of tuples” bound and timeout settable, cancelable
- Degree of completeness configurable (next slide)

nRQL Engine – Features (1)

- Internal part of Racer (otherwise drastic communication overhead!)
- Cost-based optimizer (uses ABox statistics and well-known CSP optimization techniques)
- <u>Set-at-a-time mode</u> (“Get all tuples”)
- <u>Tuple-at-a-time mode</u> (“Get next tuple”)

$\Rightarrow$ **Incremental, concurrent querying!**

- “Max. no. of tuples” bound and timeout settable, cancelable
- Degree of completeness configurable (next slide)
- Additional index structures for Racer ABoxes, suitable for mass-data processing

nRQL Engine – Features (2)

- Degree of completeness configurable:

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)
- 7th tuple-at-a-time mode: “two-phase processing”

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)
- 7th tuple-at-a-time mode: “two-phase processing”
  - Phase 1: deliver cheap tuples (incomplete)

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)
- 7th tuple-at-a-time mode: “two-phase processing”
  - Phase 1: deliver cheap tuples (incomplete)
  - Warn user; then, if next tuple requested, start

nRQL Engine – Features (2)

- Degree of completeness configurable:
  - Told information (very incomplete)
  - Told information + exploited TBox information (similar to DLDB, Ins.Store, . . .)
  - Complete Racer ABox Retrieval (expensive!)
- $3 \times \#\{set\_at\_a\_time, tuple\_at\_a\_time\} = 6$
- Variations: realize ABox / classify TBox (or not)
- 7th tuple-at-a-time mode: “two-phase processing”
  - Phase 1: deliver cheap tuples (incomplete)
  - Warn user; then, if next tuple requested, start
  - Phase 2: use full ABox reasoning to deliver remaining tuples (complete)

Incremental Query Processing

TBox:

$$\begin{aligned} person &\sqsubseteq \top \\ man &\sqsubseteq person \\ woman &\sqsubseteq person \\ spouse &\doteq woman \sqcap \\ & \quad (\exists married\_to.man) \end{aligned}$$

ABox :

$$spouse(doris)$$
$$spouse(betty)$$
$$man(adam)$$
$$woman(eve)$$
$$maried\_to(eve, adam)$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

Incremental Query Processing

TBox:

$$
\begin{aligned}
person &\sqsubseteq \top \\
man &\sqsubseteq person \\
woman &\sqsubseteq person \\
spouse &\doteq woman \sqcap \\
&\quad (\exists married\_to.man)
\end{aligned}
$$

ABox :

$$
\begin{aligned}
&spouse(doris) \\
&spouse(betty) \\
&man(adam) \\
&woman(eve) \\
&maried\_to(eve, adam)
\end{aligned}
$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `((?X DORIS))`

Incremental Query Processing

TBox:	ABox :
$person \sqsubseteq \top$	$spouse(doris)$
$man \sqsubseteq person$	$spouse(betty)$
$woman \sqsubseteq person$	$man(adam)$
$spouse \doteq woman \sqcap$	$woman(eve)$
$(\exists married\_to.man)$	$maried\_to(eve, adam)$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `((?X BETTY))`

Incremental Query Processing

TBox:

$$\begin{aligned} person &\sqsubseteq \top \\ man &\sqsubseteq person \\ woman &\sqsubseteq person \\ spouse &\doteq woman \sqcap \\ &\quad (\exists married\_to.man) \end{aligned}$$

ABox :

$$\begin{aligned} &spouse(doris) \\ &spouse(betty) \\ &man(adam) \\ &woman(eve) \\ &maried\_to(eve, adam) \end{aligned}$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `:WARNING-EXPENSIVE-PHASE-TWO-STARTS`

Incremental Query Processing

TBox:

$$\begin{aligned} person &\sqsubseteq \top \\ man &\sqsubseteq person \\ woman &\sqsubseteq person \\ spouse &\doteq woman \sqcap \\ & \quad (\exists married\_to.man) \end{aligned}$$

ABox :

$$\begin{aligned} & spouse(doris) \\ & spouse(betty) \\ & man(adam) \\ & woman(eve) \\ & maried\_to(eve, adam) \end{aligned}$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `((?X EVE))`

Incremental Query Processing

TBox:

$$
\begin{aligned}
person &\sqsubseteq \top \\
man &\sqsubseteq person \\
woman &\sqsubseteq person \\
spouse &\doteq woman \sqcap \\
&\quad (\exists married\_to.man)
\end{aligned}
$$

ABox :

$$
\begin{aligned}
&spouse(doris) \\
&spouse(betty) \\
&man(adam) \\
&woman(eve) \\
&maried\_to(eve, adam)
\end{aligned}
$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-next-tuple :query-1)`

$\Rightarrow$ `:EXHAUSTED`

Incremental Query Processing

TBox:

$$person \sqsubseteq \top$$
$$man \sqsubseteq person$$
$$woman \sqsubseteq person$$
$$spouse \doteq woman \sqcap (\exists married\_to.man)$$

ABox :

$$spouse(doris)$$
$$spouse(betty)$$
$$man(adam)$$
$$woman(eve)$$
$$maried\_to(eve, adam)$$

- `(retrieve (?x) (?x spouse))`

$\Rightarrow$ `(:QUERY-1 :RUNNING)`

- `(get-answer :query-1)`

$\Rightarrow$ `(((?X DORIS)) ((?X BETTY)) ((?X EVE)))`

nRQL Engine – Features (3)

- Reasoning with Queries
  - Incomplete for full nRQL, but still useful
  - Complete for restricted nRQL
  - Query consistency check
  - Query entailment check (subsumption)

nRQL Engine – Features (3)

- Reasoning with Queries
  - Incomplete for full nRQL, but still useful
  - Complete for restricted nRQL
  - Query consistency check
  - Query entailment check (subsumption)

$\Rightarrow$ maintenance of a “Query repository” lattice (similar to a TBox)

nRQL Engine – Features (3)

- Reasoning with Queries
  - Incomplete for full nRQL, but still useful
  - Complete for restricted nRQL
  - Query consistency check
  - Query entailment check (subsumption)

$\Rightarrow$ maintenance of a “Query repository” lattice (similar to a TBox)

$\Rightarrow$ use cached tuples of queries in repository for optimization purposes (“materialized views”)

nRQL Engine – Features (3)

- Reasoning with Queries
  - Incomplete for full nRQL, but still useful
  - Complete for restricted nRQL
  - Query consistency check
  - Query entailment check (subsumption)

  $\Rightarrow$ maintenance of a "Query repository" lattice (similar to a TBox)

  $\Rightarrow$ use cached tuples of queries in repository for optimization purposes ("materialized views")
  - Semantic optimization: query "realization" (similar to ABox realization)

    $\Rightarrow$ add implied conjuncts to enhance informdness of backtracking search

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
  - ```
    (defquery mother
              (?x ?y)
              (and (?x woman)
                   (?x ?y has-child)))

    (defquery mother-with-male-child
              (?x ?child)
              (and (:substitute
                         (mother ?x ?child))
                   (?child man)))
    ```
    ```

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
  - ```
    (defrule
       ((instance (new-ind child-of ?x ?y) human)
        (instance ?x mother)
        (instance ?y father)
        (related (new-ind child-of ?x ?y) ?x
                 has-mother)
        (related (new-ind child-of ?x ?y) ?y
                 has-father))

       (and (?x woman) (?y man) (?x ?y married))
            (neg (?x (:has-known-successor has-child))))
    ```
    ```

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
- Complex TBox queries

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
- Complex TBox queries
  - ```
    (tbox-query (?x ?y)
         (and (?x woman)
              (?x ?y has-descendant)))
    ```
    ```

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
- Complex TBox queries
- ... most of the present nRQL features have been requested by users (special thanks to R. v.d. Straeten)

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
- Complex TBox queries
- . . . most of the present nRQL features have been requested by users (special thanks to R. v.d. Straeten)
- Future work:
  - Projection operators within query body
  - “Rolling up”?
  - Approach OWL-QL?
  - Connect to real DB

nRQL Engine – Features (4)

- Defined queries (simple Macro-mechanism)
- Simple rule mechanism
- Complex TBox queries
- . . . most of the present nRQL features have been requested by users (special thanks to R. v.d. Straeten)
- Future work:
  - Projection operators within query body
  - “Rolling up”?
  - Approach OWL-QL?
  - Connect to real DB
- . . . for more nRQL peculiarities: see manual

nRQL - Syntax (1)

Let $a, b \in \mathcal{O}$; C be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ concept expression, R a nRQL role expression (a nRQL role expression is either a $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ role expression, or a <u>negated</u> $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ role expression); P one of the concrete domain expressions offered by Racer; and f, g be so-called attributes (whose range is defined to be one of the available concrete domains offered by Racer). Then, the set of <u>nRQL atoms</u> is given as follows:

- Unary concept query atoms: $C(a)$
- Binary role query atoms: $R(a, b)$
- Binary constraint query atoms: $P(f(a), g(b))$
- Binary same-as atoms: $same\_as(a, i)$
- Unary has-known-successor atoms: $has\_known\_successor(a, R)$
- Negated atoms: If A is a nRQL atom, then so is $\backslash(A)$, a so-called <u>negation as failure atom</u> or simply <u>negated atom.</u>

nRQL - Syntax (2)

A nRQL Query has a head and a body. Query bodies are defined inductively as follows:

- Each nRQL atom A is a body; and
- If $b_1 \dots b_n$ are bodies, then the following are also bodies:
  - $b_1 \wedge \dots \wedge b_n$, $b_1 \vee \dots \vee b_n$, $\backslash(b_i)$

We use the syntax $body(a_1, \dots, a_n)$ to indicate that $a_1, \dots, a_n$ are all the object names ($a_i \in \mathcal{O}$) mentioned in $body$. A nRQL Query is then an expression of the form

$$ans(a_{i_1}, \dots, a_{i_m}) \leftarrow body(a_1, \dots, a_n),$$

The expression $ans(a_{i_1}, \dots, a_{i_m})$ is also called the head, and $(i_1, \dots, i_m)$ is an index vector with $i_j \in 1 \dots n$. A conjunctive nRQL query is a query which does not contain any $\vee$ and $\backslash$ operators.

nRQL - Semantics (1)

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be an $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$ knowledge base. A positive ground query atom A is logically entailed (or implied) by $\mathcal{K}$ iff every model $\mathcal{I}$ of $\mathcal{K}$ is also a model of A. In this case we write $\mathcal{K} \models A$. Moreover, if $\mathcal{I}$ is a model of $\mathcal{K}$ (A) we write $\mathcal{I} \models \mathcal{K}$ ($\mathcal{I} \models A$). We therefore have to specify when $\mathcal{I} \models A$ holds. In the following, if the atom A contains individuals i, j, it will always be the case that $i, j \in \mathsf{inds}(\mathcal{A})$. From this it follows that $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ and $j^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, for any $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with $\mathcal{I} \models \mathcal{K}$:

- If $A = C(i)$, then $\mathcal{I} \models A$ iff $i^{\mathcal{I}} \in C^{\mathcal{I}}$.
- If $A = R(i, j)$, then $\mathcal{I} \models A$ iff $(i^{\mathcal{I}}, j^{\mathcal{I}}) \in R^{\mathcal{I}}$.
- If $A = P(f(i), g(j))$, then $\mathcal{I} \models A$ iff $(f^{\mathcal{I}}(i^{\mathcal{I}}), g^{\mathcal{I}}(j^{\mathcal{I}})) \in P^{\mathcal{I}}$.
- If $A = same\_as(i, i)$, then $\mathcal{I} \models A$.
- If $A = same\_as(i, j)$, then $\mathcal{I} \not\models A$.
- If $A = has\_known\_successor(i, R)$, then $\mathcal{I} \models A$ iff for some $j \in \mathsf{inds}(\mathcal{A})$: $\mathcal{I} \models R(i, j)$.

nRQL - Semantics (2)

Let $ans(a_{i_1}, \ldots, a_{i_m}) \leftarrow body(a_1, \ldots, a_n)$ be a nRQL query q such that $body$ is in NNF. Let $\beta(a_i) =_{def} x_{a_i}$ if $a_i \in \mathcal{I}$, and a_i otherwise; i.e., if a_i is an individual we replace it with its representative unique variable which we denote by x_{a_i}. Let $\mathcal{K}$ be the knowledge base to be queried, and $\mathcal{A}$ be its ABox. The answer set of the query q is then the following set of tuples:

$$\{ (j_{i_1}, \ldots, j_{i_m}) \mid \exists j_1, \ldots, j_n \in \mathsf{inds}(\mathcal{A}), \forall m, n, m \neq n : j_m \neq j_n, \\ \mathcal{K} \models_{NF} \alpha(body)_{[\beta(a_1) \leftarrow j_1, \ldots, \beta(a_n) \leftarrow j_n]} \}$$

Finally, we state that $\{()\} =_{def}$ TRUE and $\{\} =_{def}$ FALSE.

The LUBM

- Lehigh University Benchmark for benchmarking semantic web repositories
- See `http://www.lehigh.edu/~yug2/Research/SemanticWeb/LUBM/LUBM.htm`

The LUBM

- Lehigh University Benchmark for benchmarking semantic web repositories
- See `http://www.lehigh.edu/~yug2/Research/SemanticWeb/LUBM/LUBM.htm`
- Modeling of a university
  - OWL (DAML+OIL) classes for `department`s, various kinds of `professor`s, `student`s, ...
  - roles like `worksFor`, `subOrganization` (transitive),
  - Datatype properties `telephone, age,` ...

The LUBM

- Lehigh University Benchmark for benchmarking semantic web repositories
- See `http://www.lehigh.edu/~yug2/Research/SemanticWeb/LUBM/LUBM.htm`
- Modeling of a university
  - OWL (DAML+OIL) classes for `departments`, various kinds of `professors`, `students`, ...
  - roles like `worksFor`, `subOrganization` (transitive),
  - Datatype properties `telephone, age,` ...
- Benchmark generator generates “ABoxes”

The LUBM

- Lehigh University Benchmark for benchmarking semantic web repositories
- See `http://www.lehigh.edu/~yug2/Research/SemanticWeb/LUBM/LUBM.htm`
- Modeling of a university
  - OWL (DAML+OIL) classes for `departments`, various kinds of `professors`, `students`, . . .
  - roles like `worksFor`, `subOrganization` (transitive),
  - Datatype properties `telephone, age,` . . .
- Benchmark generator generates “ABoxes”
- 14 benchmarking queries

The LUBM

```
Query 9: (retrieve
           (?x ?y ?z)
           (and (?x Student)
                (?y Faculty)
                (?z Course)
                (?x ?y advisor)
                (?x ?z takesCourse)
                (?y ?z teacherOf)))
```

"Retrieve all triples `<?x,?y,?z>` such that `?x` is (bound to) a student undertaking a course `?z` whose teacher `?y` (from the faculty) happens to be his/her advisor"

The LUBM

```
Query 12: (retrieve
                (?x ?y www.University0.edu)
                (and (?x chair)
                     (?y Department)
                     (?x ?y memberOf)
                     (?y www.University0.edu
                         subOrganizationOf))
```

Cite LUBM: “The benchmark data do not produce any instances of class Chair. Instead, each Department individual is linked to the chair professor of that department by property headOf. Hence this query requires realization, i.e., inference that that professor is an instance of class Chair because he or she is the head of a department.”

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using <u>complete ABox querying</u>?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using complete ABox querying?
- How many LUBM departments can we “process” with a Racer ABox?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using complete ABox querying?
- How many LUBM departments can we “process” with a Racer ABox?
- With incomplete ABox querying?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using complete ABox querying?
- How many LUBM departments can we "process" with a Racer ABox?
- With incomplete ABox querying?
- How many LUBM departments can we process with a nRQL "ABox mirror"?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using <u>complete ABox querying</u>?
- How many LUBM departments can we “process” with a Racer ABox?
- With <u>incomplete ABox querying</u>?
- How many LUBM departments can we process with a nRQL “ABox mirror”?
- How bad is the incompleteness? How many tuples do we miss?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using complete ABox querying?
- How many LUBM departments can we “process” with a Racer ABox?
- With incomplete ABox querying?
- How many LUBM departments can we process with a nRQL “ABox mirror”?
- How bad is the incompleteness? How many tuples do we miss?
- Does it scale?

LUBM and Racer + nRQL?

- How far can we get with Racer + nRQL using complete ABox querying?
- How many LUBM departments can we “process” with a Racer ABox?
- With incomplete ABox querying?
- How many LUBM departments can we process with a nRQL “ABox mirror”?
- How bad is the incompleteness? How many tuples do we miss?
- Does it scale?
- How does Racer + nRQL perform (speed & completeness) compared to DLDB?

Benchmarking Racer + nRQL

- We ran LUBM queries in 3 settings:

Benchmarking Racer + nRQL

- We ran LUBM queries in 3 settings:
- Setting 1: complete ABox querying using an unrealized ABox

Benchmarking Racer + nRQL

- We ran LUBM queries in 3 settings:
- Setting 1: complete ABox querying using an <u>unrealized</u> ABox
- Setting 2: complete ABox reasoning using a <u>realized</u> ABox

Benchmarking Racer + nRQL

- We ran LUBM queries in 3 settings:
- Setting 1: complete ABox querying using an unrealized ABox
- Setting 2: complete ABox reasoning using a realized ABox
- Setting 3: “told information querying” enhanced with TBox information – “upward saturation”:

Benchmarking Racer + nRQL

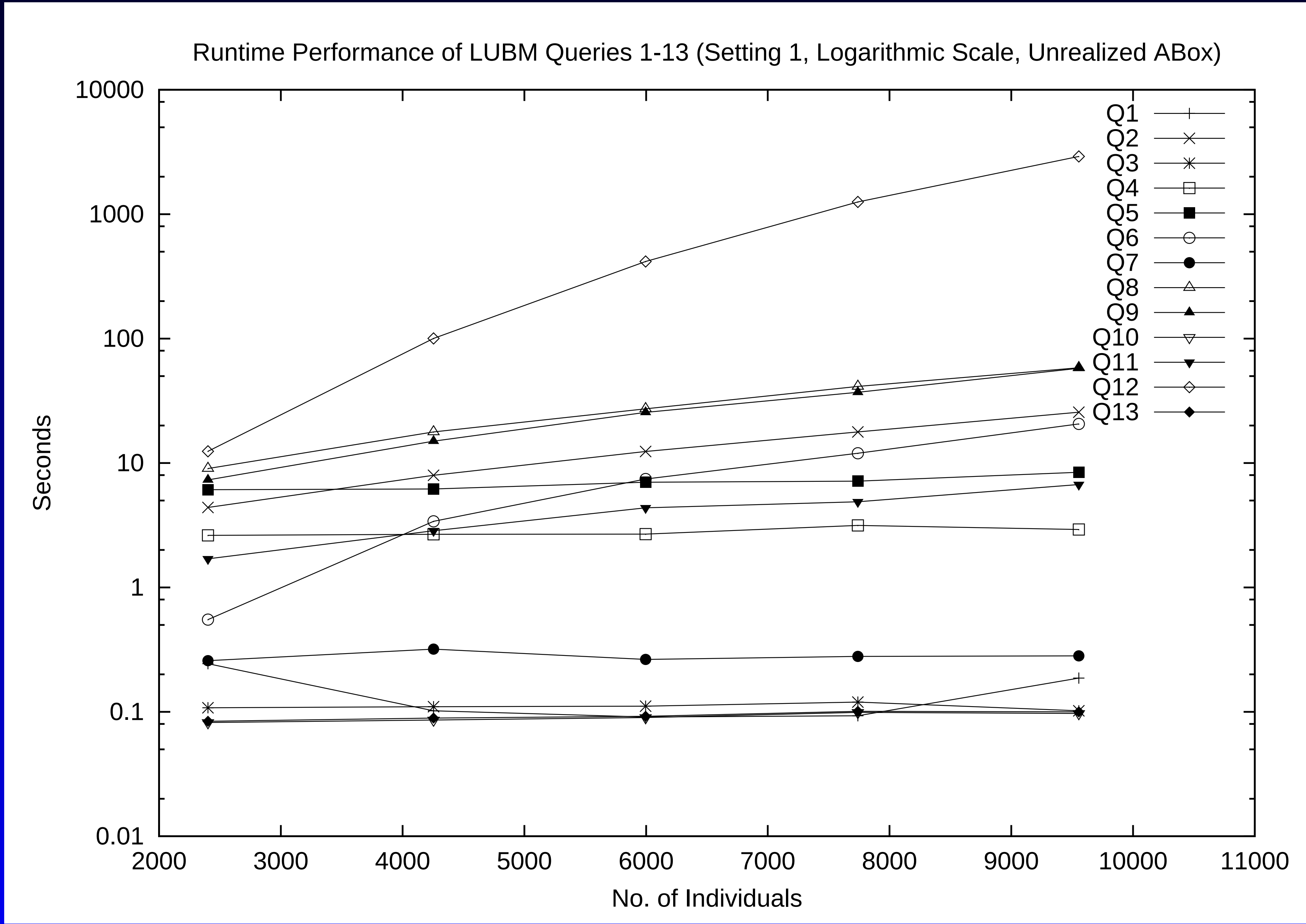
- We ran LUBM queries in 3 settings:
- Setting 1: complete ABox querying using an <u>unrealized</u> ABox
- Setting 2: complete ABox reasoning using a <u>realized</u> ABox
- Setting 3: "told information querying" enhanced with TBox information – "upward saturation":

  $\Rightarrow$ for each ABox axiom $C(i) \in \mathcal{A}$, for all $D \in \mathsf{concept\_ancestors}(C, TBox)$: put $D(i)$ into "ABox": $\mathcal{A} := \mathcal{A} \cup \{D(i)\}$

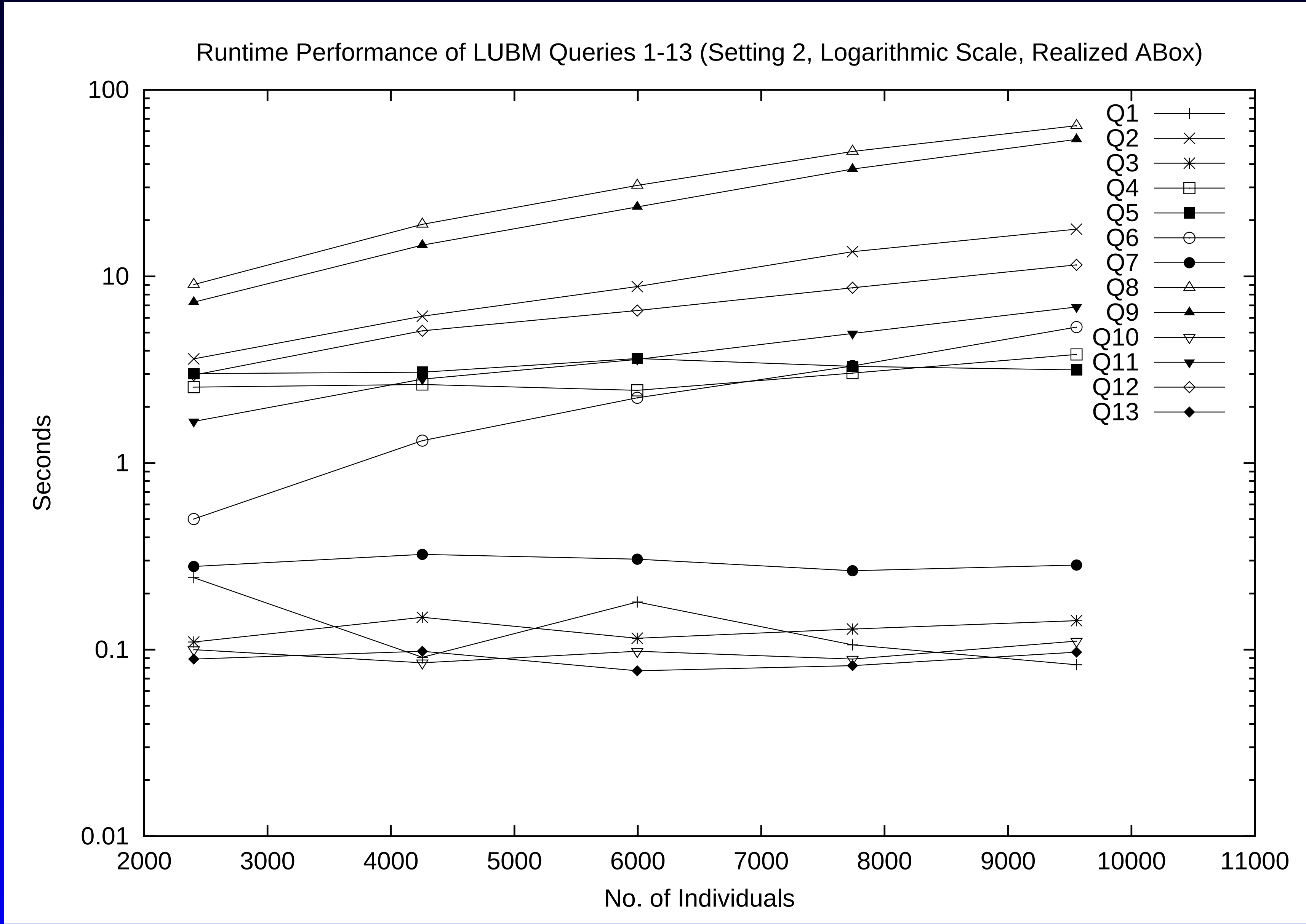
  $\Rightarrow$ same for role relationships due to role hierarchies

  - nRQL is always complete w.r.t. roles

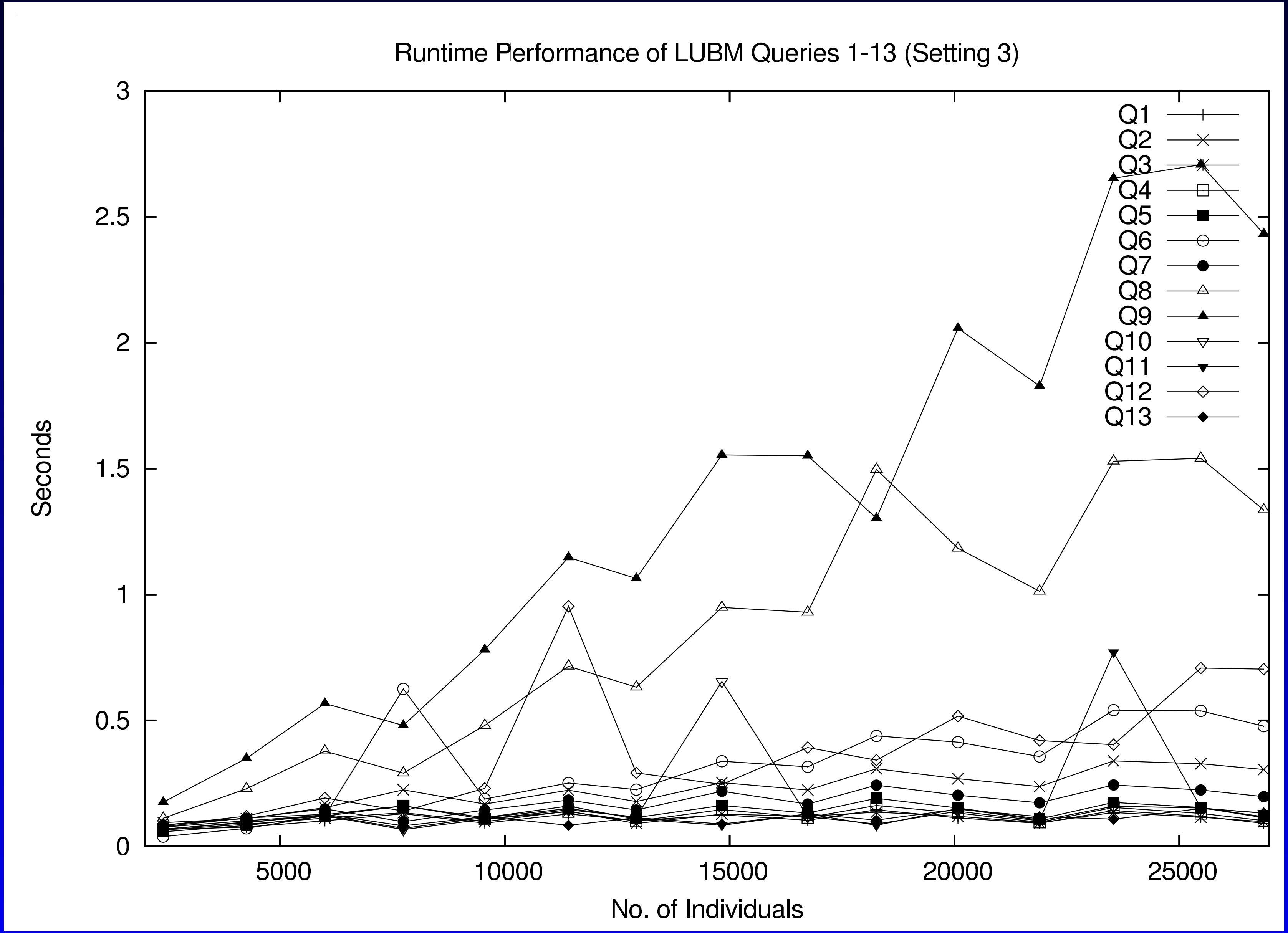
Results - Setting 1



Results - Setting 2



Results - Setting 3



Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)
- All but Q8 and Q9 can be answered in fractions of a second

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)
- All but Q8 and Q9 can be answered in fractions of a second
- Only one tuple is missed (for Q12)

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)
- All but Q8 and Q9 can be answered in fractions of a second
- Only one tuple is missed (for Q12)

$\Rightarrow$ not severely incomplete

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)
- All but Q8 and Q9 can be answered in fractions of a second
- Only one tuple is missed (for Q12)

$\Rightarrow$ not severely incomplete

$\Rightarrow$ even more complete than DLDB

Evaluation

- Using complete ABox querying we have to stop at approx. 10.000 LUBM individuals
- Initial ABox consistency test kills Racer
- If completeness is sacrificed, we can easily load and process more than 30.000 individuals (1 university)
- All but Q8 and Q9 can be answered in fractions of a second
- Only one tuple is missed (for Q12)

$\Rightarrow$ not severely incomplete

$\Rightarrow$ even more complete than DLDB

$\Rightarrow$ in this scale, answering time is quite okay!

Conclusion & Outlook

- Racer + nRQL is a semantic web repository

Conclusion & Outlook

- Racer + nRQL is a semantic web repository
- If completeness is sacrificed, the LUBM queries are easy to answer for a relatively small number of individuals (say, up to 100.000)

Conclusion & Outlook

- Racer + nRQL is a semantic web repository
- If completeness is sacrificed, the LUBM queries are easy to answer for a relatively small number of individuals (say, up to 100.000)
- The simple ABox “upward saturation” technique achieves a great degree of LUBM-completeness (an OWL TBox classifier is nevertheless needed)

Conclusion & Outlook

- Racer + nRQL is a semantic web repository
- If completeness is sacrificed, the LUBM queries are easy to answer for a relatively small number of individuals (say, up to 100.000)
- The simple ABox “upward saturation” technique achieves a great degree of LUBM-completeness (an OWL TBox classifier is nevertheless needed)

$\Rightarrow$ the LUBM is probably “to easy” in this respect

Conclusion & Outlook

- Racer + nRQL is a semantic web repository
- If completeness is sacrificed, the LUBM queries are easy to answer for a relatively small number of individuals (say, up to 100.000)
- The simple ABox "upward saturation" technique achieves a great degree of LUBM-completeness (an OWL TBox classifier is nevertheless needed)

$\Rightarrow$ the LUBM is probably "to easy" in this respect

$\Rightarrow$ the amount of data generated by the LUBM is demanding

Conclusion & Outlook

- Racer + nRQL is a semantic web repository
- If completeness is sacrificed, the LUBM queries are easy to answer for a relatively small number of individuals (say, up to 100.000)
- The simple ABox “upward saturation” technique achieves a great degree of LUBM-completeness (an OWL TBox classifier is nevertheless needed)

$\Rightarrow$ the LUBM is probably “to easy” in this respect

$\Rightarrow$ the amount of data generated by the LUBM is demanding

$\Rightarrow$ hope: future versions of Racer will be able to process much bigger ABoxes

Thanks

for your

attention!