Extended Query Facilities for Racer and an Application to Software-Engineering Problems

Volker Haarslev, Ralf Möller, Ragnhild Van Der Straeten and Michael Wessel

Cognitive Systems Group
University of Hamburg
Germany
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\text{ALCQHI}_{R^+} \neg (D^-)$

- Trend: recently, users create really big ABoxes...
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\text{ALCQHI}_{R^+}(D^-)$
  - offers ABox retrieval functions
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQH} \mathcal{I}_{\mathcal{R}^+}(\mathcal{D}^-)$
  - offers ABox retrieval functions
    - $(\text{concept-instances } C)$
A New Pragmatic ABox Query Language for Racer

- **Racer**
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$
  - offers ABox retrieval functions
    - (concept-instances C)
    - (individual-fillers i R)
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$
  - offers ABox retrieval functions
    - (concept-instances C)
    - (individual-fillers i R)
    - ...a few dozens more (see Racer manual)
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQHI}_{R^+}(D^-)$
  - offers ABox retrieval functions
  - Trend: recently, users create really big ABoxes ...
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQHI}_{R^+}(D^-)$
  - offers ABox retrieval functions
  - Trend: recently, users create really big ABoxes . . .
  - . . . or want to navigate through the relational structure of an ABox (Ragnhild, DL 2003)

A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\text{ALCQHI}_{R+}(\text{D}^-)$
  - offers ABox retrieval functions
  - Trend: recently, users create really big ABoxes . . .
  - . . . or want to navigate through the relational structure of an ABox (Ragnhild, DL 2003)
  - more sophisticated “LOOM-like” queries demanded by users
A New Pragmatic ABox Query Language for Racer

- Racer
  - Volker’s & Ralf’s ABox DL reasoner for $\text{ALCQHI}^{R+}(D^-)$
  - offers ABox retrieval functions
  - Trend: recently, users create really big ABoxes . . .
  - . . . or want to navigate through the relational structure of an ABox (Ragnhild, DL 2003)
  - $\Rightarrow$ more sophisticated “LOOM-like” queries demanded by users
  - $\Rightarrow$ RQL: Racer Query Language
A New Pragmatic ABox Query Language for Racer

• Racer
  • Volker’s & Ralf’s ABox DL reasoner for $\textit{ALCQHI}_{R^+}(D^-)$
  • offers ABox retrieval functions
  • Trend: recently, users create really big ABoxes . . .
  • . . . or want to navigate through the relational structure of an ABox (Ragnhild, DL 2003)
  ⇒ more sophisticated “LOOM-like” queries demanded by users
  ⇒ nRQL: new Racer Query Language
A New Pragmatic ABox Query Language for Racer

- **Racer**
  - Volker’s & Ralf’s ABox DL reasoner for $\mathcal{ALCQHI}_{\mathcal{R}^+}(\mathcal{D}^-)$
  - offers ABox retrieval functions
  - Trend: recently, users create really big ABoxes . . .
  - . . . or want to navigate through the relational structure of an ABox (Ragnhild, DL 2003)
  - more sophisticated “LOOM-like” queries demanded by users
  - nRQL: new Racer Query Language
    - pragmatic approach, straightforward combination of Racer’s ABox API functions
Motivating Simple Example

Alice

age = 80

Betty

Charles

has_child

has_mother

mother(alice), age(alice) = 80,
has_mother(betty, alice),
has_mother(charles, alice),
mother(betty), mother(betty)
Motivating Simple Example

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

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

  ⊗ write a “search program” (not declarative)
Motivating Simple Example

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

⊙ write a “search program” (not declarative)

⊕ use nRQL:

\[
\text{retrieve (?x ?y)}
\]

\[
\text{and (has-child ?z ?x)}
\]

\[
\text{(has-child ?z ?y))}
\]
Motivating Simple Example

- How do we retrieve pairs of siblings (e.g., \{(betty, charles)\})?
- Write a “search program” (not declarative)
- Use nRQL:
  \[ \Rightarrow ((X CHARLES) (Y BETTY)) \]
  \[ ((X BETTY) (Y CHARLES)) \]
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
  - UML diagrams are represented as ABoxes
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
- UML diagrams are represented as ABoxes
- used LOOM, but wants Racer (DL ’03)
Motivating Example

• Work of Ragnhild v.d. Straeten et al.
  • checking for inconsistencies in UML diagrams
  • e.g., are there instances of classes for which no class has been defined? (CWA!)
• UML diagrams are represented as ABoxes
• used LOOM, but wants Racer (DL ’03)

• LOOM:
  (do-retrieve (?object ?class)
    (:and (Object ?object)
      (:and (Instance-of-class ?object ?class)
        (has-classmodel ?class NIL))))
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
- UML diagrams are represented as ABoxes
- used LOOM, but wants Racer (DL ’03)
- Racer without nRQL:
  
  \[
  \text{(loop for } ?o \text{ in (concept-instances object) do (loop for } ?c \text{ in (loop for } ?o' 'instance-of-class' do (ret.-ind.-fillers } ?o \text{'has-classmodel'}) \text{ when (null (ret.-ind.-fillers } ?c \text{'has-classmodel'}) do (format t ''Classless instance'')))}
  \]
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
- UML diagrams are represented as ABoxes
- used LOOM, but wants Racer (DL ’03)

- Racer with nRQL:
  
  \[
  \text{(retrieve (?o ?c)}
  \]
  \[
  \text{ (and (?c class) (?o object)}
  \]
  \[
  \text{ (?o ?c instance-of)}
  \]
  \[
  \text{ (not (?c (:has-known-successor}
  \]
  \[
  \text{ has-classmodel)))))}
  \]
Motivating Example

- Work of Ragnhild v.d. Straeten et al.
  - checking for inconsistencies in UML diagrams
  - e.g., are there instances of classes for which no class has been defined? (CWA!)
- UML diagrams are represented as ABoxes
- used LOOM, but wants Racer (DL ’03)

- Racer with nRQL:
  
  \[
  (\text{retrieve} \ (?o \ ?c) \\
  (\text{and} \ (?c \ \text{class}) \ (?o \ \text{object}) \\
  (?o \ ?c \ \text{instance-of}) \\
  (?c \ \text{NIL has-classmodel}))
  \]
Pragmatic Design Decisions

- Variables range over ABox individuals only
Pragmatic Design Decisions

- Variables range over ABox individuals only
  - no distinction of must-bind, may-bind and do-not-bind variables (OWL-QL)
  - variables are always distinguished resp. must-bind

Support for "Negation as Failure"

Support for disjunctive queries

Support for concrete domain predicates
Pragmatic Design Decisions

- Variables range over ABox individuals only
  - no distinction of must-bind, may-bind and do-not-bind variables (OWL-QL)
  - variables are always distinguished resp. must-bind
- no automatic “rolling up”, e.g. if \(?y\) is not distinguished (no bindings wanted!)
  
  \((?x \ [\ ?y] \ R) \Rightarrow (\ ?x \ (\text{some } R \text{ top}))\)
Pragmatic Design Decisions

- Variables range over ABox individuals only
  - no distinction of must-bind, may-bind and do-not-bind variables (OWL-QL)
  - variables are always distinguished resp. must-bind
  - no automatic “rolling up”, e.g. if \(?y\) is not distinguished (no bindings wanted!)
    \((?x \ n \ y \ R) \Rightarrow (?x \ (\text{some} \ R \ \text{top}))\)
- Support for “Negation as Failure”
Pragmatic Design Decisions

- Variables range over ABox individuals only
  - no distinction of must-bind, may-bind and do-not-bind variables (OWL-QL)
  - variables are always distinguished resp. must-bind

- no automatic “rolling up”, e.g. if \(?y\) is not distinguished (no bindings wanted!)
  \( (?x \ ?y R) \Rightarrow (\ ?x (\text{some } R \text{ top})) \)

- Support for “Negation as Failure”
  - \((\text{retrieve } (?x) (\ ?x (\text{not } woman)))\)
  - \((\text{retrieve } (?x) (\text{not } (?x woman)))\)

- Support for disjunctive queries
Pragmatic Design Decisions

- Variables range over ABox individuals only
  - no distinction of must-bind, may-bind and do-not-bind variables (OWL-QL)
  - variables are always distinguished resp. must-bind
  - no automatic “rolling up”, e.g. if $?y$ is not distinguished (no bindings wanted!)
    
    $$(?x \ [\underline{?y}] \ R) \Rightarrow (?x \ (\text{some } R \top))$$

- Support for “Negation as Failure”
  - (retrieve (?x) (?x (not woman)))
  - (retrieve (?x) (not (?x woman)))

- Support for disjunctive queries
- Support for concrete domain predicates
nRQL Syntax - Query Atoms

- Unary Atoms
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: \texttt{woman(alice)}
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - **Concept query atoms:** \(\text{woman}(alice)\)
  
  \[
  \text{(retrieve () (alice woman))}
  \]
  
  \[\Rightarrow T\]
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: \textit{woman}(alice)
    (retrieve () (alice woman))
    \[\Rightarrow \text{T}\]
    
    (individual-instance? alice woman)
    \[\Rightarrow \text{T}\]
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $woman(x)$

- Binary Atoms
  - Role query atoms:
    - has child ($alice; charles$)
    - has child ($alice; y$)
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $\text{woman}(x)$
    
    \[
    \text{(retrieve } (?x) (?x \text{ woman})) \\
    \Rightarrow ((?x \text{ alice}) ((?x \text{ betty}))
    \]
nRQL Syntax - Query Atoms

• Unary Atoms

  • Concept query atoms: \( \text{woman}(x) \)
    
    (retrieve (?x) (?x woman))
    
    \( \Rightarrow (\text{(alice alice)} \text{(alice betty)}) \)

    (concept-instances woman)
    
    \( \Rightarrow (\text{betty alice}) \)
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms:
    \[\text{has}_\text{known}_\text{successor}(alice, \text{has}_\text{child})\]
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms:
    \[ \text{has\_known\_successor} (alice, has\_child) \]
    
    (retrieve ()
      (alice (:has-known-successor
        has-child))
    )

  \[ \Rightarrow T \]
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - "Has known successor" atoms:
    $\text{has\_known\_successor}(x, \text{has\_child})$
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms:
    $\text{has\_known\_successor}(x, has\_child)$

(retrieve ()
  (?x (:has-known-successor
       has-child)))
⇒ (((?x alice)))
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms: $\text{has\_known\_successor}(x, \text{has\_child})$

(retrieve ()
  (?x (:has-known-successor
       has-child)))
⇒ (((?x alice)))

• What about Betty? ABox: $\text{mother}(\text{betty})$
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms:
    \[ has\_known\_successor(x, has\_child) \]
    
    \[
    \text{retrieve ()}
    \]
    
    \[
    (?x (:has-known-successor has-child))
    \]
    
    \[
    \Rightarrow (((?x alice)))
    \]
  • What about Betty? ABox: mother(betty)
  • (retrieve (?x))
    
    \[
    (?x (some has-child top))
    \]
    
    \[
    \Rightarrow (((?x alice)) ((?x betty)))
    \]
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms: $has\_known\_successor(a, R)$

• Binary Atoms
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- Binary Atoms
  - Role query atoms: $has\_child(alice, charles)$
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $\text{has\_child}(alice, charles)$

  (retrieve ()
   (alice charles has-child))

  $\Rightarrow T$
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms: 
    $has\_known\_successor(a, R)$

• Binary Atoms
  • Role query atoms: $has\_child(alice, charles)$

(retrieve ()
  (alice charles has-child))

⇒ T

(individuals-related? alice charles has-child)

⇒ T
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- Binary Atoms
  - Role query atoms: $\text{has\_child}(alice, y)$
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: \( C(a) \)
  • “Has known successor” atoms: \( has\_known\_successor(a, R) \)

• Binary Atoms
  • Role query atoms: \( has\_child(alice, y) \)
    (retrieve (?y) (alice ?y has-child))
    \( \implies (((?y betty)) ((?y charles))) \)
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- **Binary Atoms**
  - Role query atoms: $has\_child(alice, y)$
    
    (retrieve (?y) (alice ?y has-child))
    
    $\Rightarrow$ (((?y betty)) ((?y charles)))
    
    (individual-fillers alice has-child)
    
    $\Rightarrow$ (betty charles)
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms: \( has\_known\_successor(a, R) \)

- **Binary Atoms**
  - Role query atoms: \( has\_child(x, y) \)
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms: 
    \( has\_known\_successor(a, R) \)

- **Binary Atoms**
  - Role query atoms: \( has\_child(x, y) \)
    
    (retrieve (?x ?y) (?x ?y has-child))

    \( \Rightarrow (((?x alice) (?y betty))
    (((?x alice) (?y charles))) \))
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

• Binary Atoms
  • Role query atoms: $\text{has\_child}(x, y)$
    (retrieve (?x ?y) (?x ?y has-child))
    \[ \Rightarrow (((?x alice) (?y betty))
    (((?x alice) (?y charles)))\]

(related-individuals has-child)
\[ \Rightarrow ((alice betty) (alice charles))\]
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has known successor(a, R)$

- Binary Atoms
  - Role query atoms: $R(a, b)$
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms:
    $\text{has\_known\_successor}(a, R)$

- Binary Atoms
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $= (\text{has\_mother} \circ \text{age}(x), \text{has\_mother} \circ \text{age}(y))$
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms:
    $has\_known\_successor(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $= (has\_mother \circ age(x), has\_mother \circ age(y))$
    
    (retrieve (?x ?y)
    (?x ?y (:constraint (h-mother age)
    (h-mother age) =))

nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms: \( has\_known\_successor(a, R) \)

- **Binary Atoms**
  - Role query atoms: \( R(a, b) \)
  - Constraint query atoms:
    \[ = (has\_{mother} \circ age(x), has\_{mother} \circ age(y)) \]
    
    (retrieve (?x ?y)
      (?x ?y (:constraint (h-mother age)
        (h-mother age) =)))
    
    \[\Rightarrow ( ((?x alice) (?y betty)) ... )\]
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms: $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms: $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms: \( \text{has\_known\_successor}(a, R) \)

- **Binary Atoms**
  - Role query atoms: \( R(a, b) \)
  - Constraint query atoms:
    \[ P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b)) \]

- **“Negated” Atoms**
  - \( \neg(C(a)) \)
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms: $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
  - $\neg (C(a))$
  - (retrieve () (charles woman))
  - $\Rightarrow$ NIL
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: \( C(a) \)
  - “Has known successor” atoms:
    \( \text{has\_known\_successor}(a, R) \)

- **Binary Atoms**
  - Role query atoms: \( R(a, b) \)
  - Constraint query atoms:
    \( P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b)) \)

- **“Negated” Atoms**
  - \( \neg (C(a)) \)
  - \((\text{retrieve} () (\text{not} (\text{charles woman}))))\)

\( \Rightarrow T \)
nRQL Syntax - Query Atoms

• Unary Atoms
  • Concept query atoms: $C(a)$
  • “Has known successor” atoms: $has\_known\_successor(a, R)$

• Binary Atoms
  • Role query atoms: $R(a, b)$
  • Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

• “Negated” Atoms
  • $\neg (C(a))$
    (retrieve () (charles (not woman)))
    $\Rightarrow$ NIL
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
  - $\neg(C(a))$
    (retrieve (?x) (?x woman))
    $\Rightarrow (((?x \text{ betty})) ((?x \text{ alice})))$
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- Binary Atoms
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- “Negated” Atoms
  - $\neg (C(a))$
    (retrieve (?x) (not (?x woman)))
    $\Rightarrow (((?x \text{ charles})))$
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
  - $\neg (C(a))$
    
    (retrieve (?x) (?x (not woman)))
    
    $\Rightarrow ()$
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms: $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
  - $\neg (C(a))$
  - $\neg (\text{has\_known\_successor}(a, R))$
nRQL Syntax - Query Atoms

- Unary Atoms
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $has\_known\_successor(a, R)$

- Binary Atoms
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- “Negated” Atoms
  - $\neg(C(a))$
  - $\neg(has\_known\_successor(a, R))$

LOOM: (retrieve (?x) (?x NIL R))
nRQL Syntax - Query Atoms

- **Unary Atoms**
  - Concept query atoms: $C(a)$
  - “Has known successor” atoms: $\text{has\_known\_successor}(a, R)$

- **Binary Atoms**
  - Role query atoms: $R(a, b)$
  - Constraint query atoms:
    $P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$

- **“Negated” Atoms**
  - $\neg (C(a))$
  - $\neg (\text{has\_known\_successor}(a, R))$
  - $\neg R(a, b), \neg P(f_1 \circ \cdots \circ f_n(a), g_1 \circ \cdots \circ g_m(b))$
Semantics for Query Atoms

- Atoms use variables and/or individuals
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
- For unary atoms \( C(a) \)
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
- For unary atoms $C(a)$
  - $\text{inds}(\mathcal{A}) = \text{answer}(C(a)) \cup \text{answer}(\neg C(a))$
  - $\text{answer}(C(a)) \cap \text{answer}(\neg C(a)) = \emptyset$
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
- For unary atoms $C(a)$
  - $\text{inds}(\mathcal{A}) = \text{answer}(C(a)) \cup \text{answer}(\neg C(a))$
  - $\text{answer}(C(a)) \cap \text{answer}(\neg C(a)) = \emptyset$
- For binary atoms $R(a, b)$
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
- For unary atoms $C'(a)$
  - $\text{inds}(A) = \text{answer}(C'(a)) \cup \text{answer}(\neg C'(a))$
  - $\text{answer}(C'(a)) \cap \text{answer}(\neg C'(a)) = \emptyset$
- For binary atoms $R(a, b)$
  - $\text{inds}(A)^2 = \text{answer}(R(a, b)) \cup \text{answer}(\neg R(a, b)) \cup \text{Id}$
Semantics for Query Atoms

- Atoms use variables and/or individuals
- Variables are always “distinguished”
- Default: UNA for variables (non-UNA available)
- “Negation as Failure” / complement with “\”
- For unary atoms $C(a)$
  - $\text{inds}(\mathcal{A}) = \text{answer}(C(a)) \cup \text{answer}(\neg C(a))$
  - $\text{answer}(C(a)) \cap \text{answer}(\neg C(a)) = \emptyset$
- For binary atoms $R(a, b)$
  - $\text{inds}(\mathcal{A})^2 =$
    - $\text{answer}(R(a, b)) \cup \text{answer}(\neg R(a, b)) \cup \text{Id}$
  - $\text{answer}(R(a, b)) \cap \text{answer}(\neg R(a, b)) = \emptyset$
Neg. Atoms with Individuals?

- *atom* and \(\neg (atom)\) are always complementary
Neg. Atoms with Individuals?

- $atom$ and $\neg (atom)$ are always complementary
- $(\text{retrieve } () \ (\text{betty woman}))$
  \[ \Rightarrow T \]
Neg. Atoms with Individuals?

- $atom$ and $\neg (atom)$ are always complementary
- $(\text{retrieve (betty) (betty woman)})$ $\Rightarrow (((\text{betty betty})))$
Neg. Atoms with Individuals?

- $atom$ and $\neg (atom)$ are always complementary
- $(\text{retrieve } (\text{betty}) \ (\text{betty man}))$
  $\Rightarrow ()$
Neg. Atoms with Individuals?

- $atom$ and $\neg(atom)$ are always complementary
- $(\text{retrieve (betty) (not (betty man))})$
  $$\Rightarrow (\text{((betty charles) (betty alice) (betty betty))})$$
Neg. Atoms with Individuals?

- \( \text{atom} \) and \( \neg(\text{atom}) \) are always complementary

- \( \text{retrieve}(\text{x}) \ (\neg(\text{x man})) \)

\[ \Rightarrow (((\text{x charles}) \ (\text{x alice})
       \ (\text{x betty}))) \]
Neg. Atoms with Individuals?

- $atom$ and $\neg (atom)$ are always complementary
- $(\text{retrieve (betty) (not (betty man))})$
  \[
  \Rightarrow ((\text{betty charles}) \ (\text{betty alice})
  \ (\text{betty betty}))
  \]
Neg. Atoms with Individuals?

- \( atom \) and \( \neg (atom) \) are always complementary
- \( \text{(retrieve (betty) (not (betty man))}) \)
  \[ \Rightarrow (((\text{betty charles}) \ (\text{betty alice}) \ 
                    \ (\text{betty betty})) \)
  \[ \Rightarrow \text{“negated individuals” turn into variables} \]
Neg. Atoms with Individuals?

- $atom$ and $\neg (atom)$ are always complementary
- $(\text{retrieve} \ (\text{betty}) \ (\text{not} \ (\text{betty man})))$
  $\Rightarrow (((\text{betty charles}) \ (\text{betty alice}) \ (\text{betty betty})))$
  $\Rightarrow \text{“negated individuals” turn into variables}$
- Add a $\text{bind-individual}$ conjunct
Neg. Atoms with Individuals?

- \( atom \) and \( \\neg (atom) \) are always complementary
- \( \text{(retrieve (betty) (not (betty man)))} \)
  \( \Rightarrow \text{(((betty charles) (betty alice) (betty betty)))} \)

\( \Rightarrow \) “negated individuals” turn into variables

- Add a \textit{bind–individual} conjunct

\[ \text{(retrieve (betty) (and (not (betty man)) (bind–individual betty))} \]

\[ \Rightarrow \text{(((betty betty)))} \]
Query Bodies

- Each query atom is a body
Query Bodies

- Each query atom is a body
- If $b_i$ are query bodies, then also
Query Bodies

- Each query atom is a body
- If \( b_i \) are query bodies, then also
  - \( \land b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n \)
Query Bodies

• Each query atom is a body
• If \( b_i \) are query bodies, then also
  • \( \bigwedge b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n \)
• Meaning of “or”
  • TBox: \( \top \Rightarrow woman \sqcup man \)
Query Bodies

- Each query atom is a body
- If $b_i$ are query bodies, then also
  - $\bigwedge b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n$
- Meaning of “or”
  - TBox: $\top \Rightarrow woman \sqcup man$
  - (retrieve (?x) (?x top))
    $\Rightarrow (((?x alice)) ((?x betty)) ((?x charles)))$
Query Bodies

- Each query atom is a body
- If $b_i$ are query bodies, then also
  - $\bigwedge b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n$
- Meaning of “or”
  - TBox: $\top \Rightarrow \text{woman} \sqcup \text{man}$
  - $(\text{retrieve (?x) (\text{?x (or woman man))}})$
    $\Rightarrow (((\text{?x alice})) (((\text{?x betty})) (((\text{?x charles}))))$
Query Bodies

- Each query atom is a body
- If $b_i$ are query bodies, then also
  - $\lor b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n$
- Meaning of “or”
  - TBox: $\top \Rightarrow \text{woman} \sqsubseteq \text{man}$
  - (retrieve (?x) (?x (or woman man)))
    $\Rightarrow$ (((?x alice)) ((?x betty)) ((?x charles)))
  - (retrieve (?x) [or (?x woman) (?x man)])
    $\Rightarrow$ (((?x alice)) ((?x betty)))
Query Bodies

- Each query atom is a body
- If \( b_i \) are query bodies, then also
  - \( b_1, b_1 \land \cdots \land b_n, b_1 \lor \cdots \lor b_n \)
- Meaning of “or”
  - TBox: \( \top \Rightarrow \text{woman} \sqcup \text{man} \)
  - \((\text{retrieve} (\text{?x}) (\text{?x} (\text{or} \text{woman} \text{man})))\)
    \(\Rightarrow (((\text{?x alice}) ((\text{?x betty}) ((\text{?x charles}))))\)
  - \((\text{retrieve} (\text{?x}) (\boxed{\text{or}} (\text{?x woman})
                     (\text{?x man})))\)
    \(\Rightarrow (((\text{?x alice}) ((\text{?x betty})))\)
  - \((\text{concept-instances woman}) \cup
    (\text{concept-instances man})\)
    \(\subseteq (\text{concept-instances (or woman man)})\)
Arity of Query Bodies

- Arity of \((\text{and } (?x \ C) \ (\text{or } ?y \ D))\): 2
Arity of Query Bodies

- Arity of \((\text{and} \ (\ ?x \ C) \ (\ ?y \ D))\): 2
- Arity of \((\text{or} \ (\ ?x \ C) \ (\ ?y \ D))\): ?
Arity of Query Bodies

- Arity of \((\text{and} \ (?x \ C) \ (?y \ D))\): 2
- Arity of \((\text{or} \ (?x \ C) \ (?y \ D))\): ?
- We want the NNF:
Arity of Query Bodies

- Arity of $(\text{and} \ (?x \ C) \ (?y \ D))$: 2
- Arity of $(\text{or} \ (?x \ C) \ (?y \ D))$: ?
- We want the NNF:
  - $(\text{not} \ (\text{and} \ (\text{not} \ (?x \ C)) \ (\text{not} \ (?y \ D))))$
Arity of Query Bodies

- Arity of \((\text{and} \ (?x \ C) \ (?y \ D))\): 2
- Arity of \((\text{or} \ (?x \ C) \ (?y \ D))\): ?
- We want the NNF:
  - \((\text{not} \ (\text{and} \ (\text{not} \ (?x \ C)) \ (\text{not} \ (?y \ D)))))\)
  - NOT preserves arity
Arity of Query Bodies

- Arity of \((\text{and } (?x \ C) \ (\text{?y } D))\): 2
- Arity of \((\text{or } (?x \ C) \ (\text{?y } D))\): ?
- We want the NNF:
  - \((\text{not } (\text{and } (\text{not } (?x \ C)) (\text{not } (?y \ D))))\)
  - NOT preserves arity
  \(\Rightarrow\) Arity of \((\text{or } (?x \ C) \ (\text{?y } D))\): 2
Arity of Query Bodies

- Arity of \( \text{and} \ (\text{?x C}) \ (\text{?y D}) \): 2
- Arity of \( \text{or} \ (\text{?x C}) \ (\text{?y D}) \): ?
- We want the NNF:
  - \( \text{not} \ (\text{and} \ (\text{not} \ (\text{?x C})) \ (\text{not} \ (\text{?y D}))) \)
  - NOT preserves arity

\[ \Rightarrow \text{Arity of} \ (\text{or} \ (\text{?x C}) \ (\text{?y D})) : 2 \]
- Internally rewritten into

\[ (\text{or} \ (\text{and} \ (\text{?x C}) \ (\text{?y TOP})) \ (\text{?y D}) \ (\text{?x TOP})) \]
Arity of Query Bodies

- Arity of \((\text{and} \ (\ ?x \ C) \ (\ ?y \ D))\): 2
- Arity of \((\text{or} \ (\ ?x \ C) \ (\ ?y \ D))\): ?
- We want the NNF:
  - \((\text{not} \ (\text{and} \ (\text{not} \ (\ ?x \ C)) \ (\text{not} \ (\ ?y \ D))))\)
  - NOT preserves arity
  \[\Rightarrow\] Arity of \((\text{or} \ (\ ?x \ C) \ (\ ?y \ D))\): 2
- Internally rewritten into
  \[\begin{align*}
      \text{(or} & \ (\text{and} \ (\ ?x \ C) \ (\ ?y \ TOP)) \\
       \text{(} & \ (\ ?y \ D) \ (\ ?x \ TOP))
  \end{align*}\]
  \[\Rightarrow\] If \(C\) as well as \(D\) instances are present, then
Arity of Query Bodies

- Arity of \((\text{and} \ (\ ?x \ C) \ (\ ?y \ D))\): 2
- Arity of \((\text{or} \ (\ ?x \ C) \ (\ ?y \ D))\): ?
- We want the NNF:
  - \((\text{not} \ (\text{and} \ (\text{not} \ (\ ?x \ C)) \ (\text{not} \ (\ ?y \ D))))\)
  - NOT preserves arity
  \(\Rightarrow\) Arity of \((\text{or} \ (\ ?x \ C) \ (\ ?y \ D))\): 2
- Internally rewritten into
  \((\text{or} \ (\text{and} \ (\ ?x \ C) \ (\ ?y \ \text{TOP})) \ (\ ?y \ D) \ (\ ?x \ \text{TOP}))\)

\(\Rightarrow\) If \(C\) as well as \(D\) instances are present, then
\((\text{retrieve} \ (\ ?x) \ (\text{or} \ (\ ?x \ C) \ (\ ?y \ D)))\)
Arity of Query Bodies

- Arity of \((\text{and} \ (?x \ C) \ (?y \ D))\): 2
- Arity of \((\text{or} \ (?x \ C) \ (?y \ D))\): ?
- We want the NNF:
  - \((\text{not} \ (\text{and} \ (\text{not} \ (?x \ C)) \ (\text{not} \ (?y \ D))))\)
  - NOT preserves arity
  \[ \Rightarrow \text{Arity of} \ (\text{or} \ (?x \ C) \ (?y \ D))\): 2
- Internally rewritten into
  \((\text{or} \ (\text{and} \ (?x \ C) \ (?y \ \text{TOP})) \ (?y \ D) \ (?x \ \text{TOP}))\)
  \[ \Rightarrow \text{If} \ C \ \text{as well as} \ D \ \text{instances are present, then is equivalent to} \]
Arity of Query Bodies

- Arity of \((\text{and} \ (\text{?x} \ C) \ (\text{?y} \ D))\): 2
- Arity of \((\text{or} \ (\text{?x} \ C) \ (\text{?y} \ D))\): ?
- We want the NNF:
  - \((\text{not} \ (\text{and} \ (\text{not} \ (\text{?x} \ C)) \ (\text{not} \ (\text{?y} \ D))))\)
  - \text{NOT preserves arity}
  \[ \Rightarrow \text{Arity of} \ (\text{or} \ (\text{?x} \ C) \ (\text{?y} \ D)) \text{: 2} \]
- Internally rewritten into
  \(\ (\text{or} \ (\text{and} \ (\text{?x} \ C) \ (\text{?y TOP})) \ (\text{?y D} \ (\text{?x TOP})) \)
  \[ \Rightarrow \text{If} \ C \text{ as well as} \ D \text{ instances are present, then} \]
  \(\ (\text{retrieve} \ (\text{?x}) \ (\text{?x TOP})) \)
nRQL Query Processing Chain

- Replace syntactic sugar
nRQL Query Processing Chain

- Replace syntactic sugar
  - (?x NIL R)
  - \( \Rightarrow (\text{not} (\text{?x (:has-known-successor R)))) \)
nRQL Query Processing Chain

- Replace syntactic sugar
  - $(?x \text{ NIL } R)$  
    $\Rightarrow$ (not $(?x (:\text{has-known-successor } R)))$

- Break up feature chains

  $(?x \ ?y (:\text{constraint} \ (\text{has-father age}) \ (\text{has-mother age}) =))$
  
  $\Rightarrow$ (and $(?x \ ?x\text{-father has-father})$
  
  $(?y \ ?y\text{-mother has-mother})$
  
  $(?x\text{-father } ?y\text{-mother} \ (::\text{constraint age age age =}))$
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
  - (and (?x woman)
    (?x ?y has-child)
    (?x (not mother)))
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
  - \((\text{and} (\text{?x woman}) (\text{?x ?y has-child}) (\text{?x (not mother)}))\)
  \[\Rightarrow\text{ inconsistent}\]
  - \((\text{and} (\text{?x woman}) (\text{?y mother}) (\text{?x ?y has-child}))\)
  \[\Rightarrow (\text{and} (\text{?x grandmother}) (\text{?y mother}) (\text{?x ?y has-daughter}))\]
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
  - \( (\text{and } (?x \text{ woman}) (?y \text{ man})
    \quad (?x \ ?y \text{ has-\text{child}})) \)
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
  - \((\text{and } (?x \text{ woman}) (?y \text{ man})
    \text{ (?x ?y has-child))}\)
  - \(3! = 6\) orderings of conjuncts
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
  - \((\text{and} (\text{?x woman}) (\text{?y man}) (\text{?x ?y has-child}))\)
  - \(3! = 6\) orderings of conjuncts
    - \(\ominus (\text{?x woman}) (\text{?y man}) (\text{?x ?y has-child})\)
    - \(\oplus (\text{?x woman}) (\text{?x ?y has-child}) (\text{?y man})\)
    - \(\otimes (\text{?y man}) (\text{?y ?x child-of}) (\text{?x woman})\)
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
- Execution: find solutions of a finite domain CSP
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
- Execution: find solutions of a finite domain CSP
  - Compilation of a LISP program from query
  - Backtracking Search
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
- Execution: find solutions of a finite domain CSP
- Construction of result tuples from body bindings
nRQL Query Processing Chain

- Replace syntactic sugar
- Bring query into DNF
- Reasoning (incomplete in general, but useful!)
- Heuristic Optimization
- Execution: find solutions of a finite domain CSP
- Construction of result tuples from body bindings
  - Duplications and reorderings possible, e.g.
    
    \[
    \text{(retrieve (} ?x \ ?y \ ?x) \\
    \quad \text{(and (} ?y \ \ldots) \ (} ?x \ \ldots) \ \ldots))
    \]
Future Work

- Reasoning with queries
Future Work

- Reasoning with queries
  ⇒ Functions for Racer’s API
Future Work

• Reasoning with queries
  ⇒ Functions for Racer’s API
• Server Continuations
Future Work

- Reasoning with queries
  ⇒ Functions for Racer’s API
- Server Continuations
  - “Tuple at a time”
  - “Set at a time”
  ⇒ Functions / Switches for Racer’s API
Future Work

- Reasoning with queries
  - Functions for Racer’s API
- Server Continuations
  - “Tuple at a time”
  - “Set at a time”
  - Functions / Switches for Racer’s API
- Complex constraint expressions:

  (<?x ?y (:constraint
 fian (>= (- (father age) 10)
 (mother age))))
Future Work

• Reasoning with queries
  ⇒ Functions for Racer’s API

• Server Continuations
  • “Tuple at a time”
  • “Set at a time”
  ⇒ Functions / Switches for Racer’s API

• Complex constraint expressions:

  (?x ?y (:constraint
       (>= (- (father age) 10) (mother age)))))

• Enhance efficiency
Limitations of nRQL

- Consider the ABox $\mathcal{A} = \text{def } \{ (\exists R.C)(i) \}$
Limitations of nRQL

- Consider the ABox $\mathcal{A} = \text{def } \{(\exists R.C)(i)\}$
- Logic tells us $\mathcal{A} \models \exists x.C(x)$
Limitations of nRQL

- Consider the ABox $\mathcal{A} =_{df} \{(\exists R.C)(i)\}$
- Logic tells us $\mathcal{A} \models \exists x. C(x)$
- But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$
  in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$
Limitations of nRQL

• Consider the ABox $\mathcal{A} = \textit{def} \{ (\exists R.C)(i) \} $

• Logic tells us $\mathcal{A} \models \exists x.C(x)$

• But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$ in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$

• $(\text{retrieve (}?) (\_ C)) \Rightarrow \text{NIL}$
Limitations of nRQL

- Consider the ABox \( \mathcal{A} = \text{def} \ \{(\exists R.C)(i)\} \)
- Logic tells us \( \mathcal{A} \models \exists x.C(x) \)
- But there is no \( j \in \text{individuals}(\mathcal{A}) \) with \( j^\mathcal{I} \in C^\mathcal{I} \) in all \( (\Delta^\mathcal{I}, \cdot^\mathcal{I}) \) with \( (\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A} \)
- \( \text{retrieve}(\ ?x\ ) (\ ?x\ C) ) \Rightarrow \text{NIL} \)
- Reason: \( \text{(concept-instances C) } \Rightarrow \text{NIL} \)
Limitations of nRQL

- Consider the ABox \( \mathcal{A} \equiv \text{def} \ \{ (\exists R.C)(i) \} \)
- Logic tells us \( \mathcal{A} \models \exists x.C(x) \)
- But there is no \( j \in \text{individuals}(\mathcal{A}) \) with \( j^\mathcal{I} \in C^\mathcal{I} \) in all \( (\Delta^\mathcal{I}, \cdot^\mathcal{I}) \) with \( (\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A} \)
- \( (\text{retrieve} (\exists x) (\exists x C)) \Rightarrow \text{NIL} \)
- Reason: \( (\text{concept-instances} C) \Rightarrow \text{NIL} \)

\( (\text{retrieve} (\exists x) (\exists x C)) \)

\( \Rightarrow ((\exists x \text{ temp123})) \) better?
Limitations of nRQL

- Consider the ABox $\mathcal{A} = \text{def } \{(\exists R.C)(i)\}$
- Logic tells us $\mathcal{A} \models \exists x.C(x)$
- But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$ in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$
- $\text{(retrieve } (?x) \text{ (?x C)) } \Rightarrow \text{NIL}$
- Reason: $(\text{concept-instances C}) \Rightarrow \text{NIL}$
- $(\text{retrieve } (?x) \text{ (?x C))}$
  $\Rightarrow (((?x \text{ (and C (some (inv R)) \text{ (one-of i))})))))$

better?
Limitations of nRQL

- Consider the ABox $\mathcal{A} = \text{def} \{ (\exists R.C)(i) \}$
- Logic tells us $\mathcal{A} \models \exists x. C(x)$
- But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$ in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$
- $(\text{retrieve} \ (?x) \ (?x \ C)) \Rightarrow \text{NIL}$
- Reason: $(\text{concept-instances} \ C) \Rightarrow \text{NIL}$
- $(\text{retrieve} \ (?x \ ?y) \ (?x \ ?y \ R)) \Rightarrow \text{NIL}$
Limitations of nRQL

- Consider the ABox $\mathcal{A} \equiv \{ (\exists R.C)(i) \}$
- Logic tells us $\mathcal{A} \models \exists x. C(x)$
- But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$ in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$
- (retrieve (?x) (?x C)) $\Rightarrow$ NIL
- Reason: (concept-instances C) $\Rightarrow$ NIL
- (retrieve (?x ?y) (?x ?y R)) $\Rightarrow$ NIL
- Reason: (related-individuals R) $\Rightarrow$ NIL
Limitations of nRQL

- Consider the ABox $\mathcal{A} =_{def} \{(\exists R.C)(i)\}$
- Logic tells us $\mathcal{A} \models \exists x.C(x)$
- But there is no $j \in \text{individuals}(\mathcal{A})$ with $j^\mathcal{I} \in C^\mathcal{I}$ in all $(\Delta^\mathcal{I}, \cdot^\mathcal{I})$ with $(\Delta^\mathcal{I}, \cdot^\mathcal{I}) \models \mathcal{A}$

- $(\text{retrieve } (?x) (?x \ C)) \Rightarrow \text{NIL}$
- Reason: $(\text{concept-instances } C) \Rightarrow \text{NIL}$
- $(\text{retrieve } (?x \ ?y) (?x \ ?y \ R)) \Rightarrow \text{NIL}$
- Reason: $(\text{related-individuals } R) \Rightarrow \text{NIL}$
- $(\text{retrieve } (?x) (?x \ (\text{some } R \ C)))$
  $\Rightarrow (((?x \ i)))$
Summing up, nRQL …

- … is a pragmatic ABox query language
Summing up, nRQL …

• … is a pragmatic ABox query language
• … has been requested by users
Summing up, nRQL …

- …is a pragmatic ABox query language
- …has been requested by users
- …frees them from programming loops
Summing up, nRQL …

• …is a pragmatic ABox query language
• …has been requested by users
• …frees them from programming loops
• …has been inspired by LOOM
Summing up, nRQL …

• …is a pragmatic ABox query language
• …has been requested by users
• …frees them from programming loops
• …has been inspired by LOOM
• …is easy to understand (one kind of variables)
Summing up, nRQL …

• …is a pragmatic ABox query language
• …has been requested by users
• …frees them from programming loops
• …has been inspired by LOOM
• …is easy to understand (one kind of variables)
• …has a formal semantics
Summing up, nRQL …

- …is a pragmatic ABox query language
- …has been requested by users
- …frees them from programming loops
- …has been inspired by LOOM
- …is easy to understand (one kind of variables)
- …has a formal semantics
- …which it inherits from the semantics of standard DL ABox retrieval functions
Summing up, nRQL …

- … is a pragmatic ABox query language
- … has been requested by users
- … frees them from programming loops
- … has been inspired by LOOM
- … is easy to understand (one kind of variables)
- … has a formal semantics
- … which it inherits from the semantics of standard DL ABox retrieval functions
- … combines in a uniform declarative way most of Racer’s ABox querying functions
Summing up, nRQL …

• …is a pragmatic ABox query language
• …has been requested by users
• …frees them from programming loops
• …has been inspired by LOOM
• …is easy to understand (one kind of variables)
• …has a formal semantics
• …which it inherits from the semantics of standard DL ABox retrieval functions
• …combines in a uniform declarative way most of Racer’s ABox querying functions
• …is efficiently implemented
Summing up, nRQL ...

- ...is a pragmatic ABox query language
- ...has been requested by users
- ...frees them from programming loops
- ...has been inspired by LOOM
- ...is easy to understand (one kind of variables)
- ...has a formal semantics
- ...which it inherits from the semantics of standard DL ABox retrieval functions
- ...combines in a uniform declarative way most of Racer’s ABox querying functions
- ...is efficiently implemented
- ...comes with a manual
Thank you!